



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

Documentation Changes

March 2026

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

Document Number: 252046-083



Notices & Disclaimers

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

No product or component can be absolutely secure.

Your costs and results may vary.

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

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

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

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

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

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

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

Contents

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

Revision History

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

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

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

Revision	Description	Date
-072	- Removed Documentation Changes 1-23 - Add Documentation Changes 1-19	June 2023
-073	- Removed Documentation Changes 1-19 - Add Documentation Changes 1-19	September 2023
-074	- Removed Documentation Changes 1-19 - Add Documentation Changes 1-20	December 2023
-075	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-20	March 2024
-076	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-8	June 2024
-077	- Removed Documentation Changes 1-8 - Add Documentation Changes 1-27	October 2024
-078	- Removed Documentation Changes 1-27 - Add Documentation Changes 1-15	December 2024
-079	- Removed Documentation Changes 1-15 - Add Documentation Changes 1-17	March 2025
-080	- Removed Documentation Changes 1-17 - Add Documentation Changes 1-25	June 2025
-081	- Removed Documentation Changes 1-25 - Add Documentation Changes 1-33	October 2025
-082	- Removed Documentation Changes 1-33 - Added Documentation Changes 1-32	January 2026
-083	- Removed Documentation Changes 1-32 - Added Documentation Changes 1-18	March 2026

§



Preface

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

Affected Documents

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

Nomenclature

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

Summary Tables of Changes

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

Codes Used in Summary Tables

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

Documentation Changes

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 1, Volume 1
2	Updates to Chapter 21, Volume 1
3	Updates to Chapter 4, Volume 2B
4	Updates to Chapter 2, Volume 3A
5	Updates to Chapter 4, Volume 3A
6	Updates to Chapter 8, Volume 3A
7	Updates to Chapter 10, Volume 3A
8	Updates to Chapter 17, Volume 3B
9	Updates to Chapter 20, Volume 3B
10	Updates to Chapter 22, Volume 3B
11	Updates to Chapter 27, Volume 3C
12	Updates to Chapter 28, Volume 3B
13	Updates to Chapter 29, Volume 3C
14	Updates to Chapter 30, Volume 3C
15	Updates to Chapter 31, Volume 3C
16	Updates to Chapter 32, Volume 3C
17	Updates to Chapter 41, Volume 3D
18	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 violet 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:

- In Section 1.4, references relating to the VERW instruction (Microarchitectural Data Sampling reference) and software security guidance (Hardware Features and Behavior Related to Speculative Execution reference) were added.

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® Scalable Processor 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® Scalable Processor Family

- 10th generation Intel® Core™ processors
- 11th generation Intel® Core™ processors
- 3rd generation Intel® Xeon® Scalable Processor Family
- 12th generation Intel® Core™ processors
- 13th generation Intel® Core™ processors
- 4th generation Intel® Xeon® Scalable Processor Family
- 5th generation Intel® Xeon® Scalable Processor Family
- Intel® Core™ Ultra 7 processors
- Intel® Xeon® 6 E-Core processors
- Intel® Xeon® 6 P-Core processors
- Intel® Series 2 Core™ Ultra 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® Scalable Processor 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® Scalable Processor 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® Scalable Processor 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.

The 13th generation Intel® Core™ processors are based on the Raptor Lake performance hybrid architecture and support Intel 64 architecture.

The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture and supports Intel 64 architecture.

The 5th generation Intel® Xeon® Scalable Processor Family is based on Emerald Rapids microarchitecture and supports Intel 64 architecture.

The Intel® Core™ Ultra 7 processor is based on Meteor Lake performance hybrid architecture and supports Intel 64 architecture.

The Intel® Xeon® 6 E-core processor is based on Sierra Forest microarchitecture and supports Intel 64 architecture.

The Intel® Xeon® 6 P-core processor is based on Granite Rapids microarchitecture and supports Intel 64 architecture.

The Intel® Series 2 Core™ Ultra processor is based on Lunar Lake performance hybrid architecture and supports 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 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 Intel® AVX, FMA, and Intel® 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® AVX10. Provides an overview of the Intel® AVX10 instruction set extensions and gives guidelines for writing code that access these extensions.

- Chapter 17 — 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 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 — Programming with Intel® Advanced Matrix Extensions.** Provides an overview of the Intel® Advanced Matrix Extensions and gives guidelines for writing code that access these extensions.
- Chapter 20 — Input/Output.** Describes the processor’s I/O mechanism, including I/O port addressing, I/O instructions, and I/O protection mechanisms.
- Chapter 21 — 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.
- Appendix E — Intel® Memory Protection Extensions.** Provides an overview of the Intel® Memory Protection Extensions, a feature that has been deprecated and will not be available on future processors.

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.

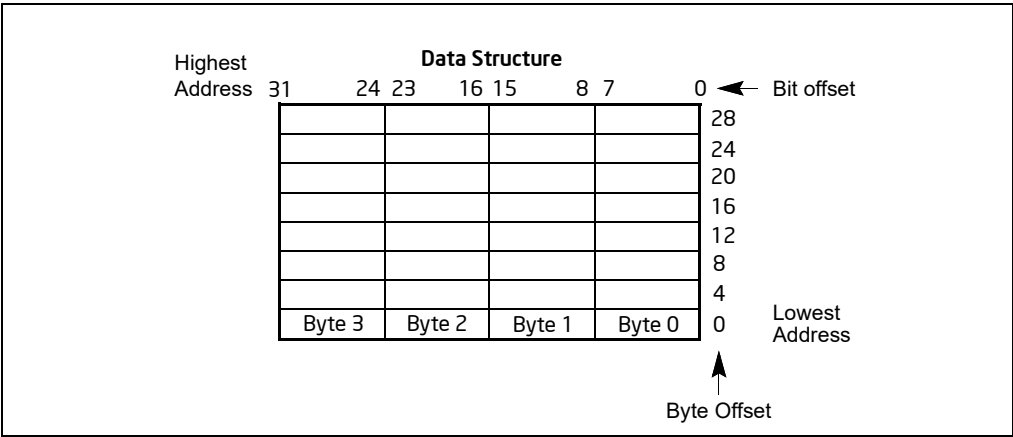


Figure 1-1. Bit and Byte Order

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 to 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 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. See Figure 1-2 for details on the syntax that represents this information.

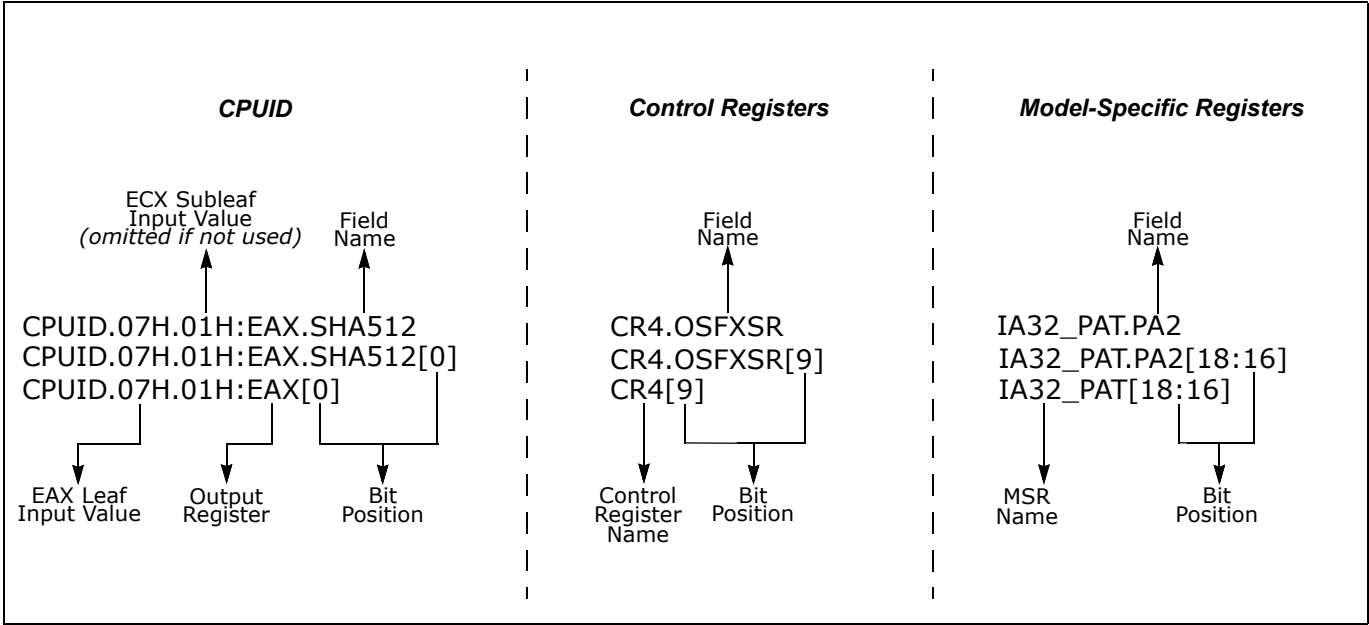


Figure 1-2. Syntax for CPUID, CR, and MSR Data Presentation

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://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm/intel64-and-ia32-architectures-optimization.html>
- Intel® Trusted Execution Technology Measured Launched Environment Developer's 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>

ABOUT THIS MANUAL

- 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>
- • **Microarchitectural Data Sampling**
<https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/intel-analysis-microarchitectural-data-sampling.html>
- • **Hardware Features and Behavior Related to Speculative Execution**
<https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/hardware-behavior-related-to-speculative-execution.html>

2. Updates to Chapter 21, Volume 1

Change bars and violet text show changes to Chapter 21 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- The following CPUID leaves and subleaves were added:
 - CPUID.10H.05H -- Cache Bandwidth Allocation
 - CPUID.10H.06H -- Resource Priority Control
 - CPUID.28H.05H -- Asymmetric Cache Bandwidth Allocation
 - CPUID.28H.06H -- Resource Priority Control
- The following CPUID bits were updated:
 - CPUID.07H.01H:EDX Extended Feature Information. Added EDX.AMX_COMPLEX[8].
 - CPUID.10H.00H -- Intel® RDT Allocation Main Sub-Leaf. In register EBX, added EBX.CBA[5] for Cache Bandwidth Allocation.
 - CPUID.10H.00H -- Intel® RDT Allocation Main Sub-Leaf. In register EBX, added EBX.RESOURCE_PRIORITY[6].
 - CPUID.28H.00H -- Intel® RDT Asymmetric Allocation Main Sub-Leaf. In register EBX, added EBX.CBA[5] for Cache Bandwidth Allocation.
 - CPUID.28H.00H -- Intel® RDT Asymmetric Allocation Main Sub-Leaf. In register EBX, added EBX.RESOURCE_PRIORITY[6].
- The page size in the description for CPUID Leaf 2 encoding was updated to 2 MByte/4 MByte.

CHAPTER 21

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

When writing software intended to run on Intel processors, it is necessary to identify the type of processor present in a system and the processor features that are available to an application.

The CPUID instruction, known as CPU Identification, was introduced with the Intel® Pentium processor to query the processor's information name space for its identity and supported features. Logically, the CPUID name space comprises a series of nodes indexed by leaf (using the input value of EAX) and in some cases further indexed by sub-leaf (using the input value of ECX). The value of a queried node is returned in EAX, EBX, ECX, and EDX. Note that not all leaves have sub-leaf indexing and the input ECX value will be ignored in those cases. The full description of CPUID can be found in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

All references to "MAX_LEAF" throughout this chapter are used as an abbreviation of "CPUID.00H:EAX.MAX_LEAF".

21.1 IMPORTANT CONSIDERATIONS WHEN USING THE CPUID INSTRUCTION

This section outlines additional factors to consider when using the CPUID instruction.

21.1.1 Guidelines for Using the CPUID Instruction

Use the CPUID instruction for processor identification in the Pentium M processor family, Pentium 4 processor family, Intel Xeon processor family, P6 family, Pentium processor, and later Intel486 processors. This instruction returns the family, model, and (for some processors) a brand string for the processor that executes the instruction. It also indicates the features that are present in the processor and gives information about the processor's caches and TLB.

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction. The CPUID instruction will cause the invalid opcode exception (#UD) if executed on a processor that does not support it.

To obtain processor identification information, a source operand value is placed in the EAX register to select the type of information to be returned. When the CPUID instruction is executed, selected information is returned in the EAX, EBX, ECX, and EDX registers.

The following guidelines are among the most important and should always be followed when using the CPUID instruction to determine available features:

- Always begin by testing for the "GenuineIntel," message in the EBX, EDX, and ECX registers when the CPUID instruction is executed with EAX equal to 0. If the processor is not genuine Intel, the feature identification flags may have different meanings than are described in Intel documentation.
- Test feature identification flags individually and do not make assumptions about undefined bits.

21.1.2 Identification of Earlier Processors

The CPUID instruction is not available in earlier Intel processors up through the earlier Intel 486 processors. For these processors, several other architectural features can be exploited to identify the processor.

The settings of bits 12 and 13 (IOPL), 14 (NT), and 15 (reserved) in the EFLAGS register are different for Intel's 32-bit processors than for the Intel 8086 and Intel 286 processors. By examining the settings of these bits (with the PUSHF/PUSHFD and POPF/POPFD instructions), an application program can determine whether the processor is an 8086, Intel 286, or one of the Intel 32-bit processors:

- 8086 processor — Bits 12 through 15 of the EFLAGS register are always set.
- Intel 286 processor — Bits 12 through 15 are always clear in real-address mode.

- 32-bit processors — In real-address mode, bit 15 is always clear and bits 12 through 14 have the last value loaded into them. In protected mode, bit 15 is always clear, bit 14 has the last value loaded into it, and the IOPL bits depend on the current privilege level (CPL). The IOPL field can be changed only if the CPL is 0.

Other EFLAGS register bits that can be used to differentiate between the 32-bit processors:

- Bit 18 (AC) — Implemented only on the Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors. The inability to set or clear this bit distinguishes an Intel386 processor from the later IA-32 processors.
- Bit 21 (ID) — Determines if the processor is able to execute the CPUID instruction. The ability to set and clear this bit indicates that it is a Pentium 4, Intel Xeon, P6 family, Pentium, or later-version Intel486 processor.

To determine whether an x87 FPU or Numeric Processor Extension (NPX) is present in a system, applications can write to the x87 FPU status and control registers using the FNINIT instruction and then verify that the correct values are read back using the FNSTENV instruction.

After determining that an x87 FPU or NPX is present, its type can then be determined. In most cases, the processor type will determine the type of FPU or NPX; however, an Intel386 processor is compatible with either an Intel 287 or Intel 387 math coprocessor.

The method the coprocessor uses to represent ∞ (after the execution of the FINIT, FNINIT, or RESET instruction) indicates which coprocessor is present. The Intel 287 math coprocessor uses the same bit representation for $+\infty$ and $-\infty$; whereas, the Intel 387 math coprocessor uses different representations for $+\infty$ and $-\infty$.

21.1.3 CPUID Basic and Extended Range

The CPUID basic range starts at CPUID.00H and ends at the maximum leaf enumerated in CPUID.00H:EAX.MAX_LEAF[31:0].

The legacy set of CPUID leaves are defined as leaves 00H, 01H, and 02H, which represent the architecture up to and including Pentium II. Processors provided legacy compatibility by limiting the exposed number of leaves to just these legacy leaves by setting IA32_MISC_ENABLE[22] (Limit CPUID Maxval). This is no longer supported on processors that report CPUID.07H.01H:EBX.CPUIDMAXVAL_LIM_RMV[3] as 1; for such processors, IA32_MISC_ENABLE[22] cannot be set to 1 to limit the value returned by CPUID.00H:EAX.MAX_LEAF.

The extended CPUID range starts at leaf 80000000H and ends at the maximum leaf enumerated in CPUID.80000000H:EAX.MAX_EXTENDED_LEAF[31:0].

Older processors before the Pentium 4 do not support the extended CPUID range and treat bit 31 of CPUID's input EAX value as zero.

If a value entered for CPUID.EAX is higher than the maximum input value for basic or extended function for that processor then the data for the highest basic information leaf is returned. Software should not rely on the values returned by the processor outside of the above ranges.

The range CPUID.40000000H to CPUID.4FFFFFFFH do not return feature information for the processor. These are allocated for emulation by software.

21.1.4 CPUID Domains

The fields of each CPUID node are classified into one of several CPUID domains. The fields may be classified separately within a specific node or in aggregate for all the nodes in a leaf or sub-leaf. On a properly configured platform, all logical processors within a CPUID domain return a consistent output value for fields belonging to that domain. As an example, the initial X2APIC ID value returned in CPUID.1FH.00H:EDX[31:0] is classified as being in the Logical Processor Domain because the value is unique for each logical processor in the platform. Whereas, the CLFLUSH Line Size returned in CPUID.00H:EBX[15:8] is classified as Platform Domain because it must be consistent for all logical processor within the entire platform.

- **Platform Domain**—A properly configured platform would provide consistent values for these CPUID fields for each logical processor in the platform.
- **Package Domain**—A properly configured platform provides consistent values for these CPUID fields for each logical processor within the same processor package. These values however can be different when comparing the values of logical processors on different packages.

- **Logical Processor Domain**—A properly configured platform can provide different values for these CPUID fields for each logical processor in the platform. The values contained within these may have their own scope as per a specific shared resource (i.e., cache, hybrid, etc.); in that case, each logical processor may need to be queried to obtain the full platform view of given features.

21.1.5 CPUID Runtime Mutable Fields

A CPUID field is said to be mutable if it can change during runtime. Such fields are affected by supervisor-mode operations that can affect processor mode, status bits, or privileged registers. Mutable fields that are expected to change dynamically as part of normal operation are shown in the table Table 21-1. Mutable fields that should remain consistent are shown in the table Table 21-2. Note all of the listed controls may not be available on all Intel processors.

Table 21-1. Runtime Mutable CPUID Fields Expected to Change During Normal Operation

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
01H	Ignored	ECX[27]	OSXSAVE	If 1, the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCRO and to support processor extended state management using XSAVE/XRSTOR.
07H	00H	ECX[4]	OSPKE	If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions).
0DH	00H	EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURE	The size of the XSAVE/XRSTOR area required for the state bits enabled in XCRO.
0DH	01H	EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURE	The size of the XSAVES/XRSTORS area required for the state bits enabled in XCRO and IA32_XSS.
19H	00H	EBX[0]	AESKLE	If 1, if the AES Key Locker instructions have been activated by system firmware and the OS has set CR4.KL[bit 19] = 1.
80000001H	Ignored	EDX[20]	SYSCALL_SYSRET_64	Intel processors support SYSCALL and SYSRET only in 64-bit mode. This feature flag is always enumerated as 0 outside 64-bit mode.

Table 21-2. Runtime Mutable CPUID Fields That Should Remain Consistent

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
00H	Ignored	EAX[31:0]	MAX_LEAF	Support for legacy software by limiting CPUID number of leaves reporting to a maximum of 2. This is set by using IA32_MISC_ENABLE[22] Limit CPUID Maxval.
01H	Ignored	ECX[3]	MONITOR	This feature flag reflects the setting in IA32_MISC_ENABLE[18] Enable Monitor FSM.
01H	Ignored	ECX[7]	EIST	This feature flag reflects the setting in IA32_MISC_ENABLE[16] Enhanced Intel SpeedStep Technology Enable.
01H	Ignored	EDX[9]	APIC	This feature flag reflects IA32_APIC_BASE[11], APIC Global Enable.
05H	Ignored	ECX[0]	MONITOR_MWAIT_EXTENSIONS	This field not available when CPUID.01H:ECX.MONITOR[3] = 0.

Table 21-2. Runtime Mutable CUID Fields That Should Remain Consistent (Contd.)

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
05H	Ignored	ECX[1]	INTERRUPT_AS_BREAK_EVENT	This field not available when CUID.01H:ECX.MONITOR[3] = 0.
05H	Ignored	EDX[15:0]	LARGEST_MONITOR_LINE_SIZE	This field not available when CUID.01H:ECX.MONITOR[3] = 0.
05H	Ignored	EAX[15:0]	SMALLEST_MONITOR_LINE_SIZE	This field not available when CUID.01H:ECX.MONITOR[3] = 0.

21.1.6 CUID Reserved Fields

Software must ignore and not rely upon the values returned by reserved fields of a CUID leaf or sub-leaf because they may have meaning on future processors. Once a previously-reserved field becomes defined, this specification will be updated to reflect that.

21.1.7 CUID Instruction for Serialization

Although the CUID instruction provides serialization, it is not the preferred method on newer processors that support the SERIALIZE instruction, which is enumerated via CUID.07H.00H:EDX[14]=1. If backward compatibility is required with older processors, use leaf 00H [CUID.00H] for serialization because it has the lowest latency when executed. See “Serializing Instructions” in Chapter 11 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A for more details.

21.1.8 IA32_BIOS_SIGN_ID Returns Microcode Update Signature

For processors that support the microcode update facility, the IA32_BIOS_SIGN_ID MSR is loaded with the update signature whenever CUID executes. The signature is returned in the upper DWORD. For details, see Chapter 11 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

21.2 METHODS FOR RETURNING BRANDING INFORMATION USING CUID

Use the following techniques to access branding information:

1. Processor brand string method.
2. Processor brand index; this method uses a software supplied brand string table.

These two methods are discussed in the following sections. For methods that are available in early processors, see Section 21.1.2, “Identification of Earlier Processors,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

21.2.1 The Processor Brand String Method

Figure 21-1 describes the algorithm used for detection of the brand string. Processor brand identification software should execute this algorithm on all Intel 64 and IA-32 processors.

This method (introduced with Pentium 4 processors) returns an ASCII brand identification string and the Processor Base frequency of the processor to the EAX, EBX, ECX, and EDX registers.

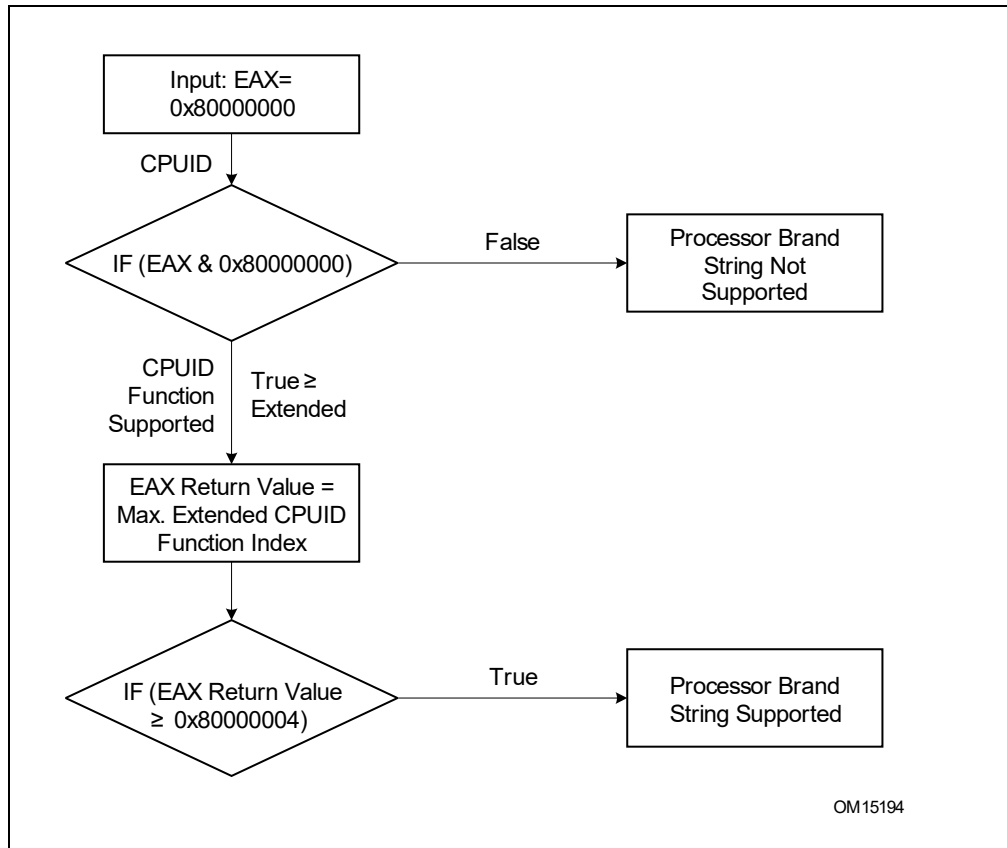


Figure 21-1. Determination of Support for the Processor Brand String

21.2.2 The Processor Brand Index Method

The brand index method (introduced with Pentium® III Xeon® processors) provides an entry point into a brand identification table that is maintained in memory by software. In this table, each brand index is associated with an ASCII brand identification string that identifies the official Intel family and model number of a processor.

When CPUID executes with EAX set to 1, the processor returns a brand index to the low byte in EBX. Software can then use this index to locate the brand identification string for the processor in the brand identification table. The first entry (brand index 0) in this table is reserved, allowing for backward compatibility with processors that do not support the brand identification feature. Starting with processor signature family ID = 0FH, model = 03H, brand index method is no longer supported. Use brand string method instead.

Table 21-3 shows brand indices that have identification strings associated with them.

Table 21-3. Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

Brand Index	Brand String
00H	This processor does not support the brand identification feature
01H	Intel® Celeron® processor ¹
02H	Intel® Pentium® III processor ¹
03H	Intel® Pentium® III Xeon® processor; If processor signature = 000006B1h, then Intel® Celeron® processor
04H	Intel® Pentium® III processor
06H	Mobile Intel® Pentium® III processor-M
07H	Mobile Intel® Celeron® processor ¹

Table 21-3. Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

08H	Intel® Pentium® 4 processor
09H	Intel® Pentium® 4 processor
0AH	Intel® Celeron® processor ¹
0BH	Intel® Xeon® processor; If processor signature = 00000F13h, then Intel® Xeon® processor MP
0CH	Intel® Xeon® processor MP
0EH	Mobile Intel® Pentium® 4 processor-M; If processor signature = 00000F13h, then Intel® Xeon® processor
0FH	Mobile Intel® Celeron® processor ¹
11H	Mobile Genuine Intel® processor
12H	Intel® Celeron® M processor
13H	Mobile Intel® Celeron® processor ¹
14H	Intel® Celeron® processor
15H	Mobile Genuine Intel® processor
16H	Intel® Pentium® M processor
17H	Mobile Intel® Celeron® processor ¹
18H – 0FFH	RESERVED

NOTES:

1. Indicates versions of these processors that were introduced after the Pentium III.

21.3 CPUID LEAVES

The remainder of this chapter provides CPUID enumeration information for Intel® 64 and IA-32 architectures.

CPUID.00H -- Maximum Input for Basic CPUID and Vendor ID

CPUID.00H returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register and is processor specific.

- This leaf is always valid.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-4. Leaf 00H Maximum Input for Basic CPUID and Vendor ID

Register	Field Name	Description	Domain
EAX[31:0]	MAX_LEAF	Maximum input value for basic CPUID Information.	Platform
EBX[31:0]	VENDOR_ID_1	"Genu"	Platform
ECX[31:0]	VENDOR_ID_2	"ntel"	Platform
EDX[31:0]	VENDOR_ID_3	"inel"	Platform

A vendor identification string is also returned in EBX, EDX, and ECX. For Intel processors, the string is "GenuineIntel" and is expressed:

EBX := 756e6547h (* "Genu", with G in the low eight bits of BL *)

EDX := 49656e69h (* "ineI", with i in the low eight bits of DL *)

ECX := 6c65746eh (* "ntel", with n in the low eight bits of CL *)

CPUID.01H -- Version and Features

CPUID.01H returns type, family, model, stepping, and feature information.

- This leaf is valid if MAX_LEAF $\geq$ 01H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-5. Leaf 01H Output Registers

CPUID Output Registers	Description
EAX[31:0]	Version information: Type, Family, Model, and Stepping ID (see “CPUID.01H:EAX—Version Information: Type, Family, Model and Stepping ID”).
EBX[31:0]	Feature information (see “CPUID.01H:EBX—Feature Information”).
ECX[31:0]	Feature information (see “CPUID.01H:ECX—Feature Information”).
EDX[31:0]	Feature information (see “CPUID.01H:EDX—Feature Information”).

CPUID.01H:EAX Version Information: Type, Family, Model and Stepping ID

The EAX register of CPUID.01H returns the information shown below.

Table 21-6. Leaf 01H Version and Features Returned in EAX

Register	Field Name	Description	Domain
EAX[3:0]	STEPPING_ID	Identifies a revision of the specific processor family and model. The stepping information is specified as a per-package basis for legacy processors. More recent processors do not allow mixing steppings.	Package
EAX[7:4]	MODEL_ID	Identifies a set of processors within a family. Certain models of Pentium® 4 processors allowed mixed Model IDs and would have this identified as a Package Domain.	Platform
EAX[11:8]	FAMILY_ID	Identifies a set of processors that have a general architectural similarity.	Platform
EAX[13:12]	PROCESSOR_TYPE	Identifies specific type of processor.	Platform
EAX[15:14]	Reserved	Reserved.	
EAX[19:16]	EXTENDED_MODEL_ID	When the Family ID is 06H or 0FH, this field is prepended to the Model ID to provide an 8-bit model identification.	Platform
EAX[27:20]	EXTENDED_FAMILY_ID	When the Family ID is 0FH, this field is added to the Family ID to provide an 8-bit family identification.	Platform
EAX[31:28]	Reserved	Reserved.	

CPUID.01H returns version information in EAX. For example: model, family, and processor type for the Intel Xeon processor 5100 series is as follows:

- Model — 1111B
- Family — 0101B
- Processor Type — 00B

See table below for available processor type values. Stepping IDs are provided as needed.

Table 21-7. Processor Type Field

Type	Encoding
Original OEM Processor	00B
Intel OverDrive® Processor	01B
Dual processor (not applicable to Intel486 processors)	10B
Intel reserved	11B

NOTE

See Section 21.1.2, “Identification of Earlier Processors,” for information on identifying earlier IA-32 processors. The Extended Family ID needs to be examined only when the Family ID is 0FH. Integrate the fields into a display using the following rule:

```

IF Family_ID ≠ 0FH THEN
    DisplayFamily = Family_ID;
ELSE
    DisplayFamily = Extended_Family_ID + Family_ID;
FI;

```

The Extended Model ID needs to be examined only when the Family ID is 06H or 0FH. Integrate the field into a display using the following rule:

```

IF (Family_ID = 06H or Family_ID = 0FH) THEN
    DisplayModel = (Extended_Model_ID « 4) + Model_ID;
ELSE
    DisplayModel = Model_ID;
FI;

```

CPUID.01H:EBX Feature information.

The EBX register of CPUID.01H returns the information shown below.

Table 21-8. Leaf 01H Version and Features Returned in EBX

Register	Field Name	Description	Domain
EBX[7:0]	BRAND_INDEX	This number provides an entry into a brand string table that contains brand strings for IA-32 processors. More information about this field is provided in Section 21.2.2, “The Processor Brand Index Method.”	Platform
EBX[15:8]	CLFLUSH_LINE_SIZE	Value * 8 = cache line size in bytes. This number indicates the size of the cache line flushed by the CLFLUSH and CLFLUSHOPT instructions in 8-byte increments. This field was introduced in the Pentium 4 processor.	Platform
EBX[23:16]	APIC_ID_SPACE	Maximum number of addressable IDs for logical processors in this physical package. The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. This field is only valid if CPUID.01H.EDX.HTT[28]= 1. See further details below on the usage of this field.	Platform

EBX[31:24]	INITIAL_APIC_ID	This number is the 8-bit ID that is assigned to the local APIC on the processor during power up. This field was introduced in the Pentium 4 processor. The 8-bit initial APIC ID in EBX[31:24] is replaced by the 32-bit x2APIC ID, available in Leaf 0BH and Leaf 1FH.	Logical Processor

The Maximum Addressable IDs for logical processors in this package should not be used on platforms that support CPUID leaf 0BH or CPUID leaf 1FH as it can be saturated and incorrect. Modern platforms can have many more processors than can be enumerated or have topology domains with discontinuous APIC ID reservations. To correctly enumerate APIC ID information on modern platforms, use CPUID.0BH or CPUID.1FH.

CPUID.01H:ECX Feature Information

The ECX register of CPUID.01H returns the information shown below. For all feature flags, a 1 indicates that the feature is supported. Software should identify Intel as the vendor to properly interpret feature flags. Software must confirm that a processor feature is present using feature flags returned by CPUID prior to using the feature. Software should not depend on future offerings retaining all features.

Table 21-9. Leaf 01H Version and Features Returned in ECX

Register	Field Name	Description	Domain
ECX[0]	SSE3	If 1, supports Streaming SIMD Extensions 3.	Platform
ECX[1]	PCLMULQDQ	If 1, supports the PCLMULQDQ instruction.	Platform
ECX[2]	DTES64	64-bit DS Area. If 1, supports DS area using 64-bit layout.	Platform
ECX[3]	MONITOR	If 1, supports the MONITOR/MWAIT and CPUID.05H.	Platform
ECX[4]	DS_CPL	If 1, supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.	Platform
ECX[5]	VMX	If 1, supports the Virtual Machine Extensions.	Platform
ECX[6]	SMX	If 1, supports Safer Mode Extensions. See Chapter 7, "Safer Mode Extensions Reference."	Platform
ECX[7]	EIST	If 1, supports Enhanced Intel SpeedStep® technology.	Platform
ECX[8]	TM2	If 1, supports Thermal Monitor 2.	Platform
ECX[9]	SSSE3	If 1, supports Supplemental Streaming SIMD Extensions 3.	Platform
ECX[10]	L1_CONTEXT_ID	If 1, the L1 data cache mode can be set to either adaptive mode or shared mode. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.	Platform
ECX[11]	DEBUG_INTERFACE	If 1, supports IA32_DEBUG_INTERFACE MSR for silicon debug.	Platform
ECX[12]	FMA	If 1, supports FMA extensions using YMM state.	Platform

ECX[13]	CMPXCHG16B	If 1, supports this instruction. See the “CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes” section in this chapter for a description.	Platform
ECX[14]	XTPR_UPDATE_CONTROL	If 1, supports changing IA32_MISC_ENABLE[bit 23].	Platform
ECX[15]	PERF_CAPABILITIES	If 1, supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.	Platform
ECX[16]	Reserved	Reserved.	
ECX[17]	PCID	If 1, supports Process-context identifiers and software setting CR4.PCIDE to 1.Process-context identifiers.	Platform
ECX[18]	DCA	If 1, supports the ability to prefetch data from a memory mapped device. See CPUID.09H.	Platform
ECX[19]	SSE4_1	If 1, supports SSE4.1.	Platform
ECX[20]	SSE4_2	If 1, supports SSE4.2.	Platform
ECX[21]	X2APIC	If 1, supports x2APIC feature.	Platform
ECX[22]	MOVBE	If 1, supports MOVBE instruction.	Platform
ECX[23]	POPCNT	If 1, supports the POPCNT instruction.	Platform
ECX[24]	TSC_DEADLINE	If 1, the processor’s local APIC timer supports one-shot operation using a TSC deadline value.	Platform
ECX[25]	AESNI	If 1, supports the AESNI instruction extensions.	Platform
ECX[26]	XSAVE	If 1, supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCRO.	Platform
ECX[27]	OSXSAVE	If 1, the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCRO and to support processor extended state management using XSAVE/XRSTOR.	Logical Processor
ECX[28]	AVX	If 1, supports the AVX instruction extensions.	Platform
ECX[29]	F16C	If 1, supports 16-bit floating-point conversion instructions.	Platform
ECX[30]	RDRAND	If 1, supports RDRAND instruction.	Platform
ECX[31]	Not Used	Intel processors always return 0. Allocated for use by software emulation.	Platform

CPUID.01H:EDX Feature Information

The EDX register of CPUID.01H returns the information shown below. For all feature flags, a 1 indicates that the feature is supported. Software should identify Intel as the vendor to properly interpret feature flags. Software must confirm that a processor feature is present using feature flags returned by CPUID prior to using the feature. Software should not depend on future offerings retaining all features.

Table 21-10. Leaf 01H Version and Features Returned in EDX

Register	Field Name	Description	Domain
EDX[0]	FPU	Floating Point Unit On-Chip. The processor contains an x87 FPU.	Platform

EDX[1]	VME	If 1, supports Virtual 8086 mode enhancements, including CR4.VME for controlling the feature; CR4.PVI for protected mode virtual interrupts; software interrupt indirection; expansion of the TSS with the software indirection bitmap; and EFLAGS.VIF and EFLAGS.VIP flags.	Platform
EDX[2]	DE	If 1, supports I/O breakpoints debugging extensions, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.	Platform
EDX[3]	PSE	P If 1, supports page size extensions for large pages of size 4 MByte, including: CR4.PSE for controlling the feature; the defined dirty bit in PDE (Page Directory Entries); optional reserved bit trapping in CR3; PDEs; and PTEs.	Platform
EDX[4]	TSC	If 1, supports the Time Stamp Counter, RDTSC instruction, including CR4.TSD for controlling privilege.	Platform
EDX[5]	MSR	If 1, supports the Model Specific Registers RDMSR and WRMSR Instructions. Some of the MSRs are implementation dependent.	Platform
EDX[6]	PAE	If 1, supports the Physical Address Extension which is for physical addresses greater than 32 bits, including: extended page table entry formats; an extra level in the page translation tables; and 2-MByte pages rather than 4 Mbyte pages.	Platform
EDX[7]	MCE	If 1, supports exception 18 for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the modelspecific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.	Platform
EDX[8]	CMPXCHG8B	If 1, supports the CMPXCHG8B (64 bits) Instruction, implicitly locked and atomic.	Platform
EDX[9]	APIC	If 1, the processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FEE00000H to FEE00FFFH (by default - some processors permit the APIC to be relocated).	Platform
EDX[10]	Reserved	Reserved.	
EDX[11]	SEP	If 1 supports the SYSENTER and SYSEXIT Instructions and associated MSRs.	Platform

EDX[12]	MTRR	If 1, supports the Memory Type Range Registers. (The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.)	Platform
EDX[13]	PGE	If 1, supports the global bit in paging-structure entries that map a page, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.	Platform
EDX[14]	MCA	If 1, supports the Machine Check Architecture feature. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.	Platform
EDX[15]	CMOV	If 1, supports the Conditional Move Instructions. If CPUID.01H:EDX.FPU[0] (x87 FPU present) is 1 also, supports the FCOMI and FCMOV instructions.	Platform
EDX[16]	PAT	If 1, supports the Page Attribute Table feature. (This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory accessed through a linear address on a 4KB granularity.)	Platform
EDX[17]	PSE_36	If 1, supports the 36-Bit Page Size Extension which enables 4-MByte pages addressing physical memory beyond 4 GBytes with 32-bit paging. This feature indicates that upper bits of the physical address of a 4-MByte page are encoded in bits 20:13 of the page directory entry. Such physical addresses are limited by MAXPHYADDR and may be up to 40 bits in size.	Platform
EDX[18]	PSN	If 1, supports the 96-bit Processor Serial Number identification number feature, and the feature is enabled. Available only in Pentium III, see CPUID.03H.	Platform
EDX[19]	CLFLUSH	If 1, supports the CLFLUSH instruction.	Platform
EDX[20]	Reserved	Reserved.	
EDX[21]	DS	If 1, supports the Debug Store feature which provides the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and processor event-based sampling (PEBS) facilities (see Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B).	Platform

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

EDX[22]	ACPI	If 1, supports the Thermal Monitor and Software Controlled Clock Facilities. These are internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.	Platform
EDX[23]	MMX	If 1, supports the Intel MMX Technology.	Platform
EDX[24]	FXSR	If 1, supports the FXSAVE and FXRSTOR Instructions, which are fast save and restore of the floating-point context, and the availability of CR4.OSFXSR for an operating system to indicate support of same.	Platform
EDX[25]	SSE	If 1, supports SSE.	Platform
EDX[26]	SSE2	If 1, supports SSE2.	Platform
EDX[27]	SELF_SNOOP	If 1, supports Self Snoop which is the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.	Platform
EDX[28]	HTT	If 1, the value in CPUID.1.EBX[23:16] (the Maximum number of addressable IDs for logical processors in this package) is valid for the package. If 0, there is only a single logical processor in the package and software should assume only a single APIC ID is reserved.	Platform
EDX[29]	TM	If 1, supports the Thermal Monitor feature in which the processor implements the thermal monitor automatic thermal control circuitry (TCC).Thermal Monitor.	Platform
EDX[30]	Reserved	Reserved.	
EDX[31]	PBE	If 1, supports the Pending Break Enable feature, which is the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt.	Platform

CPUID.02H -- TLB/Cache/Prefetch Information

CPUID.02H returns TLB, cache, and prefetch information. This leaf has been superseded by CPUID.04H for cache enumeration and CPUID.18H for TLB enumeration. These processors will also report new descriptor values of types 0FEh or 0FFh to refer enumerations to CPUID.04H and CPUID.18H.

- This leaf is valid if MAX_LEAF $\geq$ 02H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-11. Leaf 02H TLB/Cache/Prefetch Information

Register	Field Name	Description	Domain
EAX[7:0]	Reserved	Reserved with a value of 1	
EAX[15:8]	DESCRIPTOR_1	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EAX[23:16]	DESCRIPTOR_2	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EAX[31:24]	DESCRIPTOR_3	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EBX[7:0]	DESCRIPTOR_4	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EBX[15:8]	DESCRIPTOR_5	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EBX[23:16]	DESCRIPTOR_6	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EBX[31:24]	DESCRIPTOR_7	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
ECX[7:0]	DESCRIPTOR_8	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
ECX[15:8]	DESCRIPTOR_9	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
ECX[23:16]	DESCRIPTOR_10	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
ECX[31:24]	DESCRIPTOR_11	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EDX[7:0]	DESCRIPTOR_12	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EDX[15:8]	DESCRIPTOR_13	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EDX[23:16]	DESCRIPTOR_14	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EDX[31:24]	DESCRIPTOR_15	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor

CPUID.02H returns information about the processor's internal TLBs, cache, and prefetch hardware in the EAX, EBX, ECX, and EDX registers. The information is reported in encoded form and fall into the following categories:

- The least-significant byte in register EAX (register AL) will always return 01H. Software should ignore this value and not interpret it as an informational descriptor.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1-byte descriptors. There are four types of encoding values for the byte descriptor, the encoding of these descriptors and encoding type are listed in the table

below.

Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache, prefetch, or TLB types. The descriptors may appear in any order. Note also a processor may report a general descriptor type FFH and FEH and not report any byte descriptor of “cache type” or “*TLB type” via CPUID.02H.

Table 21-12. Encoding of CPUID Leaf 2 Descriptors

Descriptor Value	Type	Cache or TLB Description
00H	General	Null descriptor, this byte contains no information.
01H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 32 entries.
02H	TLB	Instruction TLB: 4 MByte pages, fully associative, 2 entries.
03H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 64 entries.
04H	TLB	Data TLB: 4 MByte pages, 4-way set associative, 8 entries.
05H	TLB	Data TLB1: 4 MByte pages, 4-way set associative, 32 entries.
06H	Cache	1st-level instruction cache: 8 KBytes, 4-way set associative, 32 byte line size.
08H	Cache	1st-level instruction cache: 16 KBytes, 4-way set associative, 32 byte line size.
09H	Cache	1st-level instruction cache: 32KBytes, 4-way set associative, 64 byte line size.
0AH	Cache	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size.
0BH	TLB	Instruction TLB: 4 MByte pages, 4-way set associative, 4 entries.
0CH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 32 byte line size.
0DH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 64 byte line size.
0EH	Cache	1st-level data cache: 24 KBytes, 6-way set associative, 64 byte line size.
1DH	Cache	2nd-level cache: 128 KBytes, 2-way set associative, 64 byte line size.
21H	Cache	2nd-level cache: 256 KBytes, 8-way set associative, 64 byte line size.
22H	Cache	3rd-level cache: 512 KBytes, 4-way set associative, 64 byte line size, 2 lines per sector.
23H	Cache	3rd-level cache: 1 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
24H	Cache	2nd-level cache: 1 MBytes, 16-way set associative, 64 byte line size.
25H	Cache	3rd-level cache: 2 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
29H	Cache	3rd-level cache: 4 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
2CH	Cache	1st-level data cache: 32 KBytes, 8-way set associative, 64 byte line size.
30H	Cache	1st-level instruction cache: 32 KBytes, 8-way set associative, 64 byte line size.
40H	Cache	No 2nd-level cache or, if processor contains a valid 2nd-level cache, no 3rd-level cache.
41H	Cache	2nd-level cache: 128 KBytes, 4-way set associative, 32 byte line size.
42H	Cache	2nd-level cache: 256 KBytes, 4-way set associative, 32 byte line size.
43H	Cache	2nd-level cache: 512 KBytes, 4-way set associative, 32 byte line size.
44H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 32 byte line size.
45H	Cache	2nd-level cache: 2 MByte, 4-way set associative, 32 byte line size.
46H	Cache	3rd-level cache: 4 MByte, 4-way set associative, 64 byte line size.
47H	Cache	3rd-level cache: 8 MByte, 8-way set associative, 64 byte line size.
48H	Cache	2nd-level cache: 3MByte, 12-way set associative, 64 byte line size.
49H	Cache	3rd-level cache: 4MB, 16-way set associative, 64-byte line size (Intel Xeon processor MP, Family 0FH, Model 06H) 2nd-level cache: 4 MByte, 16-way set associative, 64 byte line size.

4AH	Cache	3rd-level cache: 6MByte, 12-way set associative, 64 byte line size.
4BH	Cache	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size.
4CH	Cache	3rd-level cache: 12MByte, 12-way set associative, 64 byte line size.
4DH	Cache	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size.
4EH	Cache	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size.
4FH	TLB	Instruction TLB: 4 KByte pages, 32 entries.
50H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries.
51H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 128 entries.
52H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 256 entries.
55H	TLB	Instruction TLB: 2-MByte or 4-MByte pages, fully associative, 7 entries.
56H	TLB	Data TLB0: 4 MByte pages, 4-way set associative, 16 entries.
57H	TLB	Data TLB0: 4 KByte pages, 4-way associative, 16 entries.
59H	TLB	Data TLB0: 4 KByte pages, fully associative, 16 entries.
5AH	TLB	Data TLB0: 2 MByte or 4 MByte pages, 4-way set associative, 32 entries.
5BH	TLB	Data TLB: 4 KByte and 4 MByte pages, 64 entries.
5CH	TLB	Data TLB: 4 KByte and 4 MByte pages, 128 entries.
5DH	TLB	Data TLB: 4 KByte and 4 MByte pages, 256 entries.
60H	Cache	1st-level data cache: 16 KByte, 8-way set associative, 64 byte line size.
61H	TLB	Instruction TLB: 4 KByte pages, fully associative, 48 entries.
63H	TLB	Data TLB: 2 MByte or 4 MByte pages, 4-way set associative, 32 entries and a separate array with 1 GByte pages, 4-way set associative, 4 entries.
64H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 512 entries.
66H	Cache	1st-level data cache: 8 KByte, 4-way set associative, 64 byte line size.
67H	Cache	1st-level data cache: 16 KByte, 4-way set associative, 64 byte line size.
68H	Cache	1st-level data cache: 32 KByte, 4-way set associative, 64 byte line size.
6AH	Cache	uTLB: 4 KByte pages, 8-way set associative, 64 entries.
6BH	Cache	DTLB: 4 KByte pages, 8-way set associative, 256 entries.
6CH	Cache	DTLB: 2M/4M pages, 8-way set associative, 128 entries.
6DH	Cache	DTLB: 1 GByte pages, fully associative, 16 entries.
70H	Cache	Trace cache: 12 K-?op, 8-way set associative.
71H	Cache	Trace cache: 16 K-?op, 8-way set associative.
72H	Cache	Trace cache: 32 K-?op, 8-way set associative.
76H	TLB	Instruction TLB: 2M/4M pages, fully associative, 8 entries.
78H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 64 byte line size.
79H	Cache	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7AH	Cache	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7BH	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7CH	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7DH	Cache	2nd-level cache: 2 MByte, 8-way set associative, 64 byte line size.
7FH	Cache	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size.

80H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64-byte line size.
82H	Cache	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size.
83H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size.
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size.
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size.
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size.
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size.
A0H	DTLB	DTLB: 4k pages, fully associative, 32 entries.
B0H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries.
B1H	TLB	Instruction TLB: 2M pages, 4-way, 8 entries or 4M pages, 4-way, 4 entries.
B2H	TLB	Instruction TLB: 4KByte pages, 4-way set associative, 64 entries.
B3H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 128 entries.
B4H	TLB	Data TLB1: 4 KByte pages, 4-way associative, 256 entries.
B5H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 64 entries.
B6H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 128 entries.
BAH	TLB	Data TLB1: 4 KByte pages, 4-way associative, 64 entries.
C0H	TLB	Data TLB: 4 KByte and 4 MByte pages, 4-way associative, 8 entries.
C1H	STLB	Shared 2nd-Level TLB: 4 KByte/2MByte pages, 8-way associative, 1024 entries.
C2H	DTLB	DTLB: 2 MByte/4 MByte pages, 4-way associative, 16 entries.
C3H	STLB	Shared 2nd-Level TLB: 4 KByte /2 MByte pages, 6-way associative, 1536 entries. Also 1GByte pages, 4-way, 16 entries.
C4H	DTLB	DTLB: 2 MByte/ 4MByte pages, 4-way associative, 32 entries.
CAH	STLB	Shared 2nd-Level TLB: 4 KByte pages, 4-way associative, 512 entries.
D0H	Cache	3rd-level cache: 512 KByte, 4-way set associative, 64 byte line size.
D1H	Cache	3rd-level cache: 1 MByte, 4-way set associative, 64 byte line size.
D2H	Cache	3rd-level cache: 2 MByte, 4-way set associative, 64 byte line size.
D6H	Cache	3rd-level cache: 1 MByte, 8-way set associative, 64 byte line size.
D7H	Cache	3rd-level cache: 2 MByte, 8-way set associative, 64 byte line size.
D8H	Cache	3rd-level cache: 4 MByte, 8-way set associative, 64 byte line size.
DCH	Cache	3rd-level cache: 1.5 MByte, 12-way set associative, 64 byte line size.
DDH	Cache	3rd-level cache: 3 MByte, 12-way set associative, 64 byte line size.
DEH	Cache	3rd-level cache: 6 MByte, 12-way set associative, 64 byte line size.
E2H	Cache	3rd-level cache: 2 MByte, 16-way set associative, 64 byte line size.
E3H	Cache	3rd-level cache: 4 MByte, 16-way set associative, 64 byte line size.
E4H	Cache	3rd-level cache: 8 MByte, 16-way set associative, 64 byte line size.
EAH	Cache	3rd-level cache: 12MByte, 24-way set associative, 64 byte line size.
EBH	Cache	3rd-level cache: 18MByte, 24-way set associative, 64 byte line size.
ECH	Cache	3rd-level cache: 24MByte, 24-way set associative, 64 byte line size.
F0H	Prefetch	64-Byte prefetching.
F1H	Prefetch	128-Byte prefetching.

FEH	General	CPUID leaf 2 does not report TLB descriptor information; use CPUID leaf 18H to query TLB and other address translation parameters.
FFH	General	CPUID leaf 2 does not report cache descriptor information, use CPUID leaf 4 to query cache parameters.

Example 21-1. Example of Cache and TLB Interpretation

The first member of the family of Pentium 4 processors returns the following information about caches and TLBs when the CPUID executes with an input value of 2:

EAX 66 5B 50 01H

EBX 0H

ECX 0H

EDX 00 7A 70 00H

Which means:

- The least-significant byte (byte 0) of register EAX is set to 01H. This value should be ignored.
- The most-significant bit of all four registers (EAX, EBX, ECX, and EDX) is set to 0, indicating that each register contains valid 1-byte descriptors.
- Bytes 1, 2, and 3 of register EAX indicate that the processor has:
 - 50H - a 64-entry instruction TLB, for mapping 4-KByte and 2-MByte or 4-MByte pages.
 - 5BH - a 64-entry data TLB, for mapping 4-KByte and 4-MByte pages.
 - 66H - an 8-KByte 1st level data cache, 4-way set associative, with a 64-Byte cache line size.
- The descriptors in registers EBX and ECX are valid, but contain NULL descriptors.
- Bytes 0, 1, 2, and 3 of register EDX indicate that the processor has:
 - 00H - NULL descriptor.
 - 70H - Trace cache: 12 K- μ op, 8-way set associative.
 - 7AH - a 256-KByte 2nd level cache, 8-way set associative, with a sector, 64-byte cache line size.
 - 00H - NULL descriptor.

CPUID.03H -- Processor Serial Number

CPUID.03H returns the processor serial number, if available. Processor serial number (PSN) is not supported in the Pentium 4 processor or later.

- This leaf is valid if MAX_LEAF ≥ 03H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-13. Leaf 03H Processor Serial Number

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	PSN_31_0	Bits 00-31 of 96-bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)	Package
EDX[31:0]	PSN_63_32	Bits 32-63 of 96-bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)	Package

CPUID.04H -- Deterministic Cache Parameters

CPUID.04H returns the deterministic cache parameters for each cache level.

- This leaf is valid if CPUID.04H.00H:EAX[4:0] $\neq$ 0 and MAX_LEAF $\geq$ 04H.
- The sub-leaves are enumerated until sub-leaf n returns 0 in EAX[4:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EAX[4:0] as 0.

Table 21-14. Leaf 04H Deterministic Cache Parameters

Register	Field Name	Description	Domain
EAX[4:0]	CACHE_TYPE	0 = Null, no more caches. 1 = Data Cache. 2 = Instruction Cache. 3 = Unified Cache. 4-31 = Reserved.	Logical Processor
EAX[7:5]	CACHE_LEVEL	Cache level (starts at 1).	Logical Processor
EAX[8]	SELF_INITIALIZING_CACHE	Self initializing cache level (does not need software initialization).	Logical Processor
EAX[9]	FULLY_ASSOC	Fully associative cache.	Logical Processor
EAX[13:10]	Reserved	Reserved.	
EAX[25:14]	MAX_LP_ADDRESSABLE_IDS	Maximum number of addressable IDs for logical processors sharing this cache. Add one to the return value to get the result. The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache.	Logical Processor
EAX[31:26]	MAX_CORES_ADDRESSABLE_IDS_PKG	Maximum number of addressable IDs for processor cores in the physical package. Add one to the return value to get the result. The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. The returned value is constant for valid initial values in ECX. Valid ECX values start from 0. The maximum number of addressable IDs for processor cores in the physical package field may contain a saturated value and will not correctly identify the addressable ID reservations for cores in this package on processors where CPUID.0BH and/or CPUID.1FH exist. Processors which enumerate topology information in either CPUID.0BH or CPUID.1FH need to use those leaves to obtain the correct topology details.	Platform
EBX[11:0]	LINE_SIZE	System Coherency Line Size. Add one to the return value to get the result.	Platform
EBX[21:12]	PHYS_LINE_PARTITIONS	Physical line partitions. Add one to the return value to get the result.	Logical Processor
EBX[31:22]	NUM_WAYS	Ways of associativity. Add one to the return value to get the result.	Logical Processor

ECX[31:0]	NUM_SETS	Number of sets. Add one to the return value to get the result.	Logical Processor
EDX[0]	NOT_LWR_CACHE_FLUSH	0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache.	Logical Processor
EDX[1]	INCLUSIVE_CACHE	0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels.	Logical Processor
EDX[2]	COMPLEX_CACHE_INDEXING	0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits.	Logical Processor
EDX[31:3]	Reserved	Reserved.	

When CPUID executes with EAX set to 04H and ECX contains an index value, the processor returns encoded data that describe a set of deterministic cache parameters (for the cache level associated with the input in ECX). Valid index values start from 0. Software can enumerate the deterministic cache parameters for each level of the cache hierarchy starting with an index value of 0, until the parameters report the value associated with the cache type field is 0.

This Cache Size in Bytes

$$= (\text{Ways} + 1) * (\text{Partitions} + 1) * (\text{Line\_Size} + 1) * (\text{Sets} + 1)$$

$$= (\text{EBX}[31:22] + 1) * (\text{EBX}[21:12] + 1) * (\text{EBX}[11:0] + 1) * (\text{ECX} + 1)$$

The CPUID.04H also reports data that can be used to derive the topology of processor cores in a physical package on legacy processors. This information is constant for all valid index values. Software can query the raw data reported by executing CPUID with EAX=04H and ECX=0 and use it as part of the topology enumeration algorithm on processors that do not enumerate either CPUID.0BH or CPUID.1FH as described in Chapter 10, “Multiple-Processor Management,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

CPUID.05H -- MONITOR and MWAIT Features

CPUID.05H returns the MONITOR and MWAIT feature information.

- This leaf is valid if MAX_LEAF ≥ 05H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-15. Leaf 05H MONITOR and MWAIT Features

Register	Field Name	Description	Domain
EAX[15:0]	SMALLEST_MONITOR_LINE_SIZE	Smallest monitor-line size in bytes (default is processor's monitor granularity).	Platform
EAX[31:16]	Reserved	Reserved.	
EBX[15:0]	LARGEST_MONITOR_LINE_SIZE	Largest monitor-line size in bytes (default is processor's monitor granularity).	Platform
EBX[31:16]	Reserved	Reserved.	
ECX[0]	MONITOR_MWAIT_EXTENSIONS	If 1, supports enumeration of MONITOR/MWAIT extensions (beyond EAX and EBX registers).	Platform
ECX[1]	INTERRUPT_AS_BREAK_EVENT	If 1, supports treating interrupts as break-event for MWAIT, even when interrupts are disabled.	Platform
ECX[31:2]	Reserved	Reserved.	
EDX[3:0]	C0_SUB_STATES	Number of C0* sub C-states supported using MWAIT.	Platform
EDX[7:4]	C1_SUB_STATES	Number of C1* sub C-states supported using MWAIT.	Platform
EDX[11:8]	C2_SUB_STATES	Number of C2* sub C-states supported using MWAIT.	Platform
EDX[15:12]	C3_SUB_STATES	Number of C3* sub C-states supported using MWAIT.	Platform
EDX[19:16]	C4_SUB_STATES	Number of C4* sub C-states supported using MWAIT.	Platform
EDX[23:20]	C5_SUB_STATES	Number of C5* sub C-states supported using MWAIT.	Platform
EDX[27:24]	C6_SUB_STATES	Number of C6* sub C-states supported using MWAIT.	Platform
EDX[31:28]	C7_SUB_STATES	Number of C7* sub C-states supported using MWAIT.	Platform

NOTE

The definition of C0 through C7 states for MWAIT extensions are processor-specific C-states, not ACPI C-state.

CPUID.05H returns information about features available to MONITOR/MWAIT instructions. The MONITOR instruction is used for address-range monitoring in conjunction with MWAIT instruction. The MWAIT instruction optionally provides additional extensions for advanced power management.

CPUID.06H -- Thermal and Power Management Features

CPUID.06H returns information about thermal and power management features.

- This leaf is valid if MAX_LEAF $\geq$ 06H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-16. Leaf 06H Thermal and Power Management Features

Register	Field Name	Description	Domain
EAX[0]	DIGITAL_TEMP_SENSOR	If 1, supports Digital Temperature Sensor.	Platform
EAX[1]	TURBO_BOOST	If 1, supports Intel Turbo Boost Technology. (see description of IA32_MISC_ENABLE[38]).	Platform
EAX[2]	ALWAYS_RUNNING_APIC_TIMER	If 1, supports APIC-Timer-always-running feature.	Platform
EAX[3]	Reserved	Reserved.	
EAX[4]	POWER_LIMIT_NOTIFY	If 1, supports power limit notification controls.	Platform
EAX[5]	EXT_CLOCK_MOD	If 1, supports clock modulation duty cycle extension.	Platform
EAX[6]	PKG_THERM_MGMT	If 1, supports package thermal management.	Platform
EAX[7]	HWP	If 1, supports HWP base registers (IA32_PM_ENABLE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS).	Platform
EAX[8]	HWP_INTERRUPT	If 1, supports IA32_HWP_INTERRUPT MSR.	Platform
EAX[9]	HWP_ACTIVITY_WINDOW	If 1, supports IA32_HWP_REQUEST[bits 41:32].	Platform
EAX[10]	HWP_EPP	If 1, supports IA32_HWP_REQUEST[bits 31:24].	Platform
EAX[11]	HWP_REQUEST_PKG	If 1, supports IA32_HWP_REQUEST_PKG MSR.	Platform
EAX[12]	Reserved	Reserved.	
EAX[13]	HDC	If 1, supports HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, and IA32_THREAD_STALL MSRs.	Platform
EAX[14]	TURBO_BOOST_MAX	If 1, supports Intel® Turbo Boost Max Technology 3.0.	Platform
EAX[15]	HWP_CAP	If 1, supports Highest Performance change capability.	Platform
EAX[16]	HWP_PECI_OVERRIDE	If 1, supports HWP Peci override.	Platform
EAX[17]	FLEXIBLE_HWP	If 1, supports Flexible HWP.	Platform
EAX[18]	HWP_REQUEST_FAST_ACCESS	If 1, supports Fast access mode for the IA32_HWP_REQUEST MSR.	Platform
EAX[19]	HW_FEEDBACK	If 1, supports IA32_HW_FEEDBACK_PTR MSR, IA32_HW_FEEDBACK_CONFIG MSR, IA32_PACKAGE_THERM_STATUS MSR bit 26, and IA32_PACKAGE_THERM_INTERRUPT MSR bit 25.	Platform
EAX[20]	HWP_REQUEST_IGNORE_IDLE	If 1, supports Ignoring Idle Logical Processor HWP request.	Platform
EAX[21]	Reserved	Reserved.	
EAX[22]	HWP_CTL	If 1, supports IA32_HWP_CTL MSR.	Platform

EAX[23]	THREAD_DIRECTOR	If 1, supports Intel® Thread Director. IA32_HW_FEEDBACK_CHAR and IA32_HW_FEEDBACK_THREAD_CONFIG MSRs are supported if set.	Platform
EAX[31:24]	Reserved	Reserved.	
EBX[3:0]	DTS_NUM_INT_THRESHOLDS	Number of Interrupt Thresholds in Digital Thermal Sensor.	Platform
EBX[31:4]	Reserved	Reserved.	
ECX[0]	HW_FEEDBACK_CAP	If 1, supports IA32_MPERF and IA32_APERF which provide a measure of delivered processor performance (since last reset of the counters), as a percentage of the expected processor performance when running at the TSC frequency.	Platform
ECX[2:1]	Reserved	Reserved.	
ECX[3]	ENERGY_PERF_BIAS	If 1, supports performance-energy bias preference and a new architectural MSR called IA32_ENERGY_PERF_BIAS (1BOH).	Platform
ECX[7:4]	Reserved	Reserved.	
ECX[15:8]	HW_FEEDBACK_NUM_CLASSES	Number of Intel® Thread Director classes supported by the processor. Information for that many classes is written into the Intel Thread Director Table by the hardware.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[7:0]	HW_FEEDBACK_CAPS	Bitmap of supported hardware feedback interface capabilities. 0 = If 1, supports performance capability reporting. 1 = If 1, supports energy efficiency capability reporting. 2-7 = Reserved. Bits 0 and 1 will always be set together.	Package
EDX[11:8]	HW_FEEDBACK_TABLE_SIZE	Enumerates the size of the hardware feedback interface structure in number of 4 KB pages. Add one to the return value to get the result.	Package
EDX[15:12]	Reserved	Reserved.	
EDX[31:16]	HW_FEEDBACK_TABLE_INDEX	Index (starting at 0) of this logical processor's row in the hardware feedback interface structure. Note that on some parts the index may be same for multiple logical processors. On some parts the indices may not be contiguous, i.e., there may be unused rows in the hardware feedback interface structure.	Logical Processor

Details around these features are described in Chapter 16, "Power and Thermal Management," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

CPUID.07H -- Structured Extended Feature Flags

CPUID.07H returns structured extended feature flags enumeration information. The sub-sections of Section provide leaf 07H information.

- This leaf is valid if MAX_LEAF $\geq$ 07H.
- The maximum sub-leaf value for ECX is specified in CPUID.07H.00H.EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid Sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.07H.00H -- Structured Extended Feature Flags Main Sub-Leaf

CPUID.07H.00H returns the maximum input value of the highest leaf 07H sub-leaf; and EBX, ECX, and EDX contain information of extended feature flags.

Table 21-17. Leaf 07H Sub-Leaf (ECX=0) Output Registers

CPUID Output Registers	Field Name	Description	CPUID Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value for supported leaf 07H subleaves.	Platform
EBX[31:0]		Extended Feature Flags Information in EBX (see "CPUID.07H.00H:EBX—Extended Feature Flags Information")	
ECX[31:0]		Extended Feature Flags Information in ECX (see "CPUID.07H.00H:ECX—Extended Feature Flags Information")	
EDX[31:0]		Extended Feature Flags Information in EDX (see "CPUID.07H.00H:EDX—Extended Feature Flags Information")	

Table 21-18. Leaf 07H.00H Structured Extended Feature Flags Returned in EAX

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value for supported leaf 07H sub-leaves.	Platform

CPUID.07H.00H:EBX Extended Feature Flags Information

The EBX register of CPUID.07H.00H returns the information shown below.

Table 21-19. Leaf 07H.00H Structured Extended Feature Flags Returned in EBX

Register	Field Name	Description	Domain
EBX[0]	FSGSBASE	If 1, supports RDFSBASE/RDGSBASE/WRFS-BASE/WRGSBASE.	Platform
EBX[1]	TSC_ADJUST	If 1, the IA32_TSC_ADJUST MSR is supported.	Platform
EBX[2]	SGX	If 1, supports Intel® Software Guard Extensions (Intel® SGX Extensions).	Platform
EBX[3]	BMI1	If 1, supports the BMI1 instructions.	Platform
EBX[4]	HLE	If 1, supports the Hardware Lock Elision instruction set.	Platform
EBX[5]	AVX2	If 1, supports Intel® Advanced Vector Extensions 2 (Intel® AVX2).	Platform

EBX[6]	FDP_EXCPTN_ONLY	If 1, the x87 FPU Data Pointer is updated only on x87 exceptions.	Platform
EBX[7]	SMEP	If 1, supports Supervisor-Mode Execution Prevention.	Platform
EBX[8]	BMI2	If 1, supports the BMI2 instructions.	Platform
EBX[9]	ENH_REP_MOVSB_STOSB	If 1, supports Enhanced REP MOVSB/STOSB.	Platform
EBX[10]	INVPCID	If 1, supports INVPCID instruction for system software that manages process-context identifiers.	Platform
EBX[11]	RTM	If 1, supports the Restricted Transactional Memory instruction set.	Platform
EBX[12]	RDT_M	If 1, supports Intel® Resource Director Technology (Intel® RDT) Monitoring capability.	Platform
EBX[13]	FCS_FDS_DEPRECATION	If 1, deprecates FPU CS and FPU DS values.	Platform
EBX[14]	MPX	If 1, supports Intel® Memory Protection Extensions.	Platform
EBX[15]	RDT_A	If 1, supports Intel® Resource Director Technology (Intel® RDT) Allocation capability.	Platform
EBX[16]	AVX512F	If 1, supports the AVX512F instructions.	Platform
EBX[17]	AVX512DQ	If 1, supports the AVX512DQ instructions.	Platform
EBX[18]	RDSEED	If 1, supports the RDSEED instruction.	Platform
EBX[19]	ADX	If 1, supports the ADX instructions.	Platform
EBX[20]	SMAP	If 1, supports Supervisor-Mode Access Prevention and the CLAC/STAC instructions.	Platform
EBX[21]	AVX512_IFMA	If 1, supports the AVX512_IFMA instructions.	Platform
EBX[22]	Reserved	Reserved.	
EBX[23]	CLFLUSHOPT	If 1, supports the CLFLUSHOPT instruction.	Platform
EBX[24]	CLWB	If 1, supports the CLWB instruction.	Platform
EBX[25]	INTEL_PROC_TRACE	If 1, supports Intel® Processor Trace.	Platform
EBX[26]	AVX512PF	If 1, supports the AVX512PF instructions. (Intel® Xeon Phi™ only.)	Platform
EBX[27]	AVX512ER	If 1, supports the AVX512ER instructions. (Intel® Xeon Phi™ only.)	Platform
EBX[28]	AVX512CD	If 1, supports the AVX512CD instructions.	Platform
EBX[29]	SHA	If 1, supports Intel® Secure Hash Algorithm Extensions (Intel® SHA Extensions).	Platform
EBX[30]	AVX512BW	If 1, supports the AVX512BW instructions.	Platform
EBX[31]	AVX512VL	If 1, supports the AVX512VL instructions.	Platform

CPUID.07H.00H:ECX Extended Feature Flags Information

The ECX register of CPUID.07H.00H returns the information shown below.

Table 21-20. Leaf 07H.00H Structured Extended Feature Flags Returned in ECX

Register	Field Name	Description	Domain

ECX[0]	PREFETCHWT1	If 1, supports the PREFETCHWT1 instruction. (Intel® Xeon Phi™ only.)	Platform
ECX[1]	AVX512_VBMI	If 1, supports the AVX512_VBMI instructions.	Platform
ECX[2]	UMIP	If 1, supports user-mode instruction prevention.	Platform
ECX[3]	PKU	If 1, supports protection keys for user-mode pages.	Platform
ECX[4]	OSPKE	If 1, the OS has set CR4.PKE to enable protection keys and the RDPKRU/WRPKRU instructions.	Logical Processor
ECX[5]	WAITPKG	If 1, supports the TPAUSE, UMONITOR, and UMWAIT instructions.	Platform
ECX[6]	AVX512_VBMI2	If 1, supports the AVX512_VBMI2 instructions.	Platform
ECX[7]	CET_SS	If 1, supports CET shadow stack features. Processors that set this bit define bits 1:0 of the IA32_U_CET and IA32_S_CET MSRs. Enumerates support for the following MSRs: IA32_INTERRUPT_SPP_TABLE_ADDR, IA32_PL3_SSP, IA32_PL2_SSP, IA32_PL1_SSP, and IA32_PLO_SSP.	Platform
ECX[8]	GFNI	If 1, supports the GFNI instruction set.	Platform
ECX[9]	VAES	If 1 and Intel AVX supported, supports the VEX-encoded AES instruction set.	Platform
ECX[10]	VPCLMULQDQ	If 1 and Intel AVX supported, supports the VPCLMULQDQ instruction.	Platform
ECX[11]	AVX512_VNNI	If 1, supports the AVX512_VNNI instructions.	Platform
ECX[12]	AVX512_BITALG	If 1, supports the AVX512_BITALG instructions.	Platform
ECX[13]	TME_EN	If 1, the following MSRs are supported: IA32_TME_CAPABILITY, IA32_TME_ACTIVATE, IA32_TME_EXCLUDE_MASK, and IA32_TME_EXCLUDE_BASE.	Platform
ECX[14]	AVX512_VPOPCNTDQ	If 1, supports the AVX512_VPOPCNTDQ instructions.	Platform
ECX[15]	Reserved	Reserved.	
ECX[16]	LA57	If 1, supports 57-bit linear addresses and five-level paging.	Platform
ECX[21:17]	MPX_MAWAU	The value of MAWAU used by the BNDLDX and BNDSTX instructions in 64-bit mode.	Platform
ECX[22]	RDPID	If 1, RDPID and the IA32_TSC_AUX MSR are available.	Platform
ECX[23]	KEY_LOCKER	If 1, supports Key Locker.	Platform
ECX[24]	BUS_LOCK_DETECT	If 1, indicates support for OS bus-lock detection.	Platform
ECX[25]	CLDEMOT	If 1, supports cache line demote.	Platform
ECX[26]	Reserved	Reserved.	
ECX[27]	MOVDIRI	If 1, supports the MOVDIRI instruction.	Platform
ECX[28]	MOVDIR64B	If 1, supports the MOVDIR64B instruction.	Platform
ECX[29]	ENQCMD	If 1, supports Enqueue Stores.	Platform

ECX[30]	SGX_LC	If 1, supports SGX Launch Configuration.	Platform
ECX[31]	PKS	If 1, supports protection keys for supervisor-mode pages.	Platform

CPUID.07H.00H:EDX Extended Feature Flags Information

The EDX register of CPUID.07H.00H returns the information shown below.

Table 21-21. Leaf 07H.00H Structured Extended Feature Flags Returned in EDX

Register	Field Name	Description	Domain
EDX[0]	Reserved	Reserved.	
EDX[1]	SGX_KEYS	If 1, supports Attestation Services for Intel® SGX.	Platform
EDX[2]	AVX512_4VNNIW	If 1, supports the AVX512_4VNNIW instructions. (Intel® Xeon Phi™ only.)	Platform
EDX[3]	AVX512_4FMAPS	If 1, supports the AVX512_4FMAPS instructions. (Intel® Xeon Phi™ only.)	Platform
EDX[4]	FAST_SHORT_REP_MOVSB	If 1, supports Fast Short REP MOVSB.	Platform
EDX[5]	UINTR	If 1, supports user interrupts.	Platform
EDX[7:6]	Reserved	Reserved.	
EDX[8]	AVX512_VP2INTERSECT	If 1, supports the AVX512_VP2INTERSECT instruction.	Platform
EDX[9]	MCU_OPT_CTRL	If 1, supports both the IA32_MCU_OPT_CTRL MSR and its bit 0 (RNGDS_MITG_DIS).	Platform
EDX[10]	MD_CLEAR	If 1, supports MD_CLEAR.	Platform
EDX[11]	RTM_ALWAYS_ABORT	If 1, any execution of XBEGIN immediately aborts and transitions to the specified fallback address.	Platform
EDX[12]	Reserved	Reserved.	
EDX[13]	RTM_FORCE_ABORT	If 1, supports RTM_FORCE_ABORT and the IA32_TSX_FORCE_ABORT MSR. These allow software to set IA32_TSX_FORCE_ABORT[0] (RTM_FORCE_ABORT).	Platform
EDX[14]	SERIALIZE	If 1, supports SERIALIZE instruction.	Platform
EDX[15]	HYBRID	If 1, the processor is identified as a hybrid part. If CPUID.00H.MAXLEAF ≥ 1AH and CPUID.1AH:EAX <> 0, then the Native Model ID Enumeration Leaf 1AH exists.	Platform
EDX[16]	TSXLDTRK	If 1, supports Intel TSX suspend/resume of load address tracking.	Platform
EDX[17]	Reserved	Reserved.	
EDX[18]	PCONFIG	If 1, supports the PCONFIG instruction.	Platform
EDX[19]	ARCH_LBRS	If 1, supports architectural LBRS.	Platform
EDX[20]	CET_IBT	If 1, supports CET indirect branch tracking features. Processors that set this bit define bits 5:2 and bits 63:10 of the IA32_U_CET and IA32_S_CET MSRs.	Platform

EDX[21]	Reserved	Reserved.	
EDX[22]	AMX_BF16	If 1, supports tile computational operations on bfloat16 numbers.	Platform
EDX[23]	AVX512_FP16	If 1, supports the FP16 data type with AVX512 instructions.	Platform
EDX[24]	AMX_TILE	If 1, supports tile architecture.	Platform
EDX[25]	AMX_INT8	If 1, supports tile computational operations on 8-bit integers.	Platform
EDX[26]	IBRS_IBPB	If 1, supports indirect branch restricted speculation (IBRS) and the indirect branch predictor barrier (IBPB). Processors that set this bit support the IA32_SPEC_CTRL MSR and the IA32_PRED_CMD MSR. They allow software to set IA32_SPEC_CTRL[0] (IBRS) and IA32_PRED_CMD[0] (IBPB).	Platform
EDX[27]	SPEC_CTRL_ST_PREDICTORS	If 1, supports single thread indirect branch predictors (STIBP). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[1] (STIBP).	Platform
EDX[28]	L1D_FLUSH_INTERFACE	If 1, supports L1D_FLUSH. Processors that set this bit support the IA32_FLUSH_CMD MSR. They allow software to set IA32_FLUSH_CMD[0] (L1D_- FLUSH).	Platform
EDX[29]	ARCH_CAPABILITIES	If 1, supports the IA32_ARCH_CAPABILITIES MSR.	Platform
EDX[30]	CORE_CAPABILITIES	If 1, supports the IA32_CORE_CAPABILITIES MSR. IA32_CORE_CAPABILITIES is an architectural MSR that enumerates model-specific features. A bit being set in this MSR indicates that a model specific feature is supported; software must still consult CPUID family/model/stepping to determine the behavior of the enumerated feature as features enumerated in IA32_CORE_CAPABILITIES may have different behavior on different processor models. Some of these features may have behavior that is consistent across processor models (and for which consultation of CPUID family/model/stepping is not necessary); such features are identified explicitly where they are documented in this manual.	Platform
EDX[31]	SPEC_CTRL_SSBD	If 1, supports Speculative Store Bypass Disable (SSBD). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[2] (SSBD).	Platform

CPUID.07H.01H -- Structured Extended Feature Sub-Leaf 1

CPUID.07H.01H:EAX Extended Feature Information

The EAX register of CPUID.07H.01H returns the information shown below.

Table 21-22. Leaf 07H.01H Structured Extended Feature Flags Returned in EAX

Register	Field Name	Description	Domain
EAX[0]	SHA512	If 1, supports the SHA512 instructions.	Platform
EAX[1]	SM3	If 1, supports the SM3 instructions.	Platform
EAX[2]	SM4	If 1, supports the SM4 instructions.	Platform
EAX[3]	Reserved	Reserved.	
EAX[4]	AVX_VNNI	If 1, supports the VEX-encoded versions of the Vector Neural Network Instructions.	Platform
EAX[5]	AVX512_BF16	If 1, supports the Vector Neural Network Instructions supporting BFLOAT16 inputs and conversion instructions from IEEE single precision.	Platform
EAX[6]	LASS	If 1, supports Linear Address Space Separation.	Platform
EAX[7]	CMPCCXADD	If 1, supports the CMPccXADD instruction.	Platform
EAX[8]	ARCH_PERFMON_EXT	If 1, supports ArchPerfmonExt. When set, indicates that the Architectural Performance Monitoring Extended Leaf (EAX=23H) is valid.	Platform
EAX[9]	Reserved	Reserved.	
EAX[10]	FAST_REP_MOVSB	If 1, supports fast zero-length REP MOVSB.	Platform
EAX[11]	FAST_REP_STOSB	If 1, supports fast short REP STOSB.	Platform
EAX[12]	FAST_REP_CMPSB_SCASB	If 1, supports fast short REP CMPSB, REP SCASB.	Platform
EAX[16:13]	Reserved	Reserved.	
EAX[17]	FRED	If 1, supports Flexible Return and Event Delivery and the architectural state (MSRs) defined by FRED. Any Intel processor that enumerates support for FRED transitions will also enumerate support for LKGS.	Platform
EAX[18]	LKGS	If 1, supports the LKGS (load into IA32_KERNEL_GS_BASE) instruction.	Platform
EAX[19]	WRMSRNS	If 1, supports the WRMSRNS instruction.	Platform
EAX[20]	Reserved	Reserved.	
EAX[21]	AMX_FP16	If 1, supports tile computational operations on FP16 numbers.	Platform
EAX[22]	HRESET	If 1, supports history reset via the HRESET instruction and the IA32_HRESET_ENABLE MSR. When set, indicates that the Processor History Reset Leaf (EAX = 20H) is valid.	Platform
EAX[23]	AVX_IFMA	If 1, supports the AVX-IFMA instructions.	Platform
EAX[25:24]	Reserved	Reserved.	
EAX[26]	LAM	If 1, supports Linear Address Masking.	Platform
EAX[27]	MSRLIST	If 1, supports the RDMSRLIST and WRMSRLIST instructions and the IA32_BARRIER MSR.	Platform
EAX[29:28]	Reserved	Reserved.	
EAX[30]	INVD_DISABLE_POST_BIOS_DONE	If 1, supports INVD execution prevention after BIOS Done.	Platform

EAX[31]	Reserved	Reserved.	

CPUID.07H.01H:EBX Extended Feature Information

The EBX register of CPUID.07H.01H returns the information shown below.

Table 21-23. Leaf 07H.01H Structured Extended Feature Flags Returned in EBX

Register	Field Name	Description	Domain
EBX[0]	PPIN	If 1, supports the IA32_PPIN and IA32_PPIN_CTL MSRs.	Platform
EBX[1]	PBNDKB	If 1, supports the PBNDKB instruction and enumerates the existence of the IA32_TSE_CAPABILITY MSR.	Platform
EBX[2]	Reserved	Reserved.	
EBX[3]	CPUIDMAXVAL_LIM_RMV	If 1, IA32_MISC_ENABLE[bit 22] cannot be set to 1 to limit the value returned by CPUID.00H:EAX[7:0].	Platform
EBX[31:4]	Reserved	Reserved.	

CPUID.07H.01H:ECX Extended Feature Information

The ECX register of CPUID.07H.01H returns the information shown below.

Table 21-24. Leaf 07H.01H Structured Extended Feature Flags Returned in ECX

Register	Field Name	Description	Domain
ECX[0]	RDT_M_ASYM	If 1, at least one logical processor on this platform supports Asymmetrical Intel® RDT Monitoring capability.	
ECX[1]	RDT_A_ASYM	If 1, at least one logical processor on this platform supports Asymmetrical Intel® RDT Allocation capability.	
ECX[31:2]	Reserved	Reserved.	

CPUID.07H.01H:EDX Extended Feature Information

The EDX register of CPUID.07H.01H returns the information shown below.

Table 21-25. Leaf 07H.01H Structured Extended Feature Flags Returned in EDX

Register	Field Name	Description	Domain
EDX[3:0]	Reserved	Reserved.	
EDX[4]	AVX_VNNI_INT8	If 1, supports the AVX-VNNI-INT8 instructions.	Platform
EDX[5]	AVX_NE_CONVERT	If 1, supports the AVX-NE-CONVERT instructions.	Platform
EDX[7:6]	Reserved	Reserved.	
EDX[8]	AMX_COMPLEX	If 1, supports the AMX_COMPLEX instructions.	
EDX[9]	Reserved	Reserved.	
EDX[10]	AVX_VNNI_INT16	If 1, supports the AVX-VNNI-INT16 instructions.	Platform

EDX[13:11]	Reserved	Reserved.	
EDX[14]	PREFETCHI	If 1, supports the PREFETCHIT0/1 instructions.	Platform
EDX[16:15]	Reserved	Reserved.	
EDX[17]	UIRET_UIF	If 1, UIRET sets UIF to the value of bit 1 of the RFLAGS image loaded from the stack.	Platform
EDX[18]	CET_SSS	If 1, indicates that an operating system can enable supervisor shadow stacks as long as it ensures that a supervisor shadow stack cannot become prematurely busy due to page faults (see Section 17.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). When emulating the CPUID instruction, a virtual-machine monitor (VMM) should return this bit as 1 only if it ensures that VM exits cannot cause a guest supervisor shadow stack to appear to be prematurely busy. Such a VMM could set the "prematurely busy shadow stack" VM-exit control and use the additional information that it provides.	Platform
EDX[19]	AVX10	If 1, supports the Intel® AVX10 instructions and indicates the presence of CPUID.24H, which enumerates the version number.	Platform
EDX[21:20]	Reserved	Reserved.	
EDX[22]	SEC-TEE-ATTESTATION	N/A	Platform
EDX[23]	MWAIT	If 1, MWAIT is supported (even if CPUID.01H:ECX.MONITOR[3] is enumerated as 0).	Platform
EDX[24]	SLSM	Static LSM is supported on this platform. If set, IA32_INTEGRITY_STATUS (0x2DC) is available for software use.	Platform
EDX[31:25]	Reserved	Reserved.	

CPUID.07H.02H -- Structured Extended Feature Sub-Leaf 2

CPUID.07H.02H returns the structured extended feature information contained in the sub-sections of this section.

Table 21-26. Leaf 07H Sub-Leaf (ECX=2) Output Registers

CPUID Output Registers	Description
EAX[31:0]	Reserved
EBX[31:0]	Reserved
ECX[31:0]	Reserved
EDX[31:0]	Extended Feature Information (see "CPUID.07H.02H:EDX—Extended Feature Information")

CPUID.07H.02H:EDX Extended Feature Information

The EDX register of CPUID.07H.02H returns the information shown below.

Table 21-27. CPUID.07H.02H Extended Feature Information Provided in EDX¹

Register	Field Name	Description	Domain
EDX[0]	PSFD	If 1, supports bit 7 of the IA32_SPEC_CTRL MSR. Bit 7 of this MSR disables Fast Store Forwarding Predictor without disabling Speculative Store Bypass.	Platform
EDX[1]	IPRED_CTRL	If 1, supports bits 3 and 4 of the IA32_SPEC_CTRL MSR. Bit 3 of this MSR enables IPRED_DIS control for CPL3. Bit 4 of this MSR enables IPRED_DIS control for CPL0/1/2.	Platform
EDX[2]	RRSBA_CTRL	If 1, supports bits 5 and 6 of the IA32_SPEC_CTRL MSR. Bit 5 of this MSR disables RRSBA behavior for CPL3. Bit 6 of this MSR disables RRSBA behavior for CPL0/1/2.	Platform
EDX[3]	DDPD_U	If 1, supports bit 8 of the IA32_SPEC_CTRL MSR. Bit 8 of this MSR disables Data Dependent Prefetcher.	Platform
EDX[4]	BHI_CTRL	If 1, supports bit 10 of the IA32_SPEC_CTRL MSR. Bit 10 of this MSR enables BHI_DIS_S behavior.	Platform
EDX[5]	MCDT_NO	If 1, the processor does not exhibit MXCSR Configuration Dependent Timing (MCDT) behavior and does not need to be mitigated to avoid data-dependent behavior for certain instructions.	Platform
EDX[6]	UC_LOCK_DISABLE	If 1, supports the UC-lock disable feature and it causes #AC.	Platform
EDX[7]	MONITOR_MITG_NO	If 1, the MONITOR/UMONITOR instructions are not affected by performance or power issues due to MONITOR/UMONITOR instructions exceeding the capacity of an internal monitor tracking table. If 0, then the product may be affected by this issue.	Platform
EDX[31:8]	Reserved	Reserved.	

NOTE

1. Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub-leaf index, EDX returns 0.

CPUID.08H -- Reserved

This leaf is reserved.

Table 21-28. Leaf 08H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.09H -- Direct Cache Access Information

- CPUID.09H returns information about Direct Cache Access capabilities.
- This leaf is valid if CPUID.01H:ECX.DCA[18] = 1 and MAX_LEAF ≥ 09H.
 - This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-29. Leaf 09H Direct Cache Access Information

Register	Field Name	Description	Domain
EAX[31:0]	PLATFORM_DCA_CAP	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H).	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

The IA32_PLATFORM_DCA_CAP MSR is valid when CPUID.01H:ECX.DCA[18] = 1.

CPUID.0AH -- Architectural Performance Monitoring

CPUID.0AH returns information about support for architectural performance monitoring capabilities.

- This leaf is valid if CPUID.0AH:EAX[7:0] (Version ID) > 0 and MAX_LEAF ≥ 0AH.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-30. Leaf 0AH Architectural Performance Monitoring

Register	Field Name	Description	Domain
EAX[7:0]	VERSION	Version ID of architectural performance monitoring.	Platform
EAX[15:8]	NUM_GP_CTRS	Number of general-purpose performance monitoring counter(s) per logical processor.	Platform
EAX[23:16]	GP_CTR_WIDTH	Bit width of general-purpose, performance monitoring counter.	Platform
EAX[31:24]	EVENT_ENUM_LENGTH	Length of EBX bit vector to enumerate architectural performance monitoring events. Architectural event x is supported if EBX[x]=0 && EAX[31:24]>x.	Platform
EBX[0]	CORE_CYC_NA	Core cycle event not available if 1 or if EAX[31:24]<1.	Platform
EBX[1]	INTR_RET_NA	Instruction retired event not available if 1 or if EAX[31:24]<2.	Platform
EBX[2]	REF_CYC_NA	Reference cycles event not available if 1 or if EAX[31:24]<3.	Platform
EBX[3]	LLC_CYC_NA	Last-level cache reference event not available if 1 or if EAX[31:24]<4.	Platform
EBX[4]	LLC_MISSES_NA	Last-level cache misses event not available if 1 or if EAX[31:24]<5.	Platform
EBX[5]	BR_INSTR_RET_NA	Branch instruction retired event not available if 1 or if EAX[31:24]<6.	Platform
EBX[6]	BR_MISPRED_RET_NA	Branch mispredict retired event not available if 1 or if EAX[31:24]<7.	Platform
EBX[7]	SLOTS_NA	Top-down slots event not available if 1 or if EAX[31:24]<8.	Platform
EBX[8]	BACKEND_NA	Topdown backend bound not available if 1 or if EAX[31:24] < 9.	Platform
EBX[9]	BADSPEC_NA	Topdown bad speculation not available if 1 or if EAX[31:24] < 10.	Platform
EBX[10]	FRONTEND_NA	Topdown frontend bound not available if 1 or if EAX[31:24] < 11.	Platform
EBX[11]	RETIRING_NA	Topdown retiring not available if 1 or if EAX[31:24] < 12.	Platform
EBX[12]	LBR_INSERTS_NA	LBR inserts not available if 1 or if EAX[31:24] < 13.	Platform
EBX[31:13]	Reserved	Reserved.	

ECX[31:0]	FIXED_CTR_MASK	Supported fixed counters bit mask. Fixed-function performance counter 'i' is supported if bit 'i' is 1 (first counter index starts at zero). It is recommended to use the following logic to determine if a Fixed Counter is supported: FxCtr[i]_is_supported := ECX[i] (EDX[4:0] > i);	Platform
EDX[4:0]	NUM_FIXED_CTR	Number of contiguous fixed-function performance counters starting from 0 (if Version ID > 1).	Platform
EDX[12:5]	FIXED_CTR_WIDTH	Bit width of fixed-function performance counters (if Version ID > 1).	Platform
EDX[14:13]	Reserved	Reserved.	
EDX[15]	ANYTHREAD_DEPRECATION	Starting with Architectural Performance Monitoring Version 5, this field indicates that a processor supports AnyThread mode deprecation. If this field is set, software can choose to ignore guidelines in "AnyThread Counting and Software Evolution" of Chapter 21, "Performance Monitoring," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B	Platform
EDX[19:16]	SLOTS_PER_CYC	If this field is non-zero, it represents the number of Top-down Microarchitecture Analysis (TMA) slots per cycle. This number can be multiplied by the number of cycles (from CPU_CLK_UNHALTED.THREAD / CPU_CLK_UNHALTED.CORE or IA32_FIXED_CTR1) to determine the total number of slots. If this field is zero, IA32_FIXED_CTR3 should be used to determine the total number of slots.	Platform
EDX[31:20]	Reserved	Reserved.	

For each version of architectural performance monitoring capability, software must enumerate this leaf to discover the programming facilities and the architectural performance events available in the processor. The details are described in Chapter 21, "Performance Monitoring," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

CPUID.0BH -- Extended Topology

CPUID.0BH returns information about Extended Topology. CPUID.1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of leaf 1FH before using leaf 0BH.

- This leaf is valid if CPUID.0BH.00H:EBX[15:0] $\neq$ 0 and MAX_LEAF $\geq$ 0BH.
 - When the leaf is invalid, CPUID.0BH.00H:ECX.DOMAIN_TYPE[15:8] will report the Domain Type ID as Invalid (0).
- The sub-leaves are enumerated until sub-leaf n returns 0 in EBX[15:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EBX[15:0] as 0.

CPUID.0BH -- ECX $\geq$ 0

Table 21-31. Leaf 0BH Extended Topology

Register	Field Name	Description	Domain
EAX[4:0]	SHIFT_COUNT	The number of bits that the x2APIC ID must be shifted to the right to address instances of the next higher-scoped domain. When logical processor is not supported by the processor, the value of this field at the Logical Processor domain sub-leaf may be returned as either 0 (no allocated bits in the x2APIC ID) or 1 (one allocated bit in the x2APIC ID); software should plan accordingly.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[15:0]	NEXT_LEVEL_NUM_LP	The number of logical processors across all instances of this domain within the next higher-scoped domain. (For example, in a processor socket/package comprising "M" cores of "N" logical processors each, the "core" domain sub-leaf value of this field would be M*N.) This number reflects configuration as shipped by Intel. This field may also contain asymmetric values across different logical processors, as an example of a mix of cores that support more than one logical processor with cores that support only one logical processor. Note, software must not use this field to enumerate processor topology. Software must not use the value of EBX[15:0] to enumerate processor topology of the system. The value is only intended for display and diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.	Logical Processor
EBX[31:16]	Reserved	Reserved.	
ECX[7:0]	LEVEL_NUM	The input ECX sub-leaf index.	Platform

ECX[15:8]	DOMAIN_TYPE	This field provides an identification value which indicates the domain shown in the table. Although domains are ordered, their assigned identification values are not and software should not depend on it. Note that enumeration values of 0 and 3-255 are reserved.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	X2APIC_ID	The X2APIC ID of this logical processor.	Logical Processor

The sub-leaves of CPUID.0BH describe an ordered hierarchy of logical processors starting from the smallest-scoped domain of a Logical Processor (sub-leaf index 0) to the Core domain (sub-leaf index 1) to the largest-scoped domain (the last valid sub-leaf index) that is implicitly subordinate to the unenumerated highest-scoped domain of the processor package (socket).

The details of each valid domain is enumerated by a corresponding sub-leaf. Details for a domain include its type and how all instances of that domain determine the number of logical processors and x2 APIC ID partitioning at the next higher-scoped domain. The ordering of domains within the hierarchy is fixed architecturally as shown below. For a given processor, not all domains may be relevant or enumerated; however, the logical processor and core domains are always enumerated.

For two valid sub-leaves N and N+1, sub-leaf N+1 represents the next immediate higher-scoped domain with respect to the domain of sub-leaf N for the given processor.

If sub-leaf index “N” returns an invalid domain type in ECX[15:08] (00H), then all sub-leaves with an index greater than “N” also return an invalid domain type. A sub-leaf returning an invalid domain always returns 0 in EAX and EBX.

Table 21-32. Hierarchy of Valid Domain Enumerations in CPUID.0BH:ECX[15:8]

Hierarchy	Domain	Domain Type ID Value
Invalid	Invalid	0
Lowest	Logical Processor	1
...	Core	2
Highest	Package/Socket	(Implied)
Reserved	Reserved	3-255

CPUID.0CH -- Reserved

Table 21-33. Leaf 0CH Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0DH -- Processor Extended State

CPUID.0DH returns a bit-vector representation of all processor state extensions that are supported in the processor and storage size requirements of the XSAVE/XRSTOR area.

- This leaf is valid if CPUID.01H:ECX.XSAVE[26] = 1 and MAX_LEAF ≥ 0DH.
- Sub-leaves 0 and 1 are always valid; consult them to determine which other sub-leaves are present as described in “CPUID.0DH.n, n>01H—State Sub-Leaves”.

CPUID.0DH.00H -- Processor Extended State Main Sub-Leaf

CPUID.0DH.00H returns the processor extended state information.

Table 21-34. Leaf 0DH.00H Processor Extended State

Register	Field Name	Description	Domain
EAX[0]	X87	x87 state.	Platform
EAX[1]	SSE	SSE state.	Platform
EAX[2]	AVX	AVX state.	Platform
EAX[3]	MPX_BNDREGS	MPX state.	Platform
EAX[4]	MPX_BNDCSR	MPX state.	Platform
EAX[5]	AVX512_OPMASK	AVX-512 Opmask state.	Platform
EAX[6]	AVX512_ZMM_HI256	AVX-512 ZMM upper 256 data state.	Platform
EAX[7]	AVX512_HI16_ZMM	AVX-512 upper 16 ZMM registers state.	Platform
EAX[8]	N/A	Always returns 0 (Allocated for IA32_XSS).	Platform
EAX[9]	PKRU	PKRU state.	Platform
EAX[16:10]	N/A	Always returns 0 (Allocated for IA32_XSS).	Platform
EAX[17]	AMX_TILECFG	TILECFG state.	Platform
EAX[18]	AMX_TILEDATA	TILEDATA state.	Platform
EAX[31:19]	Reserved	Reserved.	
EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURES	Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.	Logical Processor
ECX[31:0]	XSAVE_BYTES_SUPPORTED_FEATURES	Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processors, i.e. all the valid bit fields in XCR0.	Platform
EDX[31:0]	VALID_XCR0_UPPER_32	Reports the supported bits of the upper 32 bits of XCR0. XCR0[n+32] can be set to 1 only if EDX[n] is 1.	Platform

The EAX register of CPUID.0DH.00H reports the supported bits of the lower 32 bits of XCR0. XCR0[n] can be set to 1 only if EAX[n] is 1. The details are described in Chapter 13.2, “Enumeration of CPUID Support for XSAVE Instructions and XSAVE-Supported Features,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

CPUID.0DH.01H -- Feature and Supervisor State Sub-Leaf

CPUID.0DH.01H returns feature and supervisor state information.

Table 21-35. Leaf 0DH.01H Processor Extended State

Register	Field Name	Description	Domain
EAX[0]	XSAVEOPT	If 1, supports XSAVEOPT.	Platform
EAX[1]	XSAVEC	If 1, supports XSAVEC and the compacted form of XRSTOR.	Platform
EAX[2]	XGETBV1	If 1, supports XGETBV with ECX = 1.	Platform
EAX[3]	XSAVES	If 1, supports XSAVES/XRSTORS and IA32_XSS.	Platform
EAX[4]	XFD	If 1, supports extended feature disable (XFD).	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	XSAVES_BYTES_ENABLED_FEATURES	The size in bytes of the XSAVE area containing all states enabled by XCR0 IA32_XSS. If EAX[3] is enumerated as 0 and EAX[1] is enumerated as 1, EBX enumerates the size of the XSAVE area containing all states enabled by XCR0. If EAX[1] and EAX[3] are both enumerated as 0, EBX enumerates zero.	Logical Processor
ECX[7:0]	N/A	Always returns 0 (Allocated for XCR0).	Platform
ECX[8]	PT	PT state.	Platform
ECX[9]	Reserved	Always returns 0 (Allocated for XCR0).	
ECX[10]	PASID	PASID state.	Platform
ECX[11]	CET_U	CET user state.	Platform
ECX[12]	CET_S	CET supervisor state.	Platform
ECX[13]	HDC	HDC state.	Platform
ECX[14]	UINTR	UINTR state.	Platform
ECX[15]	LBR	LBR state (only for the architectural LBR feature).	Platform
ECX[16]	HWP	HWP state.	Platform
ECX[18:17]	N/A	Always returns 0 (Allocated for XCR0).	Platform
ECX[31:19]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved	

NOTE

ECX reports the supported bits of the lower 32 bits of the IA32_XSS MSR. IA32_XSS[n] can be set to 1 only if ECX[n] is 1. EDX reports the supported bits of the upper 32 bits of the IA32_XSS MSR. IA32_XSS[n+32] can be set to 1 only if EDX[n] is 1. The details are described in Chapter 13.2, "Enumeration of CPUID Support for XSAVE Instructions and XSAVE-Supported Features," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

CPUID.0DH.SUB-LEAVES -- Sub-leaves

CPUID.0DH.n, where $n > 1$, returns information about the size and offset of each processor extended state save area within the XSAVE/XRSTOR area. Software can use the forward-extendable technique depicted below to query in bulk the valid sub-leaves and obtain size and offset information for each processor extended state save area that is supported:

```

/** For each supported feature indicated by sub-leaf 0 and 1, read the size and offset sub-leaf */
For j= 2 to 62
  If (CPUID.0DH.00H:<EDX:EAX>[j] == 1 or // Use 64-bit value of EDX:EAX
    CPUID.0DH.01H:<EDX:ECX>[j] == 1) // Use 64-bit value of EDX:ECX
    Read(CPUID.0DH.j) // Examine the size and offset.
  END IF
END FOR

```

Table 21-36. Leaf 0DH.SUB-LEAVES Processor Extended State

Register	Field Name	Description	Domain
EAX[31:0]	COMP_SIZE	The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index n.	Platform
EBX[31:0]	COMP_OFFSET	The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, n, does not map to a valid bit in the XCR0 register. If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n ($0 \leq n \leq 31$) is invalid if sub-leaf 0 returns 0 in EAX[n] and sub-leaf 1 returns 0 in ECX[n]. Sub-leaf n ($32 \leq n \leq 63$) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32]	Platform
ECX[0]	COMP_SUP	This bit is set if the bit n (corresponding to the sub-leaf index) is supported in the IA32_XSS MSR; it is clear if bit n is instead supported in XCR0.	Platform
ECX[1]	COMP_64B_ALIGNED	This bit is set if, when the compacted format of an XSAVE area is used, this extended state component located on the next 64-byte boundary following the preceding state component (otherwise, it is located immediately following the preceding state component)	Platform
ECX[2]	COMP_XFD	This bit is set to indicate support for XFD faulting.	Platform
ECX[31:3]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0EH -- Reserved

This leaf is reserved.

Table 21-37. Leaf 0EH Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0FH -- Intel® Resource Director Technology (Intel® RDT) Monitoring

CPUID.0FH returns information for the Intel Resource Director Technology Monitoring capabilities. As described below, software uses the bit vector returned in EDX by sub-leaf 00H to determine the available resource types (ResID) that can be monitored. This information is necessary for software to program the IA32_PQR_ASSOC and IA32_QM_EVTSEL MSRs such that Quality-of-Service data can be read afterwards from the IA32_QM_CTR MSR.

- This leaf is valid if CPUID.07H.00H:EBX.RDT_M[12] = 1 and MAX_LEAF ≥ 0FH.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.0FH.00H:EDX[n] == 1).

CPUID.0FH.00H -- Intel® RDT Monitoring Main Sub-Leaf

CPUID.0FH.00H returns information about Intel RDT Monitoring.

Table 21-38. Leaf 0FH.00H Intel® Resource Director Technology (Intel® RDT) Monitoring

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	MAX_RMID	Maximum range (zero-based) of RMID within this physical processor of all types.	Platform
ECX[31:0]	Reserved	Reserved.	
EDX[0]	Reserved	Reserved.	
EDX[1]	L3_MON	If 1, supports L3 Cache Intel RDT Monitoring. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.	Platform
EDX[31:2]	Reserved	Reserved.	

CPUID.0FH.00H returns information about the bit-vector representation of QoS monitoring resource types that are supported in the processor and maximum range of RMID values the processor can use to monitor of any supported resource types. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS monitoring capability available for that type.

CPUID.0FH.01H -- L3 Cache Intel® Resource Director Technology Monitoring

CPUID.0FH.01H returns information about L3 Cache Intel RDT monitoring.

Table 21-39. Leaf 0FH.01H Intel® Resource Director Technology (Intel® RDT) Monitoring

Register	Field Name	Description	Domain
EAX[7:0]	CTR_WIDTH	The counter width is encoded as an offset from 24b. A value of zero in this field indicates that 24-bit counters are supported. A value of 8 in this field indicates that 32-bit counters are supported.	Platform
EAX[8]	RDT_M_OVF	If 1, supports an overflow bit in the IA32_QM_CTR MSR (bit 61).	Platform
EAX[9]	IO_RDT_CMT	If 1, indicates the presence of non-CPU agent supporting Intel RDT CMT.	Platform
EAX[10]	IO_RDT_MBM	If 1, indicates the presence of non-CPU agent supporting Intel RDT MBM support.	Platform

EAX[31:11]	Reserved	Reserved.	
EBX[31:0]	CONV_FACTOR	Factor used to convert from reported IA32_QM_CTR value to derived occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics.	Platform
ECX[31:0]	MAX_RMID_L3	Maximum range (zero-based) of RMID of this resource type.	Platform
EDX[0]	CMT_L3_OCCUP	If 1, supports L3 occupancy monitoring.	Platform
EDX[1]	MBM_L3_TOTAL	If 1, supports L3 total bandwidth monitoring.	Platform
EDX[2]	MBM_L3_LOCAL	If 1, supports L3 local bandwidth monitoring.	Platform
EDX[31:3]	Reserved	Reserved.	

CPUID.10H -- Intel® Resource Director Technology (Intel® RDT) Allocation

CPUID.10H returns information for Intel Resource Director Technology Allocation. This leaf is valid when CPUID.07H.00H:EBX.RDT_A[15] = 1. As described below, software uses the bit vector returned in EBX by subleaf 00H to determine the available QoS Enforcement (allocation) resource types that are supported in the processor. This information is necessary for software to configure each class of services using capability bit masks in the QoS Mask registers, IA32_resourceType_Mask_n.

- This leaf is valid if CPUID.07H.00H:EBX.RDT_A[15] = 1 and MAX_LEAF ≥ 10H.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.10H.00H:EBX[n] == 1).

CPUID.10H.00H -- Intel® RDT Allocation Main Sub-Leaf

CPUID.10H.00H returns information about Intel RDT Allocation.

Table 21-40. Leaf 10H.00H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[0]	Reserved	Reserved.	
EBX[1]	CAT_L3	If 1, supports L3 Cache Allocation Technology. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX.	Platform
EBX[2]	CAT_L2	If 1, supports L2 Cache Allocation Technology.	Platform
EBX[3]	MBA	If 1, supports Memory Bandwidth Allocation.	Platform
EBX[4]	Reserved	Reserved.	
EBX[5]	CBA	If 1, supports Cache Bandwidth Allocation.	Platform
EBX[6]	RESOURCE_PRIORITY	If 1, supports Resource Priority.	Platform
EBX[31:7]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.10H.00H returns information about the bit-vector representation of QoS Enforcement resource types that are supported in the processor. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS enforcement capability available for that type.

CPUID.10H.01H -- L3 Cache Allocation Technology

CPUID.10H.ResID=1 returns information about L3 Cache Allocation Technology.

Table 21-41. Leaf 10H.01H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L3_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L3_CONTENTION	Bit-granular map of isolation/contention of allocation units.	Platform

ECX[0]	Reserved	Reserved.	
ECX[1]	CAT_L3_NONCPU	If 1, supports L3 CAT for non-CPU agents.	Platform
ECX[2]	CAT_L3_CDP	If 1, supports L3 Code and Data Prioritization Technology.	Platform
ECX[3]	CAT_L3_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous.	Platform
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L3_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.10H.02H -- L2 Cache Allocation Technology

CPUID.10H.ResID=2 returns information about L2 Cache Allocation Technology.

Table 21-42. Leaf 10H.02H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L2_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L2_CONTENTION	Bit-granular map of isolation/contention of allocation units.	Platform
ECX[1:0]	Reserved	Reserved.	
ECX[2]	CAT_L2_CDP	If 1, supports L2 Code and Data Prioritization Technology.	Platform
ECX[3]	CAT_L2_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L2_MASK_n registers do not have to be contiguous.	Platform
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L2_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.10H.03H -- Memory Bandwidth Allocation

CPUID.10H.ResID=3 returns information about Memory Bandwidth Allocation.

Table 21-43. Leaf 10H.03H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[11:0]	MBA_MAX	Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:12]	Reserved	Reserved.	

EBX[31:0]	Reserved	Reserved.	
ECX[0]	PER_THREAD_MBA	Per-thread MBA controls are supported.	Platform
ECX[1]	Reserved	Reserved.	
ECX[2]	MBA_LINEAR	If 1, the response of the delay values is linear.	Platform
ECX[31:3]	Reserved	Reserved.	
EDX[15:0]	MBA_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.10H.05H -- Cache Bandwidth Allocation

Table 21-44. Leaf 10H.05H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[7:0]	CBA_MAX_LEVELS	Reports the maximum core throttling level supported for the corresponding ResID. Add one to the return value to get the number of throttling levels supported.	Platform
EAX[11:8]	BW_SCOPE	If 1, indicates the logical processor scope of the IA32_QoS_Core_BW_Thrtl_n MSRs. Other values are reserved.	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[2:0]	Reserved	Reserved.	
ECX[3]	CBA_LINEAR	If 1, the response of the bandwidth control is approximately linear. If 0, the response of the bandwidth control is non-linear.	Platform
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CBA_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.10H.06H -- Resource Priority Control

Table 21-45. Leaf 10H.06H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[0]	THREAD_ENABLE	If 1, supports per-thread enable of RP through the IA32_RESOURCE_PRIORITY MSR.	Platform
EAX[1]	PACKAGE_ENABLE	If 1, supports physical processor package enable of RP through the IA32_RESOURCE_PRIORITY_PKG MSR.	Platform
EAX[31:2]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.11H -- Reserved

This leaf is reserved.

Table 21-46. Leaf 11H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.12H -- Intel® Software Guard Extensions (Intel® SGX) Capability

CPUID.12H returns information about Intel® SGX capabilities. More details can be found in Chapter 35, “Introduction to Intel® Software Guard Extensions,” and Chapter 36, “Enclave Access Control and Data Structures,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.

- This leaf is valid when CPUID.07H.00H:EBX.SGX[2] = 1 and MAX_LEAF ≥ 12H.
- If the leaf is valid, sub-leaf 00H and 01H are always valid. Sub-leaf n (n ≥ 2) is only valid when CPUID.12H.n:EAX[3:0] != 0.

CPUID.12H.00H -- Intel® SGX Main Sub-Leaf

CPUID.12H.00H returns information about Intel® SGX capabilities. It is only valid when CPUID.07H.00H:EBX.SGX = 1.

Table 21-47. Leaf 12H.00H Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain
EAX[0]	SGX1	If 1, supports the collection of SGX1 Leaf functions.	Platform
EAX[1]	SGX2	If 1, supports the collection of SGX2 Leaf functions.	Platform
EAX[6:2]	Reserved	Reserved.	
EAX[7]	EVERIFYREPORT2	If 1, supports the ENCLU instruction Leaf EVERIFYREPORT2.	Platform
EAX[9:8]	Reserved	Reserved.	
EAX[10]	EUPDATESVN	If 1, supports the ENCLS instruction Leaf EUPDATESVN.	Platform
EAX[11]	EDECCSSA	If 1, supports the ENCLU instruction Leaf EDECCSSA.	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	MISCSELECT	Bit vector of supported extended SGX features. The definition of MISCSELECT can be found in Section 36.7.2, “SECS.MISCSELECT Field,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.	Platform
ECX[31:0]	Reserved	Reserved.	
EDX[7:0]	MAX_ENCLAVE_SIZE_NOT_64	The maximum supported enclave size in non-64-bit mode is $2^{(EDX[7:0])}$.	Platform
EDX[15:8]	MAX_ENCLAVE_SIZE_64	The maximum supported enclave size in 64-bit mode is $2^{(EDX[15:8])}$.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.12H.01H -- Intel® SGX Attributes

CPUID.12H.01H returns information about Intel® SGX Attributes. It is only valid when CPUID.07H.00H:EBX.SGX = 1.

Table 21-48. Leaf 12H.01H Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain

EAX[31:0]	ECREATE_SECS_ATTRIBUTES_31_0	Reports the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE.	Platform
EBX[31:0]	ECREATE_SECS_ATTRIBUTES_63_32	Reports the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE.	Platform
ECX[31:0]	ECREATE_SECS_ATTRIBUTES_95_64	Reports the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE.	Platform
EDX[31:0]	ECREATE_SECS_ATTRIBUTES_127_96	Reports the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE.	Platform

The definition of the attributes can be found in Section 36.7.1, "ATTRIBUTES," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D

CPUID.12H -- $n \geq 2$ - Intel® SGX Enclave Page Cache

CPUID.12H with $ECX \geq 2$ returns information about Intel® SGX Enclave Page Cache and is supported if CPUID.07H.00H:EBX.SGX = 1.

For sub-leaves where $ECX \geq 2$, the definition of EAX[31:4], EBX, ECX, and EDX depends on the sub-leaf type listed below.

Sub-Leaf Encoding Type EAX[3:0] = 0000b (Invalid)

This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0.

CPUID.12H -- Sub-Leaf Encoding Type EAX[3:0] = 0001b

This sub-leaf enumerates an EPC section with EDX:ECX, EBX:EAX defined as follows.

Table 21-49. Leaf 12H.SUB-LEAF ENCODING TYPE EAX[3:0] = 0001B Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain
EAX[3:0]	SUB_LEAF_TYPE	Value is 0001b.	Platform
EAX[11:4]	Reserved	Reserved.	
EAX[31:12]	EPC_SECTION_ADDR_31_12	Bits 31:12 of the physical address of the base of the EPC section.	Platform
EBX[19:0]	EPC_SECTION_ADDR_51_32	Bits 51:32 of the physical address of the base of the EPC section.	Platform
EBX[31:20]	Reserved	Reserved.	

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

ECX[3:0]	EPC_SECTION_PROPERTY	EPC Section Property Encoding Definitions, as follows: 0000b – All bits in EDX:ECX are enumerated as 0. 0001b – This section has confidentiality, integrity, and replay protection. 0010b – This section has confidentiality protection only. 0011b – This section has confidentiality and integrity protection. All other encodings are reserved.	Platform
ECX[11:4]	Reserved	Reserved.	
ECX[31:12]	EPC_SECTION_SIZE_31_12	Bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory.	Platform
EDX[19:0]	EPC_SECTION_SIZE_51_32	Bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory.	Platform
EDX[31:20]	Reserved	Reserved.	

CPUID.13H -- Reserved

This leaf is reserved.

Table 21-50. Leaf 13H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.14H -- Intel® Processor Trace (Intel® PT)

CPUID.14H returns information about Intel® Processor Trace (PT).

CPUID.14H.00H returns information about Intel Processor Trace extensions.

CPUID.14H.n (n > 0 and less than the number of non-zero bits in CPUID.14H.00H:EAX) returns information about packet generation in Intel Processor Trace.

For more details on Intel PT, see Chapter 34, “Intel® Processor Trace,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.

- This leaf is valid when CPUID.07H.00H:EBX.INTEL_PROC_TRACE[25] = 1 and MAX_LEAF ≥ 14H.
- The maximum sub-leaf value for ECX is specified in CPUID.14H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.14H.00H -- Intel® PT Main Sub-Leaf

CPUID.14H.00H returns information about Intel Processor Trace extensions.

Table 21-51. Leaf 14H.00H Intel® Processor Trace (Intel® PT)

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum sub-leaf supported in leaf 14H.	Platform
EBX[0]	CR3_FILTER	If 1, supports that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed.	Platform
EBX[1]	CYC_ACC	If 1, supports Configurable PSB and Cycle-Accurate Mode.	Platform
EBX[2]	IP_FILTER	If 1, supports IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset.	Platform
EBX[3]	MTC	If 1, supports MTC timing packet and suppression of COFI-based packets.	Platform
EBX[4]	PTWRITE	If 1, supports PTWRITE. Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTw), and PTWRITE can generate packets.	Platform
EBX[5]	PWR_EVT_TRACE	If 1, supports Power Event Trace. Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation.	Platform
EBX[6]	PMI_PRESERVE	If 1, supports the PSB and PMI preservation. Writes can set IA32_RTIT_CTL[56] (InjectPsbPmiOnEn-able), enabling the processor to set IA32_RTIT_STATUS[7] (PendToPaPMI) and/or IA32_RTIT_STATUS[6] (PendPSB) in order to preserve ToPA PMIs and/or PSBs otherwise lost due to Intel PT disable. Writes can also set PendToPAPMI and PendPSB.	Platform
EBX[7]	EVENT_TRACE	If 1, supports that writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation.	Platform
EBX[8]	TNT_DIS	If 1, supports that writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation.	Platform

EBX[9]	PTTT	If 1, Processor Trace Trigger Tracing (PTTT) is supported.	Platform
EBX[31:10]	Reserved	Reserved.	
ECX[0]	TOPAOUT	If 1, supports that tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUPUT_MASK_PTRS MSRs can be accessed.	Platform
ECX[1]	MENTRY	If 1, supports that ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS.	Platform
ECX[2]	SNGL_RNG_OUT	If 1, supports the Single-Range Output scheme.	Platform
ECX[3]	TRACE_TRANSPORT_SUBSYSTEM	If 1, supports the output to Trace Transport subsystem.	Platform
ECX[30:4]	Reserved	Reserved.	
ECX[31]	LIP	If 1, the generated packets which contain IP payloads contain LIP. If 0, the generated packets which contain IP payloads contain Effective IP. Trace segments using a flat memory model will generate the same information regardless of how a logical processor reports this value since LIP=EIP.	Logical Processor
EDX[31:0]	Reserved	Reserved.	

CPUID.14H.01H -- Feature Information Sub-Leaf

CPUID.14H.01H returns information about packet generation in Intel Processor Trace.

Table 21-52. Leaf 14H.01H Intel® Processor Trace (Intel® PT)

Register	Field Name	Description	Domain
EAX[2:0]	RANGECNT	Number of configurable Address Ranges for filtering.	Platform
EAX[7:3]	Reserved	Reserved.	
EAX[10:8]	TRIGGER_CFG_CNT	Number of IA32_RTIT_TRIGGERx_CFG MSRs. The number of triggers supported is 4x this value.	Platform
EAX[15:11]	Reserved	Reserved.	
EAX[31:16]	MTC_RATE	Bitmap of supported MTC period encodings.	Platform
EBX[15:0]	CYC_THRESHOLDS	Bitmap of supported Cycle Threshold value encodings.	Platform
EBX[31:16]	PSB_RATE	Bitmap of supported Configurable PSB frequency encodings.	Platform
ECX[0]	ICNT	If 1, the trigger action EN_ICNT is supported.	Platform
ECX[1]	TRIGGER_PAUSE	If 1, the trigger actions TRACE_PAUSE and TRACE_RESUME are supported.	Platform
ECX[14:2]	Reserved	Reserved.	

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

ECX[15]	TRIGGER_DR_MATCH	If 1, trigger input DR match is supported.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.15H -- Time Stamp Counter and Nominal Core Crystal Clock

CPUID.15H returns information about the Time Stamp Counter and the Nominal Core Crystal Clock.

- This leaf is valid if MAX_LEAF $\geq$ 15H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-53. Leaf 15H Time Stamp Counter and Nominal Core Crystal Clock

Register	Field Name	Description	Domain
EAX[31:0]	DENOMINATOR	An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.	Platform
EBX[31:0]	NUMERATOR	An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio. If 0, the TSC/"core crystal clock" ratio is not enumerated.	Platform
ECX[31:0]	NOMINAL_ART_FREQUENCY	An unsigned integer which is the nominal frequency of the core crystal clock in Hz. If 0, the nominal core crystal clock frequency is not enumerated. Note, the core crystal clock may differ from the reference clock, bus clock or core clock frequencies.	Platform
EDX[31:0]	Reserved	Reserved.	

Dividing EBX[31:0] by EAX[31:0] provides the ratio of the TSC frequency to the core crystal clock frequency. The Timestamp Counter frequency is computed by multiplying that ratio by ECX[31:0], such as $\text{TSC\_frequency} = \text{ECX} * \text{EBX}/\text{EAX}$.

CPUID.16H -- Processor Frequency Information

CPUID.16H returns information about processor frequency information.

Data is returned from this interface in accordance with the processor's specification and does not reflect actual values. Suitable use of this data includes the display of processor information in like manner to the processor brand string and for determining the appropriate range to use when displaying processor information e.g. frequency history graphs. The returned information should not be used for any other purpose as the returned information does not accurately correlate to information / counters returned by other processor interfaces.

While a processor may support the Processor Frequency Information leaf, fields that return a value of zero are not supported.

- This leaf is valid if MAX_LEAF $\geq$ 16H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-54. Leaf 16H Processor Frequency Information

Register	Field Name	Description	Domain
EAX[15:0]	PROCESSOR_BASE_FREQUENCY	Processor Base Frequency (in MHz).	Logical Processor
EAX[31:16]	Reserved	Reserved.	
EBX[15:0]	MAXIMUM_FREQUENCY	Maximum Frequency (in MHz).	Logical Processor
EBX[31:16]	Reserved	Reserved.	
ECX[15:0]	BUS_FREQUENCY	Bus (Reference) Frequency (in MHz).	Logical Processor
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.17H -- System-on-Chip Vendor Attribute

CPUID.17H returns System-on-Chip vendor attribute information.

- This leaf is valid if CPUID.17H.00H:EAX[31:0] (MaxSOCID_Index) ≥ 3 and MAX_LEAF ≥ 17 H.
- The maximum sub-leaf value for ECX is specified in CPUID.17H.00H:EAX[31:0] MaxSOCID_Index.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.17H.00H -- Main Sub-Leaf

CPUID.17H.00H returns System-on-Chip vendor attribute information.

Table 21-55. Leaf 17H.00H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SOCID_INDEX	Reports the maximum input value of supported Sub-leaf in Leaf 17H.	Platform
EBX[15:0]	SOC_VENDOR_ID	SOC Vendor ID.	Platform
EBX[16]	IS_VENDOR_SCHEME	If 1, the SOC Vendor ID field is assigned via an industry standard enumeration scheme. Otherwise, the SOC Vendor ID field is assigned by Intel.	Platform
EBX[31:17]	Reserved	Reserved.	
ECX[31:0]	PROJECT_ID	A unique number an SOC vendor assigns to its SOC projects.	Platform
EDX[31:0]	STEPPING_ID	A unique number within an SOC project that an SOC vendor assigns.	Package

CPUID.17H.01H -- Vendor Brand String Sub-Leaf (Bytes 0 to 15)

CPUID.17H with ECX=1, 2, or 3 returns information for System-on-Chip vendor branding string.

SOC Vendor Brand String is a UTF-8 encoded string padded with trailing bytes of 00H. The complete SOC Vendor Brand String is constructed by concatenating in ascending order the output of each sub-leaf 1 to 3. For the output of each sub-leaf, byte are ordered in ascending order of EAX:EBX:ECX:EDX.

Leaf 17H sub-leaves 4 and above are reserved.

Table 21-56. Leaf 17H.01H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_0_to_3	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_4_to_7	SOC Vendor Brand String. UTF-8 encoded string.	Platform
ECX[31:0]	VENDOR_BRAND_STRING_BYTES_8_to_11	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_12_to_15	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.02H -- Vendor Brand String Sub-Leaf (Bytes 16 to 31)

Table 21-57. Leaf 17H.02H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_16_to_19	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_20_to_23	SOC Vendor Brand String. UTF-8 encoded string.	Platform

ECX[31:0]	VENDOR_BRAND_STRING_BYTES_24_to_27	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_28_to_31	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.03H -- Vendor Brand String Sub-Leaf (Bytes 32 to 47)

Table 21-58. Leaf 17H.03H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_32_to_35	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_36_to_39	SOC Vendor Brand String. UTF-8 encoded string.	Platform
ECX[31:0]	VENDOR_BRAND_STRING_BYTES_40_to_43	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_44_to_47	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.M>MAXSOCID_INDEX—RESERVED SUB-LEAVES -- m>MaxSOCID_Index—Reserved Sub-Leaves

CPUID.17H with ECX > MaxSOCID_Index is reserved and returns all zeroes.

Table 21-59. Leaf 17H.M>MAXSOCID_INDEX—RESERVED SUB-LEAVES System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved	
EBX[31:0]	Reserved	Reserved	
ECX[31:0]	Reserved	Reserved	
EDX[31:0]	Reserved	Reserved	

CPUID.18H -- Deterministic Address Translation Parameters

CPUID.18H returns information about the Deterministic Address Translation Parameters. Each sub-leaf enumerates a different address translation structure.

- This leaf is valid if CPUID.18H.00H:EAX[31:0] $\neq 0$ and MAX_LEAF $\geq 18H$.
- The maximum sub-leaf value for ECX is specified in CPUID.18H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX. A sub-leaf index is also invalid if EDX[4:0] returns 0.
- Valid sub-leaves do not need to be contiguous or in any particular order. A valid sub-leaf may be in a higher input ECX value than an invalid sub-leaf or than a valid sub-leaf of a higher or lower-level structure.

CPUID.18H.00H -- Main Sub-Leaf

Deterministic Address Translation Parameters Main Leaf

Table 21-60. Leaf 18H.00H Deterministic Address Translation Parameters

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value of supported sub-leaf in leaf 18H.	Logical Processor
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[4:0]	TYPE	Will always return 0.	Logical Processor
EDX[31:5]	Reserved	Reserved.	

CPUID.18H.ECX ≥ 1 -- Sub-Leaves

CPUID.18H with ECX ≥ 1 returns Deterministic Address Translation Parameters information.

Table 21-61. Leaf 18H.ECX ≥ 1 Deterministic Address Translation Parameters

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[0]	4KB_ENTRIES	If 1, supports 4K page size entries in this structure.	Logical Processor
EBX[1]	2MB_ENTRIES	If 1, supports 2MB page size entries in this structure.	Logical Processor
EBX[2]	4MB_ENTRIES	If 1, supports 4MB page size entries in this structure.	Logical Processor
EBX[3]	1GB_ENTRIES	If 1, supports 1 GB page size entries in this structure.	Logical Processor
EBX[7:4]	Reserved	Reserved.	
EBX[10:8]	PARTITIONING	Partitioning (0:Soft partitioning between the logical processors sharing this structure).	Logical Processor
EBX[15:11]	Reserved	Reserved.	
EBX[31:16]	NUM_WAYS	W = ways of associativity.	Logical Processor
ECX[31:0]	NUM_SETS	S = number of sets.	Logical Processor

EDX[4:0]	TYPE	00000b: Null (indicates this Sub-leaf is not valid). 00001b: Data TLB. 00010b: Instruction TLB. 00011b: Unified TLB.1 00100b: Load Only TLB. Hit on loads; fills on both loads and stores. 00101b: Store Only TLB. Hit on stores; fill on stores. All other encodings are reserved. Some unified TLBs will allow a single TLB entry to satisfy data read/write and instruction fetches. Others will require separate entries (e.g., one loaded on data read/write and another loaded on an instruction fetch). See the Intel® 64 and IA-32 Architectures Optimization Reference Manual for details of a particular product.	Logical Processor
EDX[7:5]	LEVEL_NUM	Translation cache level (starts at 1).	Logical Processor
EDX[8]	FULLY_ASSOC	Fully associative structure.	Logical Processor
EDX[13:9]	Reserved	Reserved.	
EDX[25:14]	MAX_LP_ADDRESSABLE_IDS	Maximum number of addressable IDs for logical processors sharing this translation cache. Add one to the return value to get the result.	Logical Processor
EDX[31:26]	Reserved	Reserved.	

CPUID.19H -- Key Locker

CPUID.19H returns Key Locker information.

- This leaf is valid if CPUID.07H.00H:ECX.KEY_LOCKER[23] = 1 and MAX_LEAF ≥ 19H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-62. Leaf 19H Key Locker

Register	Field Name	Description	Domain
EAX[0]	CPLD_RESTRICT	If 1, supports the Key Locker restriction of CPLD-only. ¹	Platform
EAX[1]	NO_ENCRYPT_RESTRICT	If 1, supports the Key Locker restriction of no-encrypt. ¹	Platform
EAX[2]	NO_DECRYPT_RESTRICT	If 1, supports the Key Locker restriction of no-decrypt. ¹	Platform
EAX[31:3]	Reserved	Reserved.	
EBX[0]	AESKLE	If 1, the AES Key Locker instructions are fully enabled. CPUID.19H:EBX.AESKLE[0] is enumerated as 1 if the AES Key Locker instructions have been activated by system firmware and CR4.KL[bit 19] = 1. Software can check this bit after setting CR4.KL to determine whether AES Key Locker instructions have been enabled. Note that some processors may allow enabling of those instructions without activation by system firmware. Some processors may not support the use of AES Key Locker instructions in system-management-mode (SMM). Those processors enumerate CPUID.19H:EBX.AESKLE[0] as 0 in SMM regardless of the setting of CR4.KL.	Logical Processor
EBX[1]	Reserved	Reserved.	
EBX[2]	AES_WIDE	If 1, supports the AES wide Key Locker instructions. ¹	Platform
EBX[3]	Reserved	Reserved.	
EBX[4]	IWKEYBACKUP	If 1, supports the Key Locker MSRs (IA32_COPY_LOCAL_TO_PLATFORM, IA23_COPY_PLATFORM_TO_LOCAL, IA32_COPY_STATUS, and IA32_IWKEYBACKUP_STATUS) and backing up the internal wrapping key. ¹	Platform
EBX[31:5]	Reserved	Reserved.	
ECX[0]	NOBACKUP	If 1, supports the NoBackup parameter to LOADIWKEY. ¹	Platform
ECX[1]	RAND_IWKEY	If 1, supports KeySource encoding of 1 (randomization of the internal wrapping key). ¹	Platform
ECX[31:2]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

NOTE

1. This field is valid only if CPUID.19H:EBX.AESKLE[0] = 1.

CPUID.1AH -- Native Model ID Enumeration

CPUID.1AH returns Native Model ID information.
This leaf exists on all logical processors in a hybrid package, it may also be present in other processor configurations.

- This leaf is valid if CPUID.1AH.00H:EAX[31:0] <> 0 and MAX_LEAF ≥ 1AH.
- The only valid sub-leaf is 0 and ECX must be set to 0.

Table 21-63. Leaf 1AH Native Model ID Enumeration

Register	Field Name	Description	Domain
EAX[23:0]	CORE_NATIVE_MODEL_ID	The core-type and native model ID can be used to uniquely identify the microarchitecture of the core. This native model ID is not unique across core types, and not related to the model ID reported in CPUID.01H, and does not identify the SOC.	Logical Processor
EAX[31:24]	CORE_TYPE	10H: Reserved 20H: Intel® Atom® 30H: Reserved 40H: Intel® Core The core type may only be used as an identification of the microarchitecture for this logical processor and its numeric value has no significance, neither large nor small. This field neither implies nor expresses any other attribute to this logical processor and software should not assume any.	Logical Processor
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1BH -- PCONFIG Information

CPUID.1BH -- Output Registers Format for All Sub-Leaves

CPUID.1BH returns information for PCONFIG capabilities. This information is enumerated in sub-leaves selected by the value of ECX (starting with 0).

- This leaf is valid if CPUID.07H.00H:EDX.PCONFIG[18] = 1 and MAX_LEAF ≥ 1BH.
- Sub-leaves are enumerated until sub-leaf n, where EAX[11:0] returns 0.

Table 21-64. Leaf 1BH PCONFIG Information

Register	Field Name	Description	Domain
EAX[11:0]	SUB_LEAF_TYPE	0 (Invalid)	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

Each sub-leaf of CPUID.1BH enumerates its sub-leaf type in EAX. If a sub-leaf type is 0, the sub-leaf is invalid and zero is returned in EBX, ECX, and EDX. In this case, all subsequent sub-leaves (selected by larger input values of ECX) are also invalid.

The only valid sub-leaf type currently defined is 1, indicating that the sub-leaf enumerates target identifiers for the PCONFIG instruction. Any non-zero value returned in EBX, ECX, or EDX indicates a valid target identifier of the PCONFIG instruction (any value of zero should be ignored). The only target identifier currently defined is 1, indicating TME-MK. See the “PCONFIG—Platform Configuration” instruction in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B, for more information.

CPUID.1BH.OUTPUT REGISTERS FOR SUB-LEAVE TYPE TARGET IDENTIFIER (1) -- Output Registers for Sub-Leave Type Target Identifier (1)

Table 21-65. Leaf 1BH.OUTPUT REGISTERS FOR SUB-LEAVE TYPE TARGET IDENTIFIER (1) PCONFIG Information

Register	Field Name	Description	Domain
EAX[11:0]	SUB_LEAF_TYPE	1 (Target Identifier)	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	TARGET_IDENTIFIER_1	Target identifier	Platform
ECX[31:0]	TARGET_IDENTIFIER_2	Target identifier	Platform
EDX[31:0]	TARGET_IDENTIFIER_3	Target identifier	Platform

The only current valid target identifier is 1 for MK-TME.

CPUID.1CH -- Last Branch Records (LBR) Information

CPUID.1CH returns information about architectural Last Branch Records (LBR). For details on LBR, see Chapter 20, “Last Branch Records,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B.

- This leaf is valid if CPUID.07H.00H:EDX.ARCH_LBRS[19] = 1 and MAX_LEAF ≥ 1CH.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-66. Leaf 1CH Last Branch Records (LBR) Information

Register	Field Name	Description	Domain
EAX[7:0]	LBR_DEPTH_VALUES	For each bit <i>n</i> set in this field, the IA32_LBR_DEPTH.DEPTH value $8 \cdot (n+1)$ is supported.	Platform
EAX[29:8]	Reserved	Reserved.	
EAX[30]	DEEP_C_STATE_RESET	If 1, supports that LBRs may be cleared on an MWAIT that requests a C-state numerically greater than C1.	Platform
EAX[31]	IP_VALUES_CONTAIN_LIP	If 1, the LBR IP values contain LIP. If 0, IP values contain Effective IP. Trace segments using a flat memory model will generate the same information regardless of how a logical processor reports this value since LIP = EIP	Logical Processor
EBX[0]	CPL_FILTERING	If 1, supports setting IA32_LBR_CTL[2:1] to a non-zero value.	Platform
EBX[1]	BRANCH_FILTERING	If 1, supports setting IA32_LBR_CTL[22:16] to a non-zero value.	Platform
EBX[2]	CALL_STACK_MODE	If 1, supports setting IA32_LBR_CTL[3] to 1.	Platform
EBX[31:3]	Reserved	Reserved.	
ECX[0]	MISPREDICT_BIT	If 1, IA32_LBR_x_INFO[63] holds indication of branch misprediction (MISPRED).	Platform
ECX[1]	TIMED_LBRS	If 1, IA32_LBR_x_INFO[15:0] holds CPU cycles since last LBR entry (CYC_CNT), and IA32_LBR_x_INFO[60] holds an indication of whether the value held there is valid (CYC_CNT_VALID).	Platform
ECX[2]	BRANCH_TYPE_FIELD_SUPPORTED	If 1, IA32_LBR_INFO_x[59:56] holds indication of the recorded operation’s branch type (BR_TYPE).	Platform
ECX[15:3]	Reserved	Reserved.	
ECX[19:16]	EVENT_LOGGING_BITMAP	The event logging bitmap, wherein each set bit corresponds to a programmable performance monitoring counter that supports LBR event logging.	Platform
ECX[31:20]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1DH -- Tile Information

CPUID.1DH returns information about tile architecture and tile palette 1 (see Chapter 19, “Programming with Intel® Advanced Matrix Extensions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).

- This leaf is valid if CPUID.07H.00H:EDX.AMX_TILE[24] = 1 and MAX_LEAF ≥ 1DH.
- The maximum sub-leaf value for ECX is specified in CPUID.1DH.00H:EAX[31:0] max_palette.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.1DH.00H -- Tile Information Main Sub-Leaf

CPUID.1DH.00H returns the tile architecture information.

Table 21-67. Leaf 1DH.00H Tile Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_PALETTE	Highest numbered palette sub-leaf. Value = 1.	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1DH.01H -- Tile Palette 1

CPUID.1DH.01H returns tile palette information.

Table 21-68. Leaf 1DH.01H Tile Information

Register	Field Name	Description	Domain
EAX[15:0]	TOTAL_TILE_BYTES	Palette 1 total_tile_bytes. Value = 8192.	Platform
EAX[31:16]	BYTES_PER_TILE	Palette 1 bytes_per_tile. Value = 1024.	Platform
EBX[15:0]	BYTES_PER_ROW	Palette 1 bytes_per_row. Value = 64.	Platform
EBX[31:16]	MAX_NAMES	Palette 1 max_names (number of tile registers). Value = 8.	Platform
ECX[15:0]	MAX_ROWS	Palette 1 max_rows. Value = 16.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1EH -- TMUL Information

CPUID.1EH returns information about TMUL capabilities (see Chapter 19, “Programming with Intel® Advanced Matrix Extensions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).

- This leaf is valid if CPUID.07H.00H:EDX.AMX_TILE[24] = 1 and MAX_LEAF ≥ 1EH.
- The only valid sub-leaf is 0 and ECX must be set to 0.

CPUID.1EH.00H -- TMUL Information Main Leaf

TMUL Main Leaf Information

Table 21-69. Leaf 1EH.00H TMUL Information

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[7:0]	TMUL_MAXK	tmul_maxk (rows or columns). Value = 16.	Platform
EBX[23:8]	TMUL_MAXN	tmul_maxn (column bytes). Value = 64.	Platform
EBX[31:24]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1FH -- V2 Extended Topology

CPUID.1FH returns information about V2 Extended Topology.

CPUID.1FH is a preferred superset to leaf 0BH. Intel recommends using leaf 1FH when available rather than leaf 0BH and ensuring that any leaf 0BH algorithms are updated to support leaf 1FH.

- This leaf is valid if CPUID.1FH.00H:EBX[15:0] $\neq$ 0 and MAX_LEAF $\geq$ 1FH.
 - When the leaf is invalid, CPUID.1FH.00H:ECX.DOMAIN_TYPE[15:8] will report the Domain Type ID as Invalid (0).
- The sub-leaves are enumerated until sub-leaf n returns 0 in EBX[15:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EBX[15:0] as 0.

CPUID.1FH -- ECX $\geq$ 0

Table 21-70. Leaf 1FH V2 Extended Topology

Register	Field Name	Description	Domain
EAX[4:0]	SHIFT_COUNT	The number of bits that the x2APIC ID must be shifted to the right to address instances of the next higher-scoped domain. When logical processor is not supported by the processor, the value of this field at the Logical Processor domain sub-leaf may be returned as either 0 (no allocated bits in the x2APIC ID) or 1 (one allocated bit in the x2APIC ID); software should plan accordingly.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[15:0]	NEXT_LEVEL_NUM_LP	The number of logical processors across all instances of this domain within the next higher-scoped domain relative to this current logical processor. (For example, in a processor socket/package comprising “M” dies of “N” cores each, where each core has “L” logical processors, the “die” domain sub-leaf value of this field would be M*N*L. In an asymmetric topology this would be the summation of the value across the lower domain level instances to create each upper domain level instance.) This number reflects configuration as shipped by Intel. Note that the number of logical processors can be asymmetric in which case “L” may be different on different logical processors, as an example a core with 2 logical processors on the same platform as a core with 1 logical processor. Note, software must not use this field to enumerate processor topology. Software must not use the value of EBX[15:0] to enumerate processor topology of the system. The value is only intended for display and diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.	Logical Processor
EBX[31:16]	Reserved	Reserved.	

ECX[7:0]	LEVEL_NUM	The input ECX sub-leaf index.	Platform
ECX[15:8]	DOMAIN_TYPE	This field provides an identification value which indicates the domain as shown in the table. Although domains are ordered, their assigned identification values are not and software should not depend on it. (For example, if a new domain between core and module is specified, it will have an identification value higher than 5.) See the table below for the current list of valid enumerations. Note that enumeration values of 0 and 7-255 are reserved.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	X2APIC_ID	The x2APIC ID of the current logical processor is always valid and does not vary with the sub-leaf index in ECX.	Logical Processor

The sub-leaves of CPUID.1FH describe an ordered hierarchy of logical processors starting from the smallest scoped domain of a Logical Processor (sub-leaf index 0) to the Core domain (sub-leaf index 1) to the largest scoped domain (the last valid sub-leaf index) that is implicitly subordinate to the unenumerated highest-scoped domain of the processor package (socket).

The details of each valid domain is enumerated by a corresponding sub-leaf. Details for a domain include its type and how all instances of that domain determine the number of logical processors and x2 APIC ID partitioning at the next higher-scoped domain. The ordering of domains within the hierarchy is fixed architecturally as shown below. For a given processor, not all domains may be relevant or enumerated; however, the logical processor and core domains are always enumerated. As an example, a processor may report an ordered hierarchy consisting only of "Logical Processor," "Core," and "Die."

For two valid sub-leaves N and N+1, sub-leaf N+1 represents the next immediate higher-scoped domain with respect to the domain of sub-leaf N for the given processor.

If sub-leaf index "N" returns an invalid domain type in ECX[15:08] (00H), then all sub-leaves with an index greater than "N" also return an invalid domain type. A sub-leaf returning an invalid domain always returns 0 in EAX and EBX.

Table 21-71. Hierarchy of Valid Domain Enumerations in CPUID.1FH:ECX[15:8]

Hierarchy	Domain	Domain Type ID Value
Invalid	Invalid	0
Lowest	Logical Processor	1
...	Core	2
...	Module	3
...	Tile	4
...	Die	5
...	DieGrp	6
Highest	Package/Socket	(Implied)
Reserved	Reserved	7-255

CPUID.20H -- Processor History Reset Information

CPUID.20H returns information about processor history reset when CPUID.07H.01H:EAX.HRESET[22] = 1.

- This leaf is valid if CPUID.07H.01H:EAX.HRESET[22] = 1 and MAX_LEAF ≥ 20H.
- The maximum sub-leaf value for ECX is specified in CPUID.20H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.20H.00H -- Processor History Reset Sub-leaf

Table 21-72. Leaf 20H.00H Processor History Reset Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum number of sub-leaves that are supported in leaf 20H.	Platform
EBX[0]	THREAD_DIRECTOR_HRESET	Indicates support for both HRESET's EAX[0] parameter, and IA32_HRESET_ENABLE[0] set by the OS to enable reset of Intel® Thread Director history.	Platform
EBX[31:1]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.21H -- Unimplemented

Does not return feature information for the processor. Allocated for use by TDX modules; see Intel® Trust Domain Extensions (Intel® TDX) Module Base Architecture Specification. Software emulating CPUID should not change the information returned for this leaf.

CPUID.22H -- Reserved

This leaf is reserved.

Table 21-73. Leaf 22H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H -- Architectural Performance Monitoring Extended

CPUID.23H returns architectural performance monitoring extended information.

- This leaf is valid if CPUID.07H.01H:EAX.ARCH_PERFMON_EXT[8] = 1 and MAX_LEAF ≥ 23H.
- The sub-leaves of this leaf are enumerated by a bitmask specified in CPUID.23H.00H:EAX[31:0] SUBLEAF_MASK. The bit numbers of set bits in the bitmask represent valid sub-leaf indexes.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index is invalid if the index as a bit number is clear in the Available Sub-Leaf Mask or is greater than 31.

CPUID.23H.00H -- Main Sub-Leaf

Table 21-74. Leaf 23H.00H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	SUBLEAF_MASK	If bit n is set, sub-leaf n is supported. (For unsupported sub-leaves, 0 is returned in the registers EAX, EBX, ECX, and EDX.)	Logical Processor
EBX[0]	UNITMASK2	If 1, supports the UnitMask2 field in the IA32_PERFEVTSELx MSRs.	Logical Processor
EBX[1]	EQ	If 1, supports the equal flag in the IA32_PERFEVTSELx MSRS.	Logical Processor
EBX[31:2]	Reserved	Reserved.	
ECX[7:0]	SLOTS_PER_CYC	If this field is non-zero, it represents the number of Top-down Microarchitecture Analysis (TMA) slots per cycle. This number can be multiplied by the number of cycles (from CPU_CLK_UNHALTED.THREAD / CPU_CLK_UNHALTED.CORE or IA32_FIXED_CTR1) to determine the total number of slots. If this field is zero, IA32_FIXED_CTR3 should be used to determine the total number of slots.	Logical Processor
ECX[31:8]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.01H -- Counter Information Sub-Leaf

Table 21-75. Leaf 23H.01H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	GP_COUNTERS	For each bit n set in this field, the processor supports general-purpose performance monitoring counter n.	Logical Processor
EBX[31:0]	FIXED_COUNTERS	For each bit m set in this field, the processor supports fixed-function performance monitoring counter m. The valid range of fixed-function counters is 0 through 15.	Logical Processor
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.02H -- Bitmap of Auto Counter Reload Sub-Leaf

Table 21-76. Leaf 23H.02H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	ACR_GP_RELOAD	General counters that can be reloaded. For each bit n set in this field, the processor supports ACR for general-purpose performance monitoring counter n.	Logical Processor
EBX[31:0]	ACR_FIXED_RELOAD	Fixed counters that can be reloaded. For each bit m set in this field, the processor supports ACR for fixed-function performance monitoring counter m.	Logical Processor
ECX[31:0]	ACR_GP_TRIGGER	General counters that can cause reloads. For each bit y set in this field, the processor allows general-purpose performance monitoring counter y to reload all existing general-purpose performance monitoring counters capable of being reloaded.	Logical Processor
EDX[31:0]	ACR_FIXED_TRIGGER	Fixed counters that can cause reloads. For each bit x set in this field, the processor allows fixed-function performance monitoring counter x to reload all existing fixed-function performance monitoring counters capable of being reloaded.	Logical Processor

CPUID.23H.03H -- Architectural Performance Monitoring Events Bitmap Sub-Leaf

For each bit n set in this field, the processor supports Architectural Performance Monitoring Event of index n.

Table 21-77. Leaf 23H.03H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[0]	CORE_CYC	If 1, supports architectural index 0.	Logical Processor
EAX[1]	INSTR_RET	If 1, supports architectural index 1.	Logical Processor
EAX[2]	REF_CYC	If 1, supports architectural index 2.	Logical Processor
EAX[3]	LLC_REF	If 1, supports architectural index 3.	Logical Processor
EAX[4]	LLC_MISSES	If 1, supports architectural index 4.	Logical Processor
EAX[5]	BR_INSTR_RET	If 1, supports architectural index 5	Logical Processor
EAX[6]	BR_MISPRED_RET	If 1, supports architectural index 6	Logical Processor
EAX[7]	SLOTS	If 1, supports architectural index 7	Logical Processor
EAX[8]	BACKEND	If 1, supports architectural index 8	Logical Processor
EAX[9]	BADSPEC	If 1, supports architectural index 9	Logical Processor

EAX[10]	FRONTEND	If 1, supports architectural index 10	Logical Processor
EAX[11]	RETIRING	If 1, supports architectural index 11	Logical Processor
EAX[12]	LBR_INSERTS	If 1, supports architectural index 12	Logical Processor
EAX[31:13]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.04H -- PEBS Capabilities

Table 21-78. Leaf 23H.04H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[2:0]	Reserved	Reserved.	
EBX[3]	ALLOW_IN_RECORD	If 1, indicates that the ALLOW_IN_RECORD bit is available in the IA32_PMC_GpN_CFG_C and IA32_PMC_Fxm_CFG_C MSRs.	Logical Processor
EBX[4]	CNTR_GP	If 1, indicates that counters group sub-group general-purpose counters is available.	Logical Processor
EBX[5]	CNTR_FIXED	If 1, indicates that counters group sub-group fixed-function counters is available.	Logical Processor
EBX[6]	CNTR_METRICS	If 1, indicates that counters group sub-group performance metrics is available.	Logical Processor
EBX[7]	Reserved	Reserved.	
EBX[9:8]	LBR	LBR group and both bits [41:40] are available.	Logical Processor
EBX[15:10]	Reserved	Reserved.	
EBX[23:16]	XER	XER group bits [50:49] and bits [55:53] are available. See Section 11.4.4, "XSAVEEnabled Registers Group," for XER fields.	Logical Processor
EBX[28:24]	Reserved	Reserved.	
EBX[29]	GPR	If 1, the GPR group is available.	Logical Processor
EBX[30]	AUX	If 1, the AUX group is available.	Logical Processor
EBX[31]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.05H -- Arch PEBS GP and Fixed Counters supported

Table 21-79. Leaf 23H.05H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	GP_PEBS	Bit vector of general-purpose counters for which the Architectural PEBS mechanism is available (bit n == GP counter #n). If EAX[n] == 1, then the IA32_PMC_GPn_CFG_C MSR is available, and PEBS is supported on that counter; the PEBS_EN[63] field can be set; and the RELOAD[31:0] field can be set. Note that CPUID.23H.04H:EBX governs which adaptive group bits can be set.	Logical Processor
EBX[31:0]	GP_PDIST	General-purpose counters for which PEBS supports PDIST.	Logical Processor
ECX[31:0]	FIXED_PEBS	Bit vector of fixed-function counters for which the Architectural PEBS mechanism is available. If ECX[x] == 1, then the IA32_PMC_FXm_CFG_C MSR is available, and PEBS is supported; the PEBS_EN[63] field can be set; and the RELOAD[31:0] field can be set. Note that CPUID.23H.04H:EBX governs which adaptive group bits can be set.	Logical Processor
EDX[31:0]	FIXED_PDIST	Fixed-function counters for which PEBS supports PDIST.	Logical Processor

CPUID.24H -- Converged Vector ISA

When CPUID.24H, the processor returns Intel AVX10 converged vector ISA information. This leaf is supported when CPUID.07H.01H:EDX.AVX10[19] = 1.

- This leaf is valid if CPUID.07H.01H:EDX.AVX10[19] = 1 and MAX_LEAF ≥ 24H.
- The maximum sub-leaf value for ECX is specified in CPUID.24H.00H.EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.24H.00H -- Converged Vector ISA Main Sub-Leaf

Table 21-80. Leaf 24H.00H Converged Vector ISA

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum number of sub-leaves that are supported in leaf 24H.	Platform
EBX[7:0]	VECTOR_ISA_VERSION	Reports the Intel® AVX10 Converged Vector ISA version.	Platform
EBX[15:8]	Reserved	Reserved.	
EBX[18:16]	Reserved at 111	Always 111b. Earlier versions of this specification documented these bits as enumerating support for different vector lengths. Processors enumerating Intel® AVX10 support all vector lengths.	Platform
EBX[31:19]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.27H -- Intel® Resource Director Technology (Intel® RDT) Asymmetric Monitoring

CPUID.27H returns information for the Intel Resource Director Technology Monitoring capabilities with asymmetric topology.

As described below, software uses the bit vector returned in EDX by sub-leaf 00H to determine the available resource types (ResID) that can be monitored. This information is necessary for software to program the IA32_PQR_ASSOC and IA32_QM_EVTSEL MSRs such that Quality-of-Service data can be read afterwards from the IA32_QM_CTR MSR.

- This leaf is valid if CPUID.07H.01H:ECX.RDT_M_ASYM[0] = 1 and MAX_LEAF ≥ 27H.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.27H.00H:EDX[n] == 1).
- This leaf must be read on each logical processor to determine the support on each processor.

CPUID.27H.00H -- Intel® RDT Asymmetric Monitoring Main Sub-Leaf

CPUID.27H.00H returns information about Intel RDT Monitoring Asymmetric.

Table 21-81. Leaf 27H.00H Intel® Resource Director Technology (Intel® RDT) Asymmetric Monitoring

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	MAX_RMID	Maximum range (zero-based) of RMID within this physical processor of all types.	Logical Processor
ECX[31:0]	Reserved	Reserved.	
EDX[0]	Reserved	Reserved.	
EDX[1]	L3_MON	If 1, supports L3 Cache Intel RDT Monitoring. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.	Logical Processor
EDX[31:2]	Reserved	Reserved.	

CPUID.27H.00H returns information about the bit-vector representation of QoS monitoring resource types that are supported in the processor and maximum range of RMID values the processor can use to monitor of any supported resource types. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS monitoring capability available for that type.

CPUID.27H.01H -- L3 Cache Intel® Resource Director Technology Asymmetric Monitoring

CPUID.27H.01H returns information about L3 Cache Intel RDT monitoring asymmetric.

Table 21-82. Leaf 27H.01H Intel® Resource Director Technology (Intel® RDT) Asymmetric Monitoring

Register	Field Name	Description	Domain
EAX[7:0]	CTR_WIDTH	The counter width is encoded as an offset from 24b. A value of zero in this field indicates that 24-bit counters are supported. A value of 8 in this field indicates that 32-bit counters are supported.	Logical Processor
EAX[8]	RDT_M_OVF	If 1, supports an overflow bit in the IA32_QM_CTR MSR (bit 61).	Logical Processor

EAX[9]	IO_RDT_CMT	If 1, indicates the presence of non-CPU agent supporting Intel RDT CMT.	Logical Processor
EAX[10]	IO_RDT_MBM	If 1, indicates the presence of non-CPU agent supporting Intel RDT MBM support.	Logical Processor
EAX[31:11]	Reserved	Reserved.	
EBX[31:0]	CONV_FACTOR	Factor used to convert from reported IA32_QM_CTR value to derived occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics.	Logical Processor
ECX[31:0]	MAX_RMID_L3	Maximum range (zero-based) of RMID of this resource type.	Logical Processor
EDX[0]	CMT_L3_OCCUP	If 1, supports L3 occupancy monitoring.	Logical Processor
EDX[1]	MBM_L3_TOTAL	If 1, supports L3 total bandwidth monitoring.	Logical Processor
EDX[2]	MBM_L3_LOCAL	If 1, supports L3 local bandwidth monitoring.	Logical Processor
EDX[31:3]	Reserved	Reserved.	

CPUID.28H -- Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

CPUID.28H returns information for Intel Resource Director Technology Allocation with asymmetric topology. This leaf is valid when CPUID.07H.01H:ECX.RDT_A_SYM[1] = 1. As described below, software uses the bit vector returned in EBX by subleaf 00H to determine the available QoS Enforcement (allocation) resource types that are supported in the processor. This information is necessary for software to configure each class of services using capability bit masks in the QoS Mask registers, IA32_resourceType_Mask_n.

- This leaf is valid if CPUID.07H.01H:ECX.RDT_A_SYM[1] = 1 and MAX_LEAF ≥ 28H.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.28H.00H:EBX[n] == 1).

CPUID.28H.00H -- Intel® RDT Asymmetric Allocation Main Sub-Leaf

CPUID.28H.00H returns information about Intel RDT Allocation Asymmetric.

Table 21-83. Leaf 28H.00H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[0]	Reserved	Reserved.	
EBX[1]	CAT_L3	Supports L3 Cache Allocation Technology if 1.	Logical Processor
EBX[2]	CAT_L2	Supports L2 Cache Allocation Technology if 1.	Logical Processor
EBX[3]	MBA	Supports Memory Bandwidth Allocation if 1.	Logical Processor
EBX[4]	Reserved	Reserved.	
EBX[5]	CBA	If 1, supports Cache Bandwidth Allocation.	Logical Processor
EBX[6]	RESOURCE_PRIORITY	If 1, supports Resource Priority.	Platform
EBX[31:7]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.28H.00H returns information about the bit-vector representation of QoS Enforcement resource types that are supported in the processor. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS enforcement capability available for that type.

CPUID.28H.01H -- Asymmetric L3 Cache Allocation Technology

CPUID.28H.ResID=1 returns information about Asymmetric L3 Cache Allocation Technology.

Table 21-84. Leaf 28H.01H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L3_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Logical Processor
EAX[31:5]	Reserved	Reserved.	

EBX[31:0]	CAT_L3_CONTENTION	Bit-granular map of isolation/contention of allocation units.	Logical Processor
ECX[0]	Reserved	If 1, supports L3 CAT for non-CPU agents.	
ECX[1]	CAT_L3_NONCPU	N/A	Logical Processor
ECX[2]	CAT_L3_CDP	If 1, supports L3 Code and Data Prioritization Technology.	Logical Processor
ECX[3]	CAT_L3_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous.	Logical Processor
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L3_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Logical Processor
EDX[31:16]	Reserved	Reserved.	

CPUID.28H.02H -- Asymmetric L2 Cache Allocation Technology

CPUID.28H.ResID=2 returns information about Asymmetric L2 Cache Allocation Technology.

Table 21-85. Leaf 28H.02H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L2_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Logical Processor
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L2_CONTENTION	Bit-granular map of isolation/contention of allocation units.	Logical Processor
ECX[1:0]	Reserved	Reserved.	
ECX[2]	CAT_L2_CDP	If 1, supports L2 Code and Data Prioritization Technology.	Logical Processor
ECX[3]	CAT_L2_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L2_MASK_n registers do not have to be contiguous.	Logical Processor
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L2_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Logical Processor
EDX[31:16]	Reserved	Reserved.	

CPUID.28H.03H -- Asymmetric Memory Bandwidth Allocation

CPUID.28H.ResID=3 returns information about Asymmetric Memory Bandwidth Allocation.

Table 21-86. Leaf 28H.03H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain

EAX[11:0]	MBA_MAX	Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result.	Logical Processor
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[0]	PER_THREAD_MBA	Per-thread MBA controls are supported.	Logical Processor
ECX[1]	Reserved	Reserved.	
ECX[2]	MBA_LINEAR	If 1, the response of the delay values is linear.	Logical Processor
ECX[31:3]	Reserved	Reserved.	
EDX[15:0]	MBA_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Logical Processor
EDX[31:16]	Reserved	Reserved.	

CPUID.28H.05H -- Asymmetric Cache Bandwidth Allocation

Table 21-87. Leaf 28H.05H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[7:0]	CBA_MAX_LEVELS	Reports the maximum core throttling level supported for the corresponding ResID. Add one to the return value to get the number of throttling levels supported.	Logical Processor
EAX[11:8]	BW_SCOPE	If 1, indicates the logical processor scope of the IA32_QoS_Core_BW_Thrtl_n MSRs. Other values are reserved.	Logical Processor
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[2:0]	Reserved	Reserved.	
ECX[3]	CBA_LINEAR	If 1, the response of the bandwidth control is approximately linear. If 0, the response of the bandwidth control is non-linear.	Logical Processor
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CBA_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Logical Processor
EDX[31:16]	Reserved	Reserved.	

CPUID.28H.06H -- Resource Priority Control

Table 21-88. Leaf 28H.06H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[0]	THREAD_ENABLE	If 1, supports per-thread enable of RP through the IA32_RESOURCE_PRIORITY MSR.	Platform

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

EAX[1]	PACKAGE_ENABLE	If 1, supports physical processor package enable of RP through the IA32_RESOURCE_PRIORITY_PKG MSR.	Platform
EAX[31:2]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.80000000H -- Maximum Input Value for Extended Function CPUID Information

CPUID.80000000H returns the highest value the processor recognizes for returning extended processor information. The value is returned in the EAX register and is processor specific.

- This leaf is supported starting with Pentium 4.
- Processors prior to Pentium 4 treat bit 31 as 0, and this leaf returns the values from CPUID.00H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-89. Leaf 80000000H Maximum Input Value for Extended Function CPUID Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_EXTENDED_LEAF	Maximum input value for Extended Function CPUID Information.	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.80000001H -- Extended Processor Signature and Feature Bits

CPUID.80000001H returns information about extended processor signature and features bits.

- This leaf is valid if MAX_EXTENDED_LEAF $\geq$ 80000001H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-90. Leaf 80000001H Extended Processor Signature and Feature Bits

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[0]	LAHF_SAHF_64	If 1, supports the LAHF/SAHF instructions in 64-bit mode. LAHF and SAHF are always available in other modes, regardless of the enumeration of this feature flag.	Platform
ECX[4:1]	Reserved	Reserved.	
ECX[5]	LZCNT	If 1, supports the LZCNT instruction.	Platform
ECX[7:6]	Reserved	Reserved.	
ECX[8]	PREFETCHW	If 1, supports the PREFETCHW instruction.	Platform
ECX[31:9]	Reserved	Reserved.	
EDX[10:0]	Reserved	Reserved.	
EDX[11]	SYSCALL_SYSRET_64	If 1, supports SYSCALL/SYSRET. Intel processors support SYSCALL and SYSRET only in 64-bit mode. This feature flag is always enumerated as 0 outside 64-bit mode.	Platform
EDX[19:12]	Reserved	Reserved.	
EDX[20]	EXECUTE_DIS	If 1, supports Execute Disable Bit.	Platform
EDX[25:21]	Reserved	Reserved.	
EDX[26]	PAGE_1GB	If 1, supports 1-GByte pages.	Platform
EDX[27]	RDTSCP	If 1, supports RDTSCP and IA32_TSC_AUX.	Platform
EDX[28]	Reserved	Reserved.	
EDX[29]	INTEL64	If 1, supports Intel® 64 Architecture.	Platform
EDX[31:30]	Reserved	Reserved.	

CPUID.80000002H -- Processor Brand String (Bytes 0 to 15)

CPUID.80000002H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF $\geq$ 80000002H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-91. Leaf 80000002H Processor Brand String (Bytes 0 to 15)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_0	Processor brand string.	Platform
EBX[31:0]	BRAND_NAME_1	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_2	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_3	Processor brand string continued.	Platform

CPUID.80000003H -- Processor brand string (Bytes 16 to 31)

CPUID.80000003H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000003H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-92. Leaf 80000003H Processor brand string (Bytes 16 to 31)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_4	Processor brand string continued.	Platform
EBX[31:0]	BRAND_NAME_5	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_6	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_7	Processor brand string continued.	Platform

CPUID.80000004H -- Processor brand string (Bytes 32 to 47)

CPUID.80000004H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF $\geq$ 80000004H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-93. Leaf 80000004H Processor brand string (Bytes 32 to 47)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_8	Processor brand string continued.	Platform
EBX[31:0]	BRAND_NAME_9	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_10	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_11	Processor brand string continued.	Platform

CPUID.80000005H -- Reserved

This leaf is reserved and returns all zeroes.

Table 21-94. Leaf 80000005H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.80000006H -- Extended Function CPUID Information

CPUID.80000006H returns Extended Function CPUID information. The preferred method to enumerate caching information description>is to use CPUID.04H—Deterministic Cache Parameters.

- This leaf is valid if MAX_EXTENDED_LEAF $\geq$ 80000006H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX

Table 21-95. Leaf 80000006H Extended Function CPUID Information

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[7:0]	L2_LINE_SIZE	Cache line size in bytes.	Logical Processor
ECX[11:8]	Reserved	Reserved.	
ECX[15:12]	L2_ASSOC	L2 associativity field. The L2 associativity field encodings are listed in the table below.	Logical Processor
ECX[31:16]	L2_SIZE	Cache size in 1K units.	Logical Processor
EDX[7:0]	Reserved	Reserved.	
EDX[31:8]	Reserved	Reserved.	

Table 21-96. L2 Associativity Field Encodings

Encoding Value	Description	Encoding Value	Description
00H	Disabled	08H	16 Ways
01H	1 Way (direct mapped)	09H	Reserved
02H	2 Ways	0AH	32 Ways
03H	Reserved	0BH	48 Ways
04H	4 Ways	0CH	64 Ways
05H	Reserved	0DH	96 Ways
06H	8 Ways	0EH	128 Ways
07H	See CPUID leaf 4 sub-leaf 2 ¹	0FH	Fully Associative

CPUID.80000007H -- Extended Function CPUID Information 1

CPUID.80000007H returns Extended Function CPUID information.

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000007H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-97. Leaf 80000007H Extended Function CPUID Information 1

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[7:0]	Reserved	Reserved.	
EDX[8]	TSC_INVARIANT	If 1, supports Invariant TSC.	Platform
EDX[31:9]	Reserved	Reserved.	

CPUID.80000008H -- Extended Function CPUID Information 2

CPUID.80000008H returns Extended Function CPUID information.

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000008H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-98. Leaf 80000008H Extended Function CPUID Information 2

Register	Field Name	Description	Domain
EAX[7:0]	PHYS_ADDR_SIZE	Number of physical-address bits. If TME-MK is enabled, the number of bits that can be used to address memory may be reduced by IA32_TME_ACTIVATE[35:32].	Platform
EAX[15:8]	LIN_ADDR_SIZE	Number of linear-address bits.	Platform
EAX[23:16]	GUEST_PHYS_ADDR_SIZE	Number of guest-physical-address bits (for software operating in a virtual machine). If this field is zero, PHYS_ADDR_SIZE should be used. Intel processors return zero for this field. Software emulating CPUID may return a different value.	Platform
EAX[31:24]	Reserved	Reserved.	
EBX[8:0]	Reserved	Reserved.	
EBX[9]	WBNOINVD	If 1, supports the WBNOINVD instruction.	Platform
EBX[31:10]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

3. Updates to Chapter 4, Volume 2B

Change bars and **violet** 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:

- In the SYSEXIT—Fast Return from Fast System Call instruction, the third register listed in the Description was corrected to RCX. Other sections in the instruction refer to RCX properly.
- Instruction TCMMIMFP16PS/TCMMRLFP16PS—Matrix Multiplication of Complex Tiles Accumulated into Packed Single Precision Tile was added for AMX_COMPLEX.
- The following instructions were updated for 64-bit destinations of 32-bit rotates and shifts.
 - RCL/RCR/ROL/ROR, Rotate
 - RORX, Rotate Right Logical Without Affecting Flags
 - SAL/SAR/SHL/SHR, Shift
 - SARX/SHLX/SHRX, Shift Without Affecting Flags
 - SHLD, Double Precision Shift Left
 - SHRD, Double Precision Shift Right

4.1 IMM8 CONTROL BYTE OPERATION FOR PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM

The notations introduced in this section are referenced in the reference pages of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM. The operation of the immediate control byte is common to these four string text processing instructions of SSE4.2. This section describes the common operations.

4.1.1 General Description

The operation of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM is defined by the combination of the respective opcode and the interpretation of an immediate control byte that is part of the instruction encoding.

The opcode controls the relationship of input bytes/words to each other (determines whether the inputs terminated strings or whether lengths are expressed explicitly) as well as the desired output (index or mask).

The imm8 control byte for PCMPESTRM/PCMPESTRI/PCMPISTRM/PCMPISTRI encodes a significant amount of programmable control over the functionality of those instructions. Some functionality is unique to each instruction while some is common across some or all of the four instructions. This section describes functionality which is common across the four instructions.

The arithmetic flags (ZF, CF, SF, OF, AF, PF) are set as a result of these instructions. However, the meanings of the flags have been overloaded from their typical meanings in order to provide additional information regarding the relationships of the two inputs.

PCMPxSTRx instructions perform arithmetic comparisons between all possible pairs of bytes or words, one from each packed input source operand. The boolean results of those comparisons are then aggregated in order to produce meaningful results. The imm8 control byte is used to affect the interpretation of individual input elements as well as control the arithmetic comparisons used and the specific aggregation scheme.

Specifically, the imm8 Control Byte consists of bit fields that control the following attributes:

- **Source data format** — Byte/word data element granularity, signed or unsigned elements.
- **Aggregation operation** — Encodes the mode of per-element comparison operation and the aggregation of per-element comparisons into an intermediate result.
- **Polarity** — Specifies intermediate processing to be performed on the intermediate result.
- **Output selection** — Specifies final operation to produce the output (depending on index or mask) from the intermediate result.

4.1.2 Source Data Format

Table 4-1. Source Data Format

Imm8[1:0]	Meaning	Description
00b	Unsigned bytes	Both 128-bit sources are treated as packed, unsigned bytes.
01b	Unsigned words	Both 128-bit sources are treated as packed, unsigned words.
10b	Signed bytes	Both 128-bit sources are treated as packed, signed bytes.
11b	Signed words	Both 128-bit sources are treated as packed, signed words.

If the imm8 control byte has bit[0] cleared, each source contains 16 packed bytes. If the bit is set each source contains 8 packed words. If the imm8 control byte has bit[1] cleared, each input contains unsigned data. If the bit is set each source contains signed data.

4.1.3 Aggregation Operation

Table 4-2. Aggregation Operation

Imm8[3:2]	Mode	Comparison
00b	Equal any	The arithmetic comparison is “equal.”
01b	Ranges	Arithmetic comparison is “greater than or equal” between even indexed bytes/words of reg and each byte/word of reg/mem. Arithmetic comparison is “less than or equal” between odd indexed bytes/words of reg and each byte/word of reg/mem. (reg/mem[m] >= reg[n] for n = even, reg/mem[m] <= reg[n] for n = odd)
10b	Equal each	The arithmetic comparison is “equal.”
11b	Equal ordered	The arithmetic comparison is “equal.”

All 256 (64) possible comparisons are always performed. The individual Boolean results of those comparisons are referred by “BoolRes[Reg/Mem element index, Reg element index].” Comparisons evaluating to “True” are represented with a 1, False with a 0 (positive logic). The initial results are then aggregated into a 16-bit (8-bit) intermediate result (IntRes1) using one of the modes described in the table below, as determined by imm8 control byte bits[3:2].

See Section 4.1.6 for a description of the overrideIfDataInvalid() function used in Table 4-3.

Table 4-3. Aggregation Operation

Mode	Pseudocode
Equal any (find characters from a set)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i++ IntRes1[j] OR= overrideIfDataInvalid(BoolRes[j,i])
Ranges (find characters from ranges)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i+=2 IntRes1[j] OR= (overrideIfDataInvalid(BoolRes[j,i]) AND overrideIfDataInvalid(BoolRes[j,i+1]))
Equal each (string compare)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For i = 0 to UpperBound, i++ IntRes1[i] = overrideIfDataInvalid(BoolRes[i,i])
Equal ordered (substring search)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = imm8[0] ? FFH : FFFFH For j = 0 to UpperBound, j++ For i = 0 to UpperBound-j, k=j to UpperBound, k++, i++ IntRes1[j] AND= overrideIfDataInvalid(BoolRes[k,i])

4.1.4 Polarity

IntRes1 may then be further modified by performing a 1’s complement, according to the value of the imm8 control byte bit[4]. Optionally, a mask may be used such that only those IntRes1 bits which correspond to “valid” reg/mem input elements are complemented (note that the definition of a valid input element is dependent on the specific opcode and is defined in each opcode’s description). The result of the possible negation is referred to as IntRes2.

Table 4-4. Polarity

Imm8[5:4]	Operation	Description
00b	Positive Polarity (+)	IntRes2 = IntRes1
01b	Negative Polarity (-)	IntRes2 = -1 XOR IntRes1
10b	Masked (+)	IntRes2 = IntRes1
11b	Masked (-)	IntRes2[i] = IntRes1[i] if reg/mem[i] invalid, else = ~IntRes1[i]

4.1.5 Output Selection

Table 4-5. Output Selection

Imm8[6]	Operation	Description
0b	Least significant index	The index returned to ECX is of the least significant set bit in IntRes2.
1b	Most significant index	The index returned to ECX is of the most significant set bit in IntRes2.

For PCMPESTRI/PCMPISTRI, the imm8 control byte bit[6] is used to determine if the index is of the least significant or most significant bit of IntRes2.

Table 4-6. Output Selection

Imm8[6]	Operation	Description
0b	Bit mask	IntRes2 is returned as the mask to the least significant bits of XMM0 with zero extension to 128 bits.
1b	Byte/word mask	IntRes2 is expanded into a byte/word mask (based on imm8[1]) and placed in XMM0. The expansion is performed by replicating each bit into all of the bits of the byte/word of the same index.

Specifically for PCMPESTRM/PCMPISTRM, the imm8 control byte bit[6] is used to determine if the mask is a 16 (8) bit mask or a 128 bit byte/word mask.

4.1.6 Valid/Invalid Override of Comparisons

PCMPxSTRx instructions allow for the possibility that an end-of-string (EOS) situation may occur within the 128-bit packed data value (see the instruction descriptions below for details). Any data elements on either source that are determined to be past the EOS are considered to be invalid, and the treatment of invalid data within a comparison pair varies depending on the aggregation function being performed.

In general, the individual comparison result for each element pair BoolRes[i..j] can be forced true or false if one or more elements in the pair are invalid. See Table 4-7.

Table 4-7. Comparison Result for Each Element Pair BoolRes[i..j]

xmm1 byte/ word	xmm2/ m128 byte/word	Imm8[3:2] = 00b (equal any)	Imm8[3:2] = 01b (ranges)	Imm8[3:2] = 10b (equal each)	Imm8[3:2] = 11b (equal ordered)
Invalid	Invalid	Force false	Force false	Force true	Force true
Invalid	Valid	Force false	Force false	Force false	Force true
Valid	Invalid	Force false	Force false	Force false	Force false
Valid	Valid	Do not force	Do not force	Do not force	Do not force

4.1.7 Summary of Im8 Control byte

Table 4-8. Summary of Imm8 Control Byte

Imm8	Description
-----0b	128-bit sources treated as 16 packed bytes.
-----1b	128-bit sources treated as 8 packed words.
-----0-b	Packed bytes/words are unsigned.
-----1-b	Packed bytes/words are signed.
---00--b	Mode is equal any.
---01--b	Mode is ranges.
---10--b	Mode is equal each.
---11--b	Mode is equal ordered.
---0---b	IntRes1 is unmodified.
---1---b	IntRes1 is negated (1's complement).
--0----b	Negation of IntRes1 is for all 16 (8) bits.
--1----b	Negation of IntRes1 is masked by reg/mem validity.
-0-----b	Index of the least significant, set, bit is used (regardless of corresponding input element validity). IntRes2 is returned in least significant bits of XMM0.
-1-----b	Index of the most significant, set, bit is used (regardless of corresponding input element validity). Each bit of IntRes2 is expanded to byte/word.
0-----b	This bit currently has no defined effect, should be 0.
1-----b	This bit currently has no defined effect, should be 0.

4.1.8 Diagram Comparison and Aggregation Process

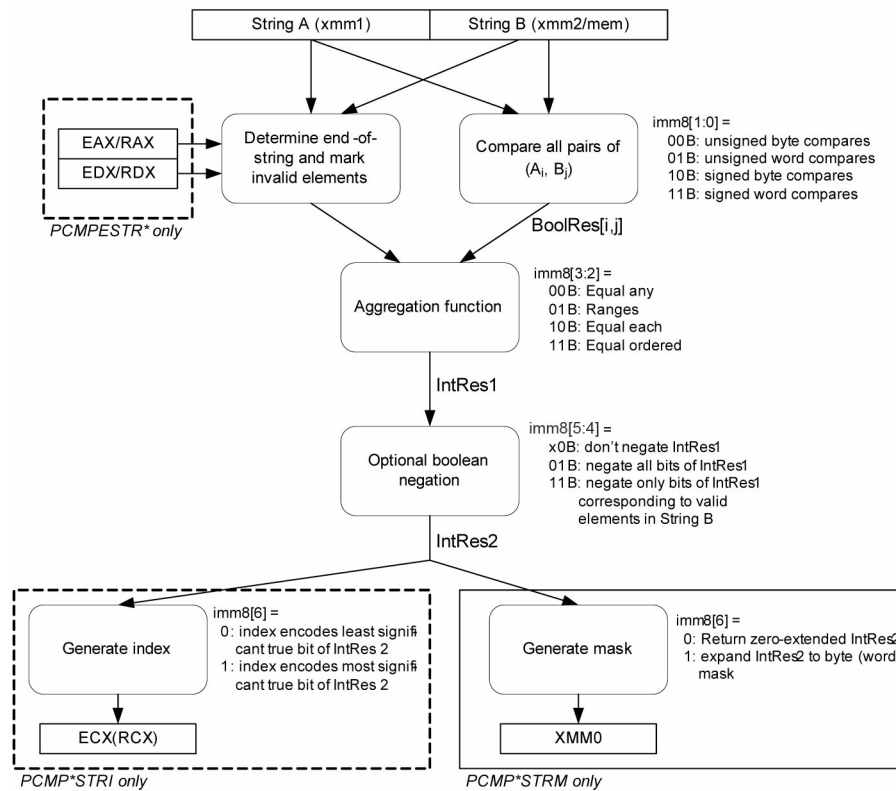


Figure 4-1. Operation of PCMPSTRx and PCMPSTRx

4.2 COMMON TRANSFORMATION AND PRIMITIVE FUNCTIONS FOR SHA1XXX AND SHA256XXX

The following primitive functions and transformations are used in the algorithmic descriptions of SHA1 and SHA256 instruction extensions SHA1NEXTE, SHA1RND4, SHA1MSG1, SHA1MSG2, SHA256RND4, SHA256MSG1, and SHA256MSG2. The operands of these primitives and transformation are generally 32-bit DWORD integers.

- $f_0()$: A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 1 to 20 processing.

$$f_0(B,C,D) := (B \text{ AND } C) \text{ XOR } ((\text{NOT}(B) \text{ AND } D))$$
- $f_1()$: A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 21 to 40 processing.

$$f_1(B,C,D) := B \text{ XOR } C \text{ XOR } D$$
- $f_2()$: A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 41 to 60 processing.

$$f_2(B,C,D) := (B \text{ AND } C) \text{ XOR } (B \text{ AND } D) \text{ XOR } (C \text{ AND } D)$$
- $f_3()$: A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 61 to 80 processing. It is the same as $f_1()$.

$$f_3(B,C,D) := B \text{ XOR } C \text{ XOR } D$$
- $\text{Ch}()$: A bit oriented logical operation that derives a new dword from three SHA256 state variables (dword).

$\text{Ch}(E,F,G) := (E \text{ AND } F) \text{ XOR } ((\text{NOT } E) \text{ AND } G)$

- $\text{Maj}()$: A bit oriented logical operation that derives a new dword from three SHA256 state variables (dword).
 $\text{Maj}(A,B,C) := (A \text{ AND } B) \text{ XOR } (A \text{ AND } C) \text{ XOR } (B \text{ AND } C)$

ROR is rotate right operation

$(A \text{ ROR } N) := A[N-1:0] \parallel A[\text{Width}-1:N]$

ROL is rotate left operation

$(A \text{ ROL } N) := A \text{ ROR } (\text{Width}-N)$

SHR is the right shift operation

$(A \text{ SHR } N) := \text{ZEROES}[N-1:0] \parallel A[\text{Width}-1:N]$

- $\Sigma_0()$: A bit oriented logical and rotational transformation performed on a dword SHA256 state variable.
 $\Sigma_0(A) := (A \text{ ROR } 2) \text{ XOR } (A \text{ ROR } 13) \text{ XOR } (A \text{ ROR } 22)$
- $\Sigma_1()$: A bit oriented logical and rotational transformation performed on a dword SHA256 state variable.
 $\Sigma_1(E) := (E \text{ ROR } 6) \text{ XOR } (E \text{ ROR } 11) \text{ XOR } (E \text{ ROR } 25)$
- $\sigma_0()$: A bit oriented logical and rotational transformation performed on a SHA256 message dword used in the message scheduling.
 $\sigma_0(W) := (W \text{ ROR } 7) \text{ XOR } (W \text{ ROR } 18) \text{ XOR } (W \text{ SHR } 3)$
- $\sigma_1()$: A bit oriented logical and rotational transformation performed on a SHA256 message dword used in the message scheduling.
 $\sigma_1(W) := (W \text{ ROR } 17) \text{ XOR } (W \text{ ROR } 19) \text{ XOR } (W \text{ SHR } 10)$
- K_i : SHA1 Constants dependent on immediate i.
 $K0 = 0x5A827999$
 $K1 = 0x6ED9EBA1$
 $K2 = 0x8F1BBCDC$
 $K3 = 0xCA62C1D6$

4.3 INSTRUCTIONS (M-U)

Chapter 4 continues an alphabetical discussion of Intel® 64 and IA-32 instructions (M-U). See also: Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A; Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C; and Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

RCL/RCR/ROL/ROR—Rotate

Opcode ¹	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
D0 /2	RCL r/m8 ² , 1	M1	Valid	Valid	Rotate 9 bits (CF, r/m8) left once.
D2 /2	RCL r/m8 ² , CL	MC	Valid	Valid	Rotate 9 bits (CF, r/m8) left CL times.
C0 /2 ib	RCL r/m8 ² , imm8	MI	Valid	Valid	Rotate 9 bits (CF, r/m8) left imm8 times.
D1 /2	RCL r/m16, 1	M1	Valid	Valid	Rotate 17 bits (CF, r/m16) left once.
D3 /2	RCL r/m16, CL	MC	Valid	Valid	Rotate 17 bits (CF, r/m16) left CL times.
C1 /2 ib	RCL r/m16, imm8	MI	Valid	Valid	Rotate 17 bits (CF, r/m16) left imm8 times.
D1 /2	RCL r/m32, 1	M1	Valid	Valid	Rotate 33 bits (CF, r/m32) left once.
REX.W + D1 /2	RCL r/m64, 1	M1	Valid	N.E.	Rotate 65 bits (CF, r/m64) left once. Uses a 6 bit count.
D3 /2	RCL r/m32, CL	MC	Valid	Valid	Rotate 33 bits (CF, r/m32) left CL times.
REX.W + D3 /2	RCL r/m64, CL	MC	Valid	N.E.	Rotate 65 bits (CF, r/m64) left CL times. Uses a 6 bit count.
C1 /2 ib	RCL r/m32, imm8	MI	Valid	Valid	Rotate 33 bits (CF, r/m32) left imm8 times.
REX.W + C1 /2 ib	RCL r/m64, imm8	MI	Valid	N.E.	Rotate 65 bits (CF, r/m64) left imm8 times. Uses a 6 bit count.
D0 /3	RCR r/m8 ² , 1	M1	Valid	Valid	Rotate 9 bits (CF, r/m8) right once.
D2 /3	RCR r/m8 ² , CL	MC	Valid	Valid	Rotate 9 bits (CF, r/m8) right CL times.
C0 /3 ib	RCR r/m8 ² , imm8	MI	Valid	Valid	Rotate 9 bits (CF, r/m8) right imm8 times.
D1 /3	RCR r/m16, 1	M1	Valid	Valid	Rotate 17 bits (CF, r/m16) right once.
D3 /3	RCR r/m16, CL	MC	Valid	Valid	Rotate 17 bits (CF, r/m16) right CL times.
C1 /3 ib	RCR r/m16, imm8	MI	Valid	Valid	Rotate 17 bits (CF, r/m16) right imm8 times.
D1 /3	RCR r/m32, 1	M1	Valid	Valid	Rotate 33 bits (CF, r/m32) right once. Uses a 6 bit count.
REX.W + D1 /3	RCR r/m64, 1	M1	Valid	N.E.	Rotate 65 bits (CF, r/m64) right once. Uses a 6 bit count.
D3 /3	RCR r/m32, CL	MC	Valid	Valid	Rotate 33 bits (CF, r/m32) right CL times.
REX.W + D3 /3	RCR r/m64, CL	MC	Valid	N.E.	Rotate 65 bits (CF, r/m64) right CL times. Uses a 6 bit count.
C1 /3 ib	RCR r/m32, imm8	MI	Valid	Valid	Rotate 33 bits (CF, r/m32) right imm8 times.
REX.W + C1 /3 ib	RCR r/m64, imm8	MI	Valid	N.E.	Rotate 65 bits (CF, r/m64) right imm8 times. Uses a 6 bit count.
D0 /0	ROL r/m8 ² , 1	M1	Valid	Valid	Rotate 8 bits r/m8 left once.
D2 /0	ROL r/m8 ² , CL	MC	Valid	Valid	Rotate 8 bits r/m8 left CL times.
C0 /0 ib	ROL r/m8 ² , imm8	MI	Valid	Valid	Rotate 8 bits r/m8 left imm8 times.
D1 /0	ROL r/m16, 1	M1	Valid	Valid	Rotate 16 bits r/m16 left once.
D3 /0	ROL r/m16, CL	MC	Valid	Valid	Rotate 16 bits r/m16 left CL times.
C1 /0 ib	ROL r/m16, imm8	MI	Valid	Valid	Rotate 16 bits r/m16 left imm8 times.
D1 /0	ROL r/m32, 1	M1	Valid	Valid	Rotate 32 bits r/m32 left once.
REX.W + D1 /0	ROL r/m64, 1	M1	Valid	N.E.	Rotate 64 bits r/m64 left once. Uses a 6 bit count.
D3 /0	ROL r/m32, CL	MC	Valid	Valid	Rotate 32 bits r/m32 left CL times.

Opcode ¹	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
REX.W + D3 /0	ROL r/m64, CL	MC	Valid	N.E.	Rotate 64 bits r/m64 left CL times. Uses a 6 bit count.
C1 /0 ib	ROL r/m32, imm8	MI	Valid	Valid	Rotate 32 bits r/m32 left imm8 times.
REX.W + C1 /0 ib	ROL r/m64, imm8	MI	Valid	N.E.	Rotate 64 bits r/m64 left imm8 times. Uses a 6 bit count.
D0 /1	ROR r/m8 ² , 1	M1	Valid	Valid	Rotate 8 bits r/m8 right once.
D2 /1	ROR r/m8 ² , CL	MC	Valid	Valid	Rotate 8 bits r/m8 right CL times.
C0 /1 ib	ROR r/m8 ² , imm8	MI	Valid	Valid	Rotate 8 bits r/m16 right imm8 times.
D1 /1	ROR r/m16, 1	M1	Valid	Valid	Rotate 16 bits r/m16 right once.
D3 /1	ROR r/m16, CL	MC	Valid	Valid	Rotate 16 bits r/m16 right CL times.
C1 /1 ib	ROR r/m16, imm8	MI	Valid	Valid	Rotate 16 bits r/m16 right imm8 times.
D1 /1	ROR r/m32, 1	M1	Valid	Valid	Rotate 32 bits r/m32 right once.
REX.W + D1 /1	ROR r/m64, 1	M1	Valid	N.E.	Rotate 64 bits r/m64 right once. Uses a 6 bit count.
D3 /1	ROR r/m32, CL	MC	Valid	Valid	Rotate 32 bits r/m32 right CL times.
REX.W + D3 /1	ROR r/m64, CL	MC	Valid	N.E.	Rotate 64 bits r/m64 right CL times. Uses a 6 bit count.
C1 /1 ib	ROR r/m32, imm8	MI	Valid	Valid	Rotate 32 bits r/m32 right imm8 times.
REX.W + C1 /1 ib	ROR r/m64, imm8	MI	Valid	N.E.	Rotate 64 bits r/m64 right imm8 times. Uses a 6 bit count.

NOTES:

1. See the IA-32 Architecture Compatibility section below.

2. With a REX prefix in 64-bit mode, attempts to access AH, BH, CH, or DH will instead access SPL, DIL, BPL, or SIL, respectively.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M1	ModRM:r/m (w)	1	N/A	N/A
MC	ModRM:r/m (w)	CL	N/A	N/A
MI	ModRM:r/m (w)	imm8	N/A	N/A

Description

Shifts (rotates) the bits of the first operand (destination operand) the number of bit positions specified in the second operand (count operand) and stores the result in the destination operand. The destination operand can be a register or a memory location; the count operand is an unsigned integer that can be an immediate or a value in the CL register. The count is masked to 5 bits (or 6 bits if in 64-bit mode and REX.W = 1).

The rotate left (ROL) and rotate through carry left (RCL) instructions shift all the bits toward more-significant bit positions, except for the most-significant bit, which is rotated to the least-significant bit location. The rotate right (ROR) and rotate through carry right (RCR) instructions shift all the bits toward less significant bit positions, except for the least-significant bit, which is rotated to the most-significant bit location.

The RCL and RCR instructions include the CF flag in the rotation. The RCL instruction shifts the CF flag into the least-significant bit and shifts the most-significant bit into the CF flag. The RCR instruction shifts the CF flag into the most-significant bit and shifts the least-significant bit into the CF flag. For the ROL and ROR instructions, the original value of the CF flag is not a part of the result, but the CF flag receives a copy of the bit that was shifted from one end to the other.

The OF flag is defined only for the 1-bit rotates; it is undefined in all other cases (except RCL and RCR instructions only: a zero-bit rotate does nothing, that is affects no flags). For left rotates, the OF flag is set to the exclusive OR of the CF bit (after the rotate) and the most-significant bit of the result. For right rotates, the OF flag is set to the exclusive OR of the two most-significant bits of the result.

In 64-bit mode, using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Use of REX.W promotes the first operand to 64 bits and causes the count operand to become a 6-bit counter.

IA-32 Architecture Compatibility

The 8086 does not mask the rotation count. However, all other IA-32 processors (starting with the Intel 286 processor) do mask the rotation count to 5 bits, resulting in a maximum count of 31. This masking is done in all operating modes (including the virtual-8086 mode) to reduce the maximum execution time of the instructions.

Operation

(* RCL and RCR Instructions *)

SIZE := OperandSize;

CASE (determine count) OF

SIZE := 8: tempCOUNT := (COUNT AND 1FH) MOD 9;
 SIZE := 16: tempCOUNT := (COUNT AND 1FH) MOD 17;
 SIZE := 32: tempCOUNT := COUNT AND 1FH;
 SIZE := 64: tempCOUNT := COUNT AND 3FH;

ESAC;

IF OperandSize = 64

THEN COUNTMASK = 3FH;

ELSE COUNTMASK = 1FH;

FI;

(* RCL Instruction Operation *)

tempDEST := DEST;

WHILE (tempCOUNT ≠ 0)

DO

tempCF := MSB(tempDEST);
 tempDEST := (tempDEST * 2) + CF;
 CF := tempCF;
 tempCOUNT := tempCOUNT - 1;

OD;

ELIHW;

IF (COUNT & COUNTMASK) = 1

THEN OF := MSB(tempDEST) XOR CF;

ELSE OF is undefined;

FI;

DEST := tempDEST;

(* RCR Instruction Operation *)

```
tempDEST := DEST;
IF (COUNT & COUNTMASK) = 1
  THEN OF := MSB(tempDEST) XOR CF;
  ELSE OF is undefined;
FI;
WHILE (tempCOUNT ≠ 0)
  DO
    tempCF := LSB(SRC);
    tempDEST := (tempDEST / 2) + (CF * 2SIZE);
    CF := tempCF;
    tempCOUNT := tempCOUNT - 1;
  OD;
DEST := tempDEST;
```

(* ROL Instruction Operation *)

```
tempCOUNT := (COUNT & COUNTMASK) MOD SIZE
tempDEST := DEST;
WHILE (tempCOUNT ≠ 0)
  DO
    tempCF := MSB(tempDEST);
    tempDEST := (tempDEST * 2) + tempCF;
    tempCOUNT := tempCOUNT - 1;
  OD;
ELIHW;
IF (COUNT & COUNTMASK) ≠ 0
  THEN CF := LSB(tempDEST);
FI;
IF (COUNT & COUNTMASK) = 1
  THEN OF := MSB(tempDEST) XOR CF;
  ELSE OF is undefined;
FI;
DEST := tempDEST;
```

(* ROR Instruction Operation *)

```
tempCOUNT := (COUNT & COUNTMASK) MOD SIZE
tempDEST := DEST;
WHILE (tempCOUNT ≠ 0)
  DO
    tempCF := LSB(SRC);
    tempDEST := (tempDEST / 2) + (tempCF * 2SIZE);
    tempCOUNT := tempCOUNT - 1;
  OD;
ELIHW;
IF (COUNT & COUNTMASK) ≠ 0
  THEN CF := MSB(tempDEST);
FI;
IF (COUNT & COUNTMASK) = 1
  THEN OF := MSB(tempDEST) XOR MSB - 1(tempDEST);
  ELSE OF is undefined;
FI;
DEST := tempDEST;
```

Flags Affected

For RCL and RCR instructions, a zero-bit rotate does nothing, i.e., affects no flags. For ROL and ROR instructions, if the masked count is 0, the flags are not affected. If the masked count is 1, then the OF flag is affected, otherwise (masked count is greater than 1) the OF flag is undefined.

For all instructions, the CF flag is affected when the masked count is non-zero. The SF, ZF, AF, and PF flags are always unaffected.

Protected Mode Exceptions

#GP(0)	If the source operand is located in a non-writable segment. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the source operand is located in a nonwritable segment. If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

RORX — Rotate Right Logical Without Affecting Flags

Opcode/ Instruction	Op/ En	64/32- bit Mode	CPUID Feature Flag	Description
VEX.LZ.F2.0F3A.W0 F0 /r ib RORX r32, r/m32, imm8	RMI	V/V	BMI2	Rotate 32-bit r/m32 right imm8 times and load into r32 without affecting arithmetic flags.
VEX.LZ.F2.0F3A.W1 F0 /r ib RORX r64, r/m64, imm8	RMI	V/N.E.	BMI2	Rotate 64-bit r/m64 right imm8 times and load into r64 without affecting arithmetic flags.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RMI	ModRM:reg (w)	ModRM:r/m (r)	imm8	N/A

Description

Rotates the bits of **source** operand right by the count value specified in imm8 without affecting arithmetic flags. The result is written to the destination operand.

This instruction is not supported in real mode and virtual-8086 mode. The operand size is always 32 bits if not in 64-bit mode. In 64-bit mode operand size 64 requires VEX.W1. VEX.W1 is ignored in non-64-bit modes. An attempt to execute this instruction with VEX.L not equal to 0 will cause #UD.

Operation

```
IF (OperandSize = 32)
    y := imm8 AND 1FH;
    DEST := (SRC >> y) | (SRC << (32-y));
ELSEIF (OperandSize = 64)
    y := imm8 AND 3FH;
    DEST := (SRC >> y) | (SRC << (64-y));
FI;
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

Auto-generated from high-level language.

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Table 2-29, "Type 13 Class Exception Conditions."

SAL/SAR/SHL/SHR—Shift

Opcode ¹	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
D0 /4	SAL r/m8 ² , 1	M1	Valid	Valid	Multiply r/m8 by 2, once.
D2 /4	SAL r/m8 ² , CL	MC	Valid	Valid	Multiply r/m8 by 2, CL times.
C0 /4 ib	SAL r/m8 ² , imm8	MI	Valid	Valid	Multiply r/m8 by 2, imm8 times.
D1 /4	SAL r/m16, 1	M1	Valid	Valid	Multiply r/m16 by 2, once.
D3 /4	SAL r/m16, CL	MC	Valid	Valid	Multiply r/m16 by 2, CL times.
C1 /4 ib	SAL r/m16, imm8	MI	Valid	Valid	Multiply r/m16 by 2, imm8 times.
D1 /4	SAL r/m32, 1	M1	Valid	Valid	Multiply r/m32 by 2, once.
REX.W + D1 /4	SAL r/m64, 1	M1	Valid	N.E.	Multiply r/m64 by 2, once.
D3 /4	SAL r/m32, CL	MC	Valid	Valid	Multiply r/m32 by 2, CL times.
REX.W + D3 /4	SAL r/m64, CL	MC	Valid	N.E.	Multiply r/m64 by 2, CL times.
C1 /4 ib	SAL r/m32, imm8	MI	Valid	Valid	Multiply r/m32 by 2, imm8 times.
REX.W + C1 /4 ib	SAL r/m64, imm8	MI	Valid	N.E.	Multiply r/m64 by 2, imm8 times.
D0 /7	SAR r/m8 ² , 1	M1	Valid	Valid	Signed divide ³ r/m8 by 2, once.
D2 /7	SAR r/m8 ² , CL	MC	Valid	Valid	Signed divide ³ r/m8 by 2, CL times.
C0 /7 ib	SAR r/m8 ² , imm8	MI	Valid	Valid	Signed divide ³ r/m8 by 2, imm8 times.
D1 /7	SAR r/m16, 1	M1	Valid	Valid	Signed divide ³ r/m16 by 2, once.
D3 /7	SAR r/m16, CL	MC	Valid	Valid	Signed divide ³ r/m16 by 2, CL times.
C1 /7 ib	SAR r/m16, imm8	MI	Valid	Valid	Signed divide ³ r/m16 by 2, imm8 times.
D1 /7	SAR r/m32, 1	M1	Valid	Valid	Signed divide ³ r/m32 by 2, once.
REX.W + D1 /7	SAR r/m64, 1	M1	Valid	N.E.	Signed divide ³ r/m64 by 2, once.
D3 /7	SAR r/m32, CL	MC	Valid	Valid	Signed divide ³ r/m32 by 2, CL times.
REX.W + D3 /7	SAR r/m64, CL	MC	Valid	N.E.	Signed divide ³ r/m64 by 2, CL times.
C1 /7 ib	SAR r/m32, imm8	MI	Valid	Valid	Signed divide ³ r/m32 by 2, imm8 times.
REX.W + C1 /7 ib	SAR r/m64, imm8	MI	Valid	N.E.	Signed divide ³ r/m64 by 2, imm8 times.
D0 /4	SHL r/m8 ² , 1	M1	Valid	Valid	Multiply r/m8 by 2, once.
D2 /4	SHL r/m8 ² , CL	MC	Valid	Valid	Multiply r/m8 by 2, CL times.
C0 /4 ib	SHL r/m8 ² , imm8	MI	Valid	Valid	Multiply r/m8 by 2, imm8 times.
D1 /4	SHL r/m16, 1	M1	Valid	Valid	Multiply r/m16 by 2, once.
D3 /4	SHL r/m16, CL	MC	Valid	Valid	Multiply r/m16 by 2, CL times.
C1 /4 ib	SHL r/m16, imm8	MI	Valid	Valid	Multiply r/m16 by 2, imm8 times.
D1 /4	SHL r/m32, 1	M1	Valid	Valid	Multiply r/m32 by 2, once.
REX.W + D1 /4	SHL r/m64, 1	M1	Valid	N.E.	Multiply r/m64 by 2, once.
D3 /4	SHL r/m32, CL	MC	Valid	Valid	Multiply r/m32 by 2, CL times.
REX.W + D3 /4	SHL r/m64, CL	MC	Valid	N.E.	Multiply r/m64 by 2, CL times.
C1 /4 ib	SHL r/m32, imm8	MI	Valid	Valid	Multiply r/m32 by 2, imm8 times.
REX.W + C1 /4 ib	SHL r/m64, imm8	MI	Valid	N.E.	Multiply r/m64 by 2, imm8 times.
D0 /5	SHR r/m8 ² , 1	M1	Valid	Valid	Unsigned divide r/m8 by 2, once.
D2 /5	SHR r/m8 ² , CL	MC	Valid	Valid	Unsigned divide r/m8 by 2, CL times.
C0 /5 ib	SHR r/m8 ² , imm8	MI	Valid	Valid	Unsigned divide r/m8 by 2, imm8 times.
D1 /5	SHR r/m16, 1	M1	Valid	Valid	Unsigned divide r/m16 by 2, once.

Opcode ¹	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
D3 /5	SHR r/m16, CL	MC	Valid	Valid	Unsigned divide r/m16 by 2, CL times
C1 /5 ib	SHR r/m16, imm8	MI	Valid	Valid	Unsigned divide r/m16 by 2, imm8 times.
D1 /5	SHR r/m32, 1	M1	Valid	Valid	Unsigned divide r/m32 by 2, once.
REX.W + D1 /5	SHR r/m64, 1	M1	Valid	N.E.	Unsigned divide r/m64 by 2, once.
D3 /5	SHR r/m32, CL	MC	Valid	Valid	Unsigned divide r/m32 by 2, CL times.
REX.W + D3 /5	SHR r/m64, CL	MC	Valid	N.E.	Unsigned divide r/m64 by 2, CL times.
C1 /5 ib	SHR r/m32, imm8	MI	Valid	Valid	Unsigned divide r/m32 by 2, imm8 times.
REX.W + C1 /5 ib	SHR r/m64, imm8	MI	Valid	N.E.	Unsigned divide r/m64 by 2, imm8 times.

NOTES:

1. See the IA-32 Architecture Compatibility section below.
2. With a REX prefix in 64-bit mode, attempts to access AH, BH, CH, or DH will instead access SPL, DIL, BPL, or SIL, respectively.
3. Not the same form of division as IDIV; rounding is toward negative infinity.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M1	ModRM:r/m (r, w)	1	N/A	N/A
MC	ModRM:r/m (r, w)	CL	N/A	N/A
MI	ModRM:r/m (r, w)	imm8	N/A	N/A

Description

Shifts the bits in the first operand (destination operand) to the left or right by the number of bits specified in the second operand (count operand). Bits shifted beyond the destination operand boundary are first shifted into the CF flag, then discarded. At the end of the shift operation, the CF flag contains the last bit shifted out of the destination operand.

The destination operand can be a register or a memory location. The count operand can be an immediate value or the CL register. The count is masked to 5 bits (or 6 bits with a 64-bit operand). The count range is limited to 0 to 31 (or 63 with a 64-bit operand). A special opcode encoding is provided for a count of 1.

The shift arithmetic left (SAL) and shift logical left (SHL) instructions perform the same operation; they shift the bits in the destination operand to the left (toward more significant bit locations). For each shift count, the most significant bit of the destination operand is shifted into the CF flag, and the least significant bit is cleared (see Figure 7-7 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1).

The shift arithmetic right (SAR) and shift logical right (SHR) instructions shift the bits of the destination operand to the right (toward less significant bit locations). For each shift count, the least significant bit of the destination operand is shifted into the CF flag, and the most significant bit is either set or cleared depending on the instruction type. The SHR instruction clears the most significant bit (see Figure 7-8 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1); the SAR instruction sets or clears the most significant bit to correspond to the sign (most significant bit) of the original value in the destination operand. In effect, the SAR instruction fills the empty bit position's shifted value with the sign of the unshifted value (see Figure 7-9 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1).

The SAR and SHR instructions can be used to perform signed or unsigned division, respectively, of the destination operand by powers of 2. For example, using the SAR instruction to shift a signed integer 1 bit to the right divides the value by 2.

Using the SAR instruction to perform a division operation does not produce the same result as the IDIV instruction. The quotient from the IDIV instruction is rounded toward zero, whereas the "quotient" of the SAR instruction is rounded toward negative infinity. This difference is apparent only for negative numbers. For example, when the IDIV instruction is used to divide -9 by 4, the result is -2 with a remainder of -1. If the SAR instruction is used to

shift -9 right by two bits, the result is -3 and the “remainder” is +3; however, the SAR instruction stores only the most significant bit of the remainder (in the CF flag).

The OF flag is affected only on 1-bit shifts. For left shifts, the OF flag is set to 0 if the most-significant bit of the result is the same as the CF flag (that is, the top two bits of the original operand were the same); otherwise, it is set to 1. For the SAR instruction, the OF flag is cleared for all 1-bit shifts. For the SHR instruction, the OF flag is set to the most-significant bit of the original operand.

In 64-bit mode, the instruction’s default operation size is 32 bits and the mask width for CL is 5 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64-bits and sets the mask width for CL to 6 bits. See the summary chart at the beginning of this section for encoding data and limits.

IA-32 Architecture Compatibility

The 8086 does not mask the shift count. However, all other IA-32 processors (starting with the Intel 286 processor) do mask the shift count to 5 bits, resulting in a maximum count of 31. This masking is done in all operating modes (including the virtual-8086 mode) to reduce the maximum execution time of the instructions.

Operation

```
IF OperandSize = 64
    THEN
        countMASK := 3FH;
    ELSE
        countMASK := 1FH;
FI

tempCOUNT := (COUNT AND countMASK);

origDEST := DEST;
tempDEST := DEST;
WHILE (tempCOUNT ≠ 0)
DO
    IF instruction is SAL or SHL
        THEN
            CF := MSB(tempDEST);
        ELSE (* Instruction is SAR or SHR *)
            CF := LSB(tempDEST);
        FI;
    IF instruction is SAL or SHL
        THEN
            tempDEST := tempDEST * 2;
        ELSE
            IF instruction is SAR
                THEN
                    tempDEST := tempDEST / 2; (* Signed divide, rounding toward negative infinity *)
                ELSE (* Instruction is SHR *)
                    tempDEST := tempDEST / 2; (* Unsigned divide *)
                FI;
            FI;
        tempCOUNT := tempCOUNT - 1;
OD;

(* Determine overflow for the various instructions *)
IF (COUNT and countMASK) = 1
    THEN
        IF instruction is SAL or SHL
```



```

        THEN
            OF := MSB(tempDEST) XOR CF;
        ELSE
            IF instruction is SAR
                THEN
                    OF := 0;
                ELSE (* Instruction is SHR *)
                    OF := MSB(origDEST);
            FI;
        FI;
    ELSE IF (COUNT AND countMASK) = 0
        THEN
            All flags unchanged;
        ELSE (* COUNT not 1 or 0 *)
            OF := undefined;
        FI;
    FI;
    DEST := tempDEST;

```

Flags Affected

The CF flag contains the value of the last bit shifted out of the destination operand; it is undefined for SHL and SHR instructions where the count is greater than or equal to the size (in bits) of the destination operand. The OF flag is affected only for 1-bit shifts (see “Description” above); otherwise, it is undefined. The SF, ZF, and PF flags are set according to the result. If the count is 0, the flags are not affected. For a non-zero count, the AF flag is undefined.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

SARX/SHLX/SHRX—Shift Without Affecting Flags

Opcode/ Instruction	Op/ En	64/32- bit Mode	CPUID Feature Flag	Description
VEX.LZ.F3.0F38.W0 F7 /r SARX r32a, r/m32, r32b	RMV	V/V	BMI2	Shift r/m32 arithmetically right with count specified in r32b and load into r32a.
VEX.LZ.66.0F38.W0 F7 /r SHLX r32a, r/m32, r32b	RMV	V/V	BMI2	Shift r/m32 logically left with count specified in r32b and load into r32a.
VEX.LZ.F2.0F38.W0 F7 /r SHRX r32a, r/m32, r32b	RMV	V/V	BMI2	Shift r/m32 logically right with count specified in r32b and load into r32a.
VEX.LZ.F3.0F38.W1 F7 /r SARX r64a, r/m64, r64b	RMV	V/N.E.	BMI2	Shift r/m64 arithmetically right with count specified in r64b and load into r64a.
VEX.LZ.66.0F38.W1 F7 /r SHLX r64a, r/m64, r64b	RMV	V/N.E.	BMI2	Shift r/m64 logically left with count specified in r64b and load into r64a.
VEX.LZ.F2.0F38.W1 F7 /r SHRX r64a, r/m64, r64b	RMV	V/N.E.	BMI2	Shift r/m64 logically right with count specified in r64b and load into r64a.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RMV	ModRM:reg (w)	ModRM:r/m (r)	VEX.vvvv (r)	N/A

Description

Shifts the bits of the first source operand (the second operand) to the left or right by a COUNT value specified in the second source operand (the third operand). The result is written to the destination operand (the first operand).

The shift arithmetic right (SARX) and shift logical right (SHRX) instructions shift the bits of the destination operand to the right (toward less significant bit locations), SARX keeps and propagates the most significant bit (sign bit) while shifting.

The logical shift left (SHLX) shifts the bits of the destination operand to the left (toward more significant bit locations).

This instruction is not supported in real mode and virtual-8086 mode. The operand size is always 32 bits if not in 64-bit mode. In 64-bit mode operand size 64 requires VEX.W1. VEX.W1 is ignored in non-64-bit modes. An attempt to execute this instruction with VEX.L not equal to 0 will cause #UD.

If the value specified in the first source operand exceeds OperandSize -1, the COUNT value is masked.

SARX,SHRX, and SHLX instructions do not update flags.

Operation

```

TEMP := SRC1;
IF VEX.W1 and CS.L = 1
THEN
    countMASK := 3FH;
ELSE
    countMASK := 1FH;
FI
COUNT := SRC2 AND countMASK;

DO WHILE (COUNT ≠ 0)
    IF instruction is SHLX
        THEN
            TEMP := TEMP *2;
    
```

```

        ELSE IF instruction is SHRX
            THEN
                TEMP := TEMP / 2; //unsigned divide
            ELSE
                // SARX
                TEMP := TEMP / 2; // signed divide, round toward negative infinity
        FI;
        COUNT := COUNT - 1;
    OD
    DEST := TEMP;

```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

Auto-generated from high-level language.

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Table 2-29, “Type 13 Class Exception Conditions.”

SHLD—Double Precision Shift Left

Opcode*	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF A4 /r ib	SHLD r/m16, r16, imm8	MRI	Valid	Valid	Shift r/m16 to left imm8 places while shifting bits from r16 in from the right.
OF A5 /r	SHLD r/m16, r16, CL	MRC	Valid	Valid	Shift r/m16 to left CL places while shifting bits from r16 in from the right.
OF A4 /r ib	SHLD r/m32, r32, imm8	MRI	Valid	Valid	Shift r/m32 to left imm8 places while shifting bits from r32 in from the right.
REX.W + OF A4 /r ib	SHLD r/m64, r64, imm8	MRI	Valid	N.E.	Shift r/m64 to left imm8 places while shifting bits from r64 in from the right.
OF A5 /r	SHLD r/m32, r32, CL	MRC	Valid	Valid	Shift r/m32 to left CL places while shifting bits from r32 in from the right.
REX.W + OF A5 /r	SHLD r/m64, r64, CL	MRC	Valid	N.E.	Shift r/m64 to left CL places while shifting bits from r64 in from the right.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MRI	ModRM:r/m (w)	ModRM:reg (r)	imm8	N/A
MRC	ModRM:r/m (w)	ModRM:reg (r)	CL	N/A

Description

The SHLD instruction is used for multi-precision shifts of 64 bits or more.

The instruction shifts the first operand (destination operand) to the left the number of bits specified by the third operand (count operand). The second operand (source operand) provides bits to shift in from the right (starting with bit 0 of the destination operand).

The destination operand can be a register or a memory location; the source operand is a register. The count operand is an unsigned integer that can be stored in an immediate byte or in the CL register. If the count operand is CL, the shift count is the logical AND of CL and a count mask. In non-64-bit modes and default 64-bit mode; only bits 0 through 4 of the count are used. This masks the count to a value between 0 and 31. If a count is greater than the operand size, the result is undefined.

If the count is 1 or greater, the CF flag is filled with the last bit shifted out of the destination operand. For a 1-bit shift, the OF flag is set if a sign change occurred; otherwise, it is cleared. If the count operand is 0, flags are not affected.

In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits (upgrading the count mask to 6 bits). See the summary chart at the beginning of this section for encoding data and limits.

Operation

```

IF (In 64-Bit Mode and REX.W = 1)
    THEN COUNT := COUNT MOD 64;
    ELSE COUNT := COUNT MOD 32;
FI
SIZE := OperandSize;
tempDEST := DEST;
IF COUNT > SIZE
    THEN (* Bad parameters *)
        tempDEST is undefined;

```

```

    CF, OF, SF, ZF, AF, PF are undefined;
ELSE IF COUNT > 0 (* Perform the shift *)
    CF := BIT[tempDEST, SIZE - COUNT];
    (* Last bit shifted out on exit *)
    FOR i := SIZE - 1 DOWN TO COUNT
        DO
            Bit(tempDEST, i) := Bit(tempDEST, i - COUNT);
        OD;
    FOR i := COUNT - 1 DOWN TO 0
        DO
            BIT[tempDEST, i] := BIT[SRC, i - COUNT + SIZE];
        OD;
FI;
DEST := tempDEST;

```

Flags Affected

If the count is 1 or greater, the CF flag is filled with the last bit shifted out of the destination operand and the SF, ZF, and PF flags are set according to the value of the result. For a 1-bit shift, the OF flag is set if a sign change occurred; otherwise, it is cleared. For shifts greater than 1 bit, the OF flag is undefined. If a shift occurs, the AF flag is undefined. If the count operand is 0, the flags are not affected. If the count is greater than the operand size, the flags are undefined.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.

#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

SHRD—Double Precision Shift Right

Opcode*	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF AC /r ib	SHRD r/m16, r16, imm8	MRI	Valid	Valid	Shift r/m16 to right imm8 places while shifting bits from r16 in from the left.
OF AD /r	SHRD r/m16, r16, CL	MRC	Valid	Valid	Shift r/m16 to right CL places while shifting bits from r16 in from the left.
OF AC /r ib	SHRD r/m32, r32, imm8	MRI	Valid	Valid	Shift r/m32 to right imm8 places while shifting bits from r32 in from the left.
REX.W + OF AC /r ib	SHRD r/m64, r64, imm8	MRI	Valid	N.E.	Shift r/m64 to right imm8 places while shifting bits from r64 in from the left.
OF AD /r	SHRD r/m32, r32, CL	MRC	Valid	Valid	Shift r/m32 to right CL places while shifting bits from r32 in from the left.
REX.W + OF AD /r	SHRD r/m64, r64, CL	MRC	Valid	N.E.	Shift r/m64 to right CL places while shifting bits from r64 in from the left.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MRI	ModRM:r/m (w)	ModRM:reg (r)	imm8	N/A
MRC	ModRM:r/m (w)	ModRM:reg (r)	CL	N/A

Description

The SHRD instruction is useful for multi-precision shifts of 64 bits or more.

The instruction shifts the first operand (destination operand) to the right the number of bits specified by the third operand (count operand). The second operand (source operand) provides bits to shift in from the left (starting with the most significant bit of the destination operand).

The destination operand can be a register or a memory location; the source operand is a register. The count operand is an unsigned integer that can be stored in an immediate byte or the CL register. If the count operand is CL, the shift count is the logical AND of CL and a count mask. In non-64-bit modes and default 64-bit mode, the width of the count mask is 5 bits. Only bits 0 through 4 of the count register are used (masking the count to a value between 0 and 31). If the count is greater than the operand size, the result is undefined.

If the count is 1 or greater, the CF flag is filled with the last bit shifted out of the destination operand. For a 1-bit shift, the OF flag is set if a sign change occurred; otherwise, it is cleared. If the count operand is 0, flags are not affected.

In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits (upgrading the count mask to 6 bits). See the summary chart at the beginning of this section for encoding data and limits.

Operation

```
IF (In 64-Bit Mode and REX.W = 1)
    THEN COUNT := COUNT MOD 64;
    ELSE COUNT := COUNT MOD 32;
FI
SIZE := OperandSize;
tempDEST := DEST;
IF COUNT > SIZE
    THEN (* Bad parameters *)
        tempDEST is undefined;
```



```

    CF, OF, SF, ZF, AF, PF are undefined;
ELSE IF COUNT > 0 (* Perform the shift *)
    CF := BIT[tempDEST, COUNT - 1]; (* Last bit shifted out on exit *)
    FOR i := 0 TO SIZE - 1 - COUNT
        DO
            BIT[tempDEST, i] := BIT[tempDEST, i + COUNT];
        OD;
    FOR i := SIZE - COUNT TO SIZE - 1
        DO
            BIT[tempDEST, i] := BIT[SRC, i + COUNT - SIZE];
        OD;
FI;
DEST := tempDEST;

```

Flags Affected

If the count is 1 or greater, the CF flag is filled with the last bit shifted out of the destination operand and the SF, ZF, and PF flags are set according to the value of the result. For a 1-bit shift, the OF flag is set if a sign change occurred; otherwise, it is cleared. For shifts greater than 1 bit, the OF flag is undefined. If a shift occurs, the AF flag is undefined. If the count operand is 0, the flags are not affected. If the count is greater than the operand size, the flags are undefined.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.

#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

SYSEXIT—Fast Return from Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 35	SYSEXIT	Z0	Valid	Valid	Fast return to privilege level 3 user code.
REX.W + OF 35	SYSEXIT	Z0	Valid	Valid	Fast return to 64-bit mode privilege level 3 user code.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

Executes a fast return to privilege level 3 user code. SYSEXIT is a companion instruction to the SYSENTER instruction. The instruction is optimized to provide the maximum performance for returns from system procedures executing at protection levels 0 to user procedures executing at protection level 3. It must be executed from code executing at privilege level 0.

With a 64-bit operand size, SYSEXIT remains in 64-bit mode; otherwise, it either enters compatibility mode (if the logical processor is in IA-32e mode) or remains in protected mode (if it is not).

Prior to executing SYSEXIT, software must specify the privilege level 3 code segment and code entry point, and the privilege level 3 stack segment and stack pointer by writing values into the following MSR and general-purpose registers:

- **IA32_SYSENTER_CS** (MSR address 174H) — Contains a 32-bit value that is used to determine the segment selectors for the privilege level 3 code and stack segments (see the Operation section)
- **RDX** — The canonical address in this register is loaded into RIP (thus, this value references the first instruction to be executed in the user code). If the return is not to 64-bit mode, only bits 31:0 are loaded.
- **RCX** — The canonical address in this register is loaded into RSP (thus, this value contains the stack pointer for the privilege level 3 stack). If the return is not to 64-bit mode, only bits 31:0 are loaded.

The IA32_SYSENTER_CS MSR can be read from and written to using RDMSR and WRMSR.

While SYSEXIT loads the CS and SS selectors with values derived from the IA32_SYSENTER_CS MSR, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSEXIT instruction does not ensure this correspondence.

The SYSEXIT instruction can be invoked from all operating modes except real-address mode and virtual-8086 mode.

The SYSENTER and SYSEXIT instructions were introduced into the IA-32 architecture in the Pentium II processor. The availability of these instructions on a processor is indicated with the SYSENTER/SYSEXIT present (SEP) feature flag returned to the EDX register by the CPUID instruction. An operating system that qualifies the SEP flag must also qualify the processor family and model to ensure that the SYSENTER/SYSEXIT instructions are actually present. For example:

```
IF CPUID SEP bit is set
    THEN IF (Family = 6) and (Model < 3) and (Stepping < 3)
        THEN
            SYSENTER/SYSEXIT_Not_Supported; FI;
        ELSE
            SYSENTER/SYSEXIT_Supported; FI;
    FI;
```

When the CPUID instruction is executed on the Pentium Pro processor (model 1), the processor returns a the SEP flag as set, but does not support the SYSENTER/SYSEXIT instructions.

When shadow stacks are enabled at privilege level 3 the instruction loads SSP with value from IA32_PL3_SSP MSR. Refer to Chapter 7, “Interrupt and Exception Handling,” and Chapter 18, “Control-flow Enforcement Technology (CET),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for additional CET details.

The instruction cannot be executed when FRED transitions are enabled. An operating system that has enabled FRED transitions should use ERETU instead.

Instruction ordering. Instructions following a SYSEXIT may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the SYSEXIT have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Operation

```
IF CR4.FRED = 1
    THEN #UD; FI;

IF IA32_SYSENTER_CS[15:2] = 0 OR CR0.PE = 0 OR CPL ≠ 0 THEN #GP(0); FI;

IF operand size is 64-bit
    THEN (* Return to 64-bit mode *)
        RSP := RCX;
        RIP := RDX;
    ELSE (* Return to protected mode or compatibility mode *)
        RSP := ECX;
        RIP := EDX;

FI;

IF operand size is 64-bit (* Operating system provides CS; RPL forced to 3 *)
    THEN CS.Selector := IA32_SYSENTER_CS[15:0] + 32;
    ELSE CS.Selector := IA32_SYSENTER_CS[15:0] + 16;

FI;
CS.Selector := CS.Selector OR 3; (* RPL forced to 3 *)
(* Set rest of CS to a fixed value *)
CS.Base := 0; (* Flat segment *)
CS.Limit := FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)
CS.Type := 11; (* Execute/read code, accessed *)
CS.S := 1;
CS.DPL := 3;
CS.P := 1;

IF operand size is 64-bit
    THEN (* return to 64-bit mode *)
        CS.L := 1; (* 64-bit code segment *)
        CS.D := 0; (* Required if CS.L = 1 *)
    ELSE (* return to protected mode or compatibility mode *)
        CS.L := 0;
        CS.D := 1; (* 32-bit code segment *)

FI;
CS.G := 1; (* 4-KByte granularity *)
CPL := 3;

IF ShadowStackEnabled(CPL)
    THEN SSP := IA32_PL3_SSP;

FI;

SS.Selector := CS.Selector + 8; (* SS just above CS *)
(* Set rest of SS to a fixed value *)
SS.Base := 0; (* Flat segment *)
SS.Limit := FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)
```

SS.Type := 3;	(* Read/write data, accessed *)
SS.S := 1;	
SS.DPL := 3;	
SS.P := 1;	
SS.B := 1;	(* 32-bit stack segment*)
SS.G := 1;	(* 4-KByte granularity *)

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If IA32_SYSENTER_CS[15:2] = 0. If CPL ≠ 0.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	The SYSEXIT instruction is not recognized in real-address mode.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	The SYSEXIT instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#GP(0)	If IA32_SYSENTER_CS[15:2] = 0. If CPL ≠ 0.
#UD	If the LOCK prefix is used. If CR4.FRED = 1.

64-Bit Mode Exceptions

#GP(0)	If IA32_SYSENTER_CS = 0. If CPL ≠ 0. If RCX or RDX contains a non-canonical address.
#UD	If the LOCK prefix is used. If CR4.FRED = 1.

TCMMIMFP16PS/TCMMRLFP16PS—Matrix Multiplication of Complex Tiles Accumulated into Packed Single Precision Tile

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 6C 11:rrr:bbb TCMMIMFP16PS tmm1, tmm2, tmm3	A	V/N.E.	AMX_COMPLEX	Matrix multiply complex elements from tmm2 and tmm3, and accumulate the imaginary part into single precision elements in tmm1.
VEX.128.NP.0F38.W0 6C 11:rrr:bbb TCMMRLFP16PS tmm1, tmm2, tmm3	A	V/N.E.	AMX_COMPLEX	Matrix multiply complex elements from tmm2 and tmm3, and accumulate the real part into single precision elements in tmm1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (r, w)	ModRM:r/m (r)	VEX.vvvv (r)	N/A

Description

These instructions perform matrix multiplication of two tiles containing complex elements and accumulate the results into a packed single precision tile. Each dword element in input tiles tmm2 and tmm3 is interpreted as a complex number with FP16 real part and FP16 imaginary part.

TCMMRLFP16PS calculates the real part of the result. For each possible combination of (row of tmm2, column of tmm3), the instruction performs a set of multiplication and accumulations on all corresponding complex numbers (one from tmm2 and one from tmm3). The real part of the tmm2 element is multiplied with the real part of the corresponding tmm3 element, and the negated imaginary part of the tmm2 element is multiplied with the imaginary part of the corresponding tmm3 elements. The two accumulated results are added, and then accumulated into the corresponding row and column of tmm1.

TCMMIMFP16PS calculates the imaginary part of the result. For each possible combination of (row of tmm2, column of tmm3), the instruction performs a set of multiplication and accumulations on all corresponding complex numbers (one from tmm2 and one from tmm3). The imaginary part of the tmm2 element is multiplied with the real part of the corresponding tmm3 element, and the real part of the tmm2 element is multiplied with the imaginary part of the corresponding tmm3 elements. The two accumulated results are added, and then accumulated into the corresponding row and column of tmm1.

“Round to nearest even” rounding mode is used when doing each accumulation of the FMA. Output denormals are always flushed to zero but FP16 input denormals are not treated as zero.

MXCSR is not consulted nor updated.

Any attempt to execute these instructions inside an Intel TSX transaction will result in a transaction abort.

Operation

TCMMIMFP16PS tsrcdest, tsrc1, tsrc2

// C = m x n (tsrcdest), A = m x k (tsrc1), B = k x n (tsrc2)

src1 and src2 elements are pairs of fp16

elements_src1 := tsrc1.colsb / 4

elements_dest := tsrcdest.colsb / 4

elements_temp := tsrcdest.colsb / 2 // Count is in fp16 prior to horizontal

for m in 0 ... tsrcdest.rows-1:

 temp1[0 ... elements_temp-1] := 0

 for k in 0 ... elements_src1-1:

 for n in 0 ... elements_dest-1:

```

s1e = cvt_fp16_to_fp32(tsrc1.row[m].fp16[2*k+0]) // real
s2e = cvt_fp16_to_fp32(tsrc2.row[k].fp16[2*n+0]) // real
s1o = cvt_fp16_to_fp32(tsrc1.row[m].fp16[2*k+1]) // imaginary
s2o = cvt_fp16_to_fp32(tsrc2.row[k].fp16[2*n+1]) // imaginary

// FP32 FMA with DAZ=FTZ=1, RNE rounding.
// MXCSR is neither consulted nor updated.
// No exceptions raised or denoted.

temp1.fp32[2*n+0] = fma32(temp1.fp32[2*n+0], s1e, s2e, daz=1, ftz=1, sae=1, rc=RNE)
temp1.fp32[2*n+1] = fma32(temp1.fp32[2*n+1], s1e, s2o, daz=1, ftz=1, sae=1, rc=RNE)

for n in 0 ... elements_dest-1:
    // DAZ=FTZ=1, RNE rounding.
    // MXCSR is neither consulted nor updated.
    // No exceptions raised or denoted.
    tmpf32 := temp1.fp32[2*n] + temp1.fp32[2*n+1]
    srcdest.row[m].fp32[n] := srcdest.row[m].fp32[n] + tmpf32
write_row_and_zero(tsrcdest, m, tmp, tsrcdest.colsb)

zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tileconfig_start()

TCMMRLFP16PS tsrcdest, tsrc1, tsrc2
// C = m x n (tsrcdest), A = m x k (tsrc1), B = k x n (tsrc2)

# src1 and src2 elements are pairs of fp16
elements_src1 := tsrc1.colsb / 4
elements_dest := tsrcdest.colsb / 4
elements_temp := tsrcdest.colsb / 2 // Count is in fp16 prior to horizontal

for m in 0 ... tsrcdest.rows-1:
    temp1[ 0 ... elements_temp-1 ] := 0
    for k in 0 ... elements_src1-1:
        for n in 0 ... elements_dest-1:

            s1e = cvt_fp16_to_fp32(tsrc1.row[m].fp16[2*k+0]) // real
            s2e = cvt_fp16_to_fp32(tsrc2.row[k].fp16[2*n+0]) // real
            s1o = cvt_fp16_to_fp32(-tsrc1.row[m].fp16[2*k+1]) // imaginary: "-" is for imaginary*imaginary
            s2o = cvt_fp16_to_fp32(tsrc2.row[k].fp16[2*n+1]) // imaginary

            // FP32 FMA with DAZ=FTZ=1, RNE rounding.
            // MXCSR is neither consulted nor updated.
            // No exceptions raised or denoted.

            temp1.fp32[2*n+0] = fma32(temp1.fp32[2*n+0], s1e, s2e, daz=1, ftz=1, sae=1, rc=RNE) // real
            temp1.fp32[2*n+1] = fma32(temp1.fp32[2*n+1], s1o, s2o, daz=1, ftz=1, sae=1, rc=RNE) // imaginary

        for n in 0 ... elements_dest-1:
            // DAZ=FTZ=1, RNE rounding.
            // MXCSR is neither consulted nor updated.
            // No exceptions raised or denoted.
            tmpf32 := temp1.fp32[2*n] + temp1.fp32[2*n+1]

```

```
srcdest.row[m].fp32[n] := srcdest.row[m].fp32[n] + tmpf32  
write_row_and_zero(tsrcdest, m, tmp, tsrcdest.colsb)
```

```
zero_upper_rows(tsrcdest, tsrcdest.rows)  
zero_tileconfig_start()
```

Flags Affected

None.

Exceptions

AMX-E4; see Section 2.10, “Intel® AMX Instruction Exception Classes,” for details.

4. Updates to Chapter 2, Volume 3A

Change bars and violet text show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Updated Table 2-3, "Summary of System Instructions," adding instructions WBNOINVD, INVPCID, RDPID, RDPKRU, WRPKRU, INT, INT1, INT3, INTO, SYSENTER, and SYSCALL.
- Inserted new Section 2.8.2, "Modifying Kernel GS," and Section 2.8.3, "Reading and Writing Protection Key Rights for User Pages," and renumbered subsequent subsections.
- In Section 2.8.6, "Invalidating Caches and TLBs," instructions WBNOINVD and INVPCID were added.
- Previous subsection "Reading Performance-Monitoring Counters and Time-Stamp Counters" was split to Section 2.8.8, "Reading Performance-Monitoring Counters," and Section Section 2.8.9, "Reading Time-Stamp Counter and IA32_TSC_AUX," RDTSCP was added to Section 2.8.9.
- Section 2.8.10, "Reading and Writing Model-Specific Registers," was updated to add RDMSRLIST, WRMSRLIST, and WRMSRNS.

IA-32 architecture (beginning with the Intel386 processor family) provides extensive support for operating-system and system-development software. This support offers multiple modes of operation, which include:

- Real mode, protected mode, virtual 8086 mode, and system management mode. These are sometimes referred to as legacy modes.

Intel 64 architecture supports almost all the system programming facilities available in IA-32 architecture and extends them to a new operating mode (IA-32e mode) that supports a 64-bit programming environment. IA-32e mode allows software to operate in one of two sub-modes:

- 64-bit mode supports 64-bit OS and 64-bit applications
- Compatibility mode allows most legacy software to run; it co-exists with 64-bit applications under a 64-bit OS.

The IA-32 system-level architecture includes features to assist in the following operations:

- Memory management.
- Protection of software modules.
- Multitasking.
- Exception and interrupt handling.
- Multiprocessing.
- Cache management.
- Hardware resource and power management.
- Debugging and performance monitoring.

This chapter provides a description of each part of this architecture. It also describes the system registers that are used to set up and control the processor at the system level and gives a brief overview of the processor's system-level (operating system) instructions.

Many features of the system-level architecture are used only by system programmers. However, application programmers may need to read this chapter and the following chapters in order to create a reliable and secure environment for application programs.

This overview and most subsequent chapters of this book focus on protected-mode operation of the IA-32 architecture. IA-32e mode operation of the Intel 64 architecture, as it differs from protected mode operation, is also described.

All Intel 64 and IA-32 processors enter real-address mode following a power-up or reset (see Chapter 12, "Processor Management and Initialization"). Software then initiates the switch from real-address mode to protected mode. If IA-32e mode operation is desired, software also initiates a switch from protected mode to IA-32e mode.

2.1 OVERVIEW OF THE SYSTEM-LEVEL ARCHITECTURE

System-level architecture consists of a set of registers, data structures, and instructions designed to support basic system-level operations such as memory management, interrupt and exception handling, task management, and control of multiple processors.

Figure 2-1 provides a summary of system registers and data structures that applies to 32-bit modes. System registers and data structures that apply to IA-32e mode are shown in Figure 2-2.

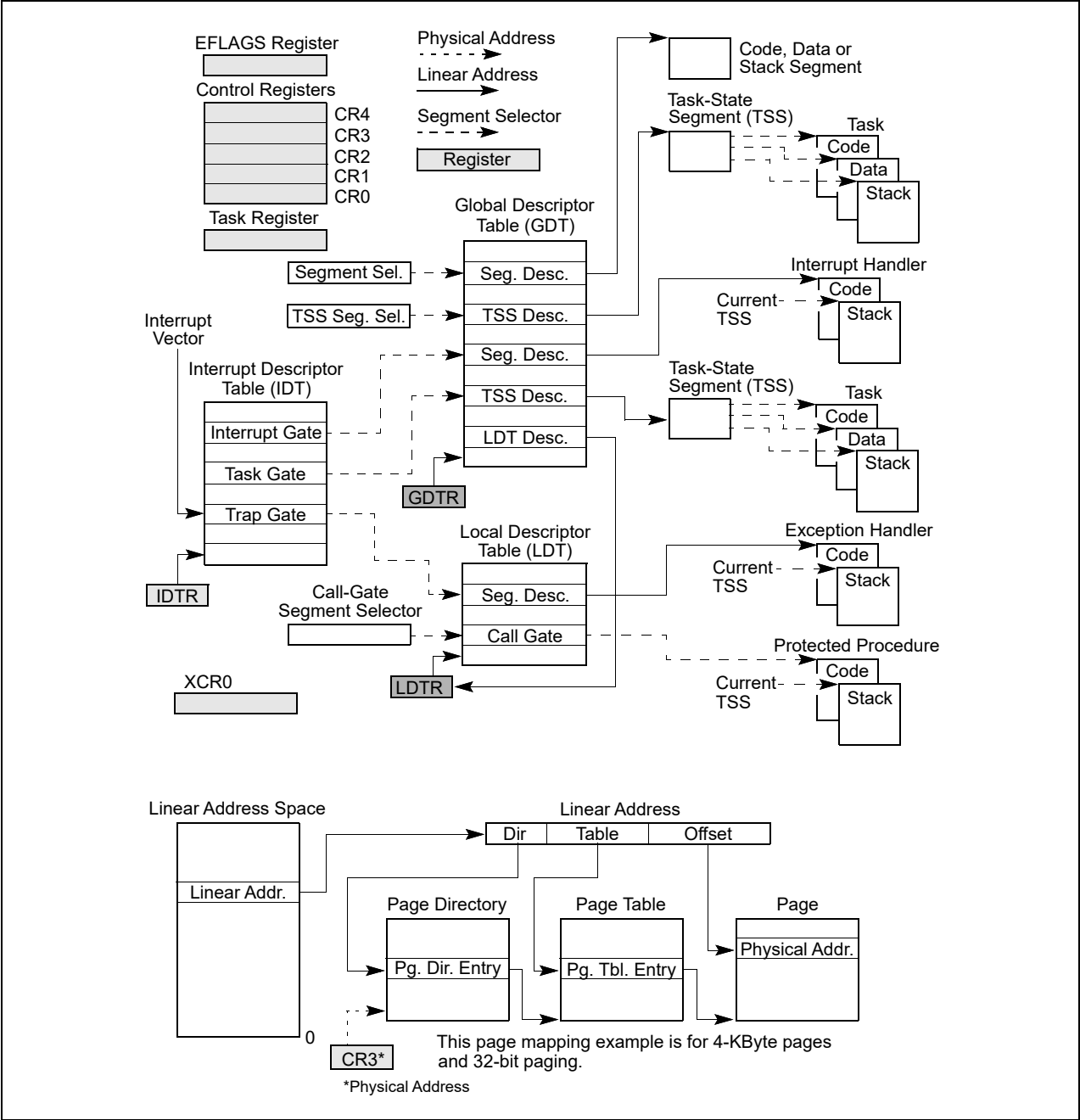


Figure 2-1. IA-32 System-Level Registers and Data Structures

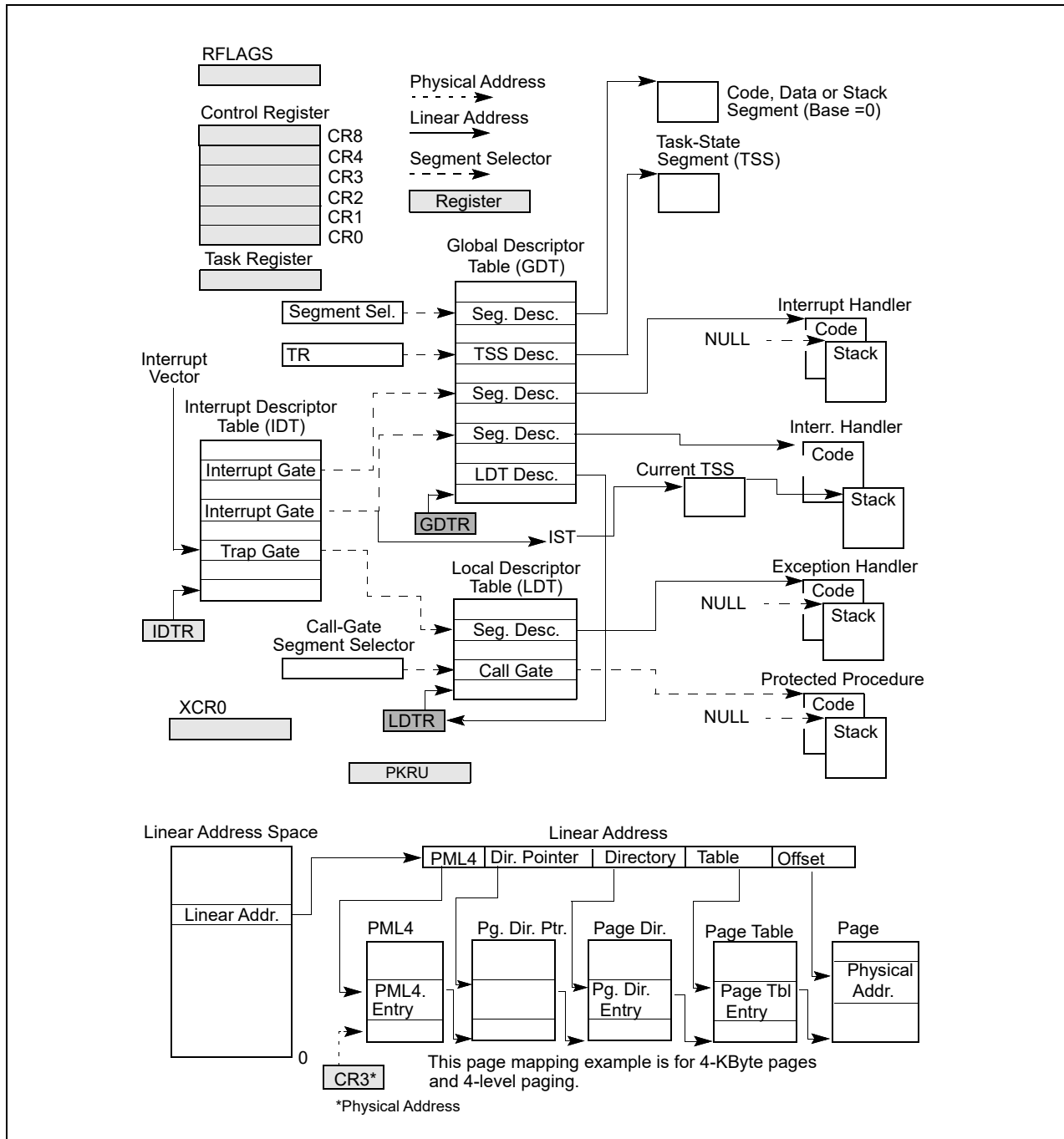


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode and 4-Level Paging

2.1.1 Global and Local Descriptor Tables

When operating in protected mode, all memory accesses pass through either the global descriptor table (GDT) or an optional local descriptor table (LDT) as shown in Figure 2-1. These tables contain entries called segment descriptors. Segment descriptors provide the base address of segments well as access rights, type, and usage information.

Each segment descriptor has an associated segment selector. A segment selector provides the software that uses it with an index into the GDT or LDT (the offset of its associated segment descriptor), a global/local flag (determines whether the selector points to the GDT or the LDT), and access rights information.

To access a byte in a segment, a segment selector and an offset must be supplied. The segment selector provides access to the segment descriptor for the segment (in the GDT or LDT). From the segment descriptor, the processor obtains the base address of the segment in the linear address space. The offset then provides the location of the byte relative to the base address. This mechanism can be used to access any valid code, data, or stack segment, provided the segment is accessible from the current privilege level (CPL) at which the processor is operating. The CPL is defined as the protection level of the currently executing code segment.

See Figure 2-1. The solid arrows in the figure indicate a linear address, dashed lines indicate a segment selector, and the dotted arrows indicate a physical address. For simplicity, many of the segment selectors are shown as direct pointers to a segment. However, the actual path from a segment selector to its associated segment is always through a GDT or LDT.

The linear address of the base of the GDT is contained in the GDT register (GDTR); the linear address of the LDT is contained in the LDT register (LDTR).

2.1.1.1 Global and Local Descriptor Tables in IA-32e Mode

GDTR and LDTR registers are expanded to 64-bits wide in both IA-32e sub-modes (64-bit mode and compatibility mode). For more information: see Section 3.5.2, "Segment Descriptor Tables in IA-32e Mode."

Global and local descriptor tables are expanded in 64-bit mode to support 64-bit base addresses, (16-byte LDT descriptors hold a 64-bit base address and various attributes). In compatibility mode, descriptors are not expanded.

2.1.2 System Segments, Segment Descriptors, and Gates

Besides code, data, and stack segments that make up the execution environment of a program or procedure, the architecture defines two system segments: the task-state segment (TSS) and the LDT. The GDT is not considered a segment because it is not accessed by means of a segment selector and segment descriptor. TSSs and LDTs have segment descriptors defined for them.

The architecture also defines a set of special descriptors called gates (call gates, interrupt gates, trap gates, and task gates). These provide protected gateways to system procedures and handlers that may operate at a different privilege level than application programs and most procedures. For example, a CALL to a call gate can provide access to a procedure in a code segment that is at the same or a numerically lower privilege level (more privileged) than the current code segment. To access a procedure through a call gate, the calling procedure¹ supplies the selector for the call gate. The processor then performs an access rights check on the call gate, comparing the CPL with the privilege level of the call gate and the destination code segment pointed to by the call gate.

If access to the destination code segment is allowed, the processor gets the segment selector for the destination code segment and an offset into that code segment from the call gate. If the call requires a change in privilege level, the processor also switches to the stack for the targeted privilege level. The segment selector for the new stack is obtained from the TSS for the currently running task. Gates also facilitate transitions between 16-bit and 32-bit code segments, and vice versa.

2.1.2.1 Gates in IA-32e Mode

In IA-32e mode, the following descriptors are 16-byte descriptors (expanded to allow a 64-bit base): LDT descriptors, 64-bit TSSs, call gates, interrupt gates, and trap gates.

Call gates facilitate transitions between 64-bit mode and compatibility mode. Task gates are not supported in IA-32e mode. On privilege level changes, stack segment selectors are not read from the TSS. Instead, they are set to NULL.

1. The word "procedure" is commonly used in this document as a general term for a logical unit or block of code (such as a program, procedure, function, or routine).

2.1.3 Task-State Segments and Task Gates

The TSS (see Figure 2-1) defines the state of the execution environment for a task. It includes the state of general-purpose registers, segment registers, the EFLAGS register, the EIP register, and segment selectors with stack pointers for three stack segments (one stack for each privilege level). The TSS also includes the segment selector for the LDT associated with the task and the base address of the paging-structure hierarchy.

All program execution in protected mode happens within the context of a task (called the current task). The segment selector for the TSS for the current task is stored in the task register. The simplest method for switching to a task is to make a call or jump to the new task. Here, the segment selector for the TSS of the new task is given in the CALL or JMP instruction. In switching tasks, the processor performs the following actions:

1. Stores the state of the current task in the current TSS.
2. Loads the task register with the segment selector for the new task.
3. Accesses the new TSS through a segment descriptor in the GDT.
4. Loads the state of the new task from the new TSS into the general-purpose registers, the segment registers, the LDTR, control register CR3 (base address of the paging-structure hierarchy), the EFLAGS register, and the EIP register.
5. Begins execution of the new task.

A task can also be accessed through a task gate. A task gate is similar to a call gate, except that it provides access (through a segment selector) to a TSS rather than a code segment.

2.1.3.1 Task-State Segments in IA-32e Mode

Hardware task switches are not supported in IA-32e mode. However, TSSs continue to exist. The base address of a TSS is specified by its descriptor.

A 64-bit TSS holds the following information that is important to 64-bit operation:

- Stack pointer addresses for each privilege level.
- Pointer addresses for the interrupt stack table.
- Offset address of the IO-permission bitmap (from the TSS base).

The task register is expanded to hold 64-bit base addresses in IA-32e mode. See also: Section 10.7, “Task Management in 64-bit Mode.”

2.1.4 Interrupt and Exception Handling

NOTE

This section focuses on event handling based on IDT event delivery and on return using the IRET instruction. FRED transitions are alternatives to these mechanisms. In general, the material in this section does not apply when FRED transitions are enabled. See Chapter 8, “Flexible Return and Event Delivery (FRED),” for more information about FRED transitions.

External interrupts, software interrupts and exceptions are handled through the interrupt descriptor table (IDT). The IDT stores a collection of gate descriptors that provide access to interrupt and exception handlers. Like the GDT, the IDT is not a segment. The linear address for the base of the IDT is contained in the IDT register (IDTR).

Gate descriptors in the IDT can be interrupt, trap, or task gate descriptors. To access an interrupt or exception handler, the processor first receives an interrupt vector from internal hardware, an external interrupt controller, or from software by means of an INT *n*, INTO, INT3, INT1, or BOUND instruction. The interrupt vector provides an index into the IDT. If the selected gate descriptor is an interrupt gate or a trap gate, the associated handler procedure is accessed in a manner similar to calling a procedure through a call gate. If the descriptor is a task gate, the handler is accessed through a task switch.

2.1.4.1 Interrupt and Exception Handling IA-32e Mode

In IA-32e mode, interrupt gate descriptors are expanded to 16 bytes to support 64-bit base addresses. This is true for 64-bit mode and compatibility mode.

The IDTR register is expanded to hold a 64-bit base address. Task gates are not supported.

2.1.5 Memory Management

System architecture supports either direct physical addressing of memory or virtual memory (through paging). When physical addressing is used, a linear address is treated as a physical address. When paging is used: all code, data, stack, and system segments (including the GDT and IDT) can be paged with only the most recently accessed pages being held in physical memory.

The location of pages (sometimes called page frames) in physical memory is contained in the paging structures. These structures reside in physical memory (see Figure 2-1 for the case of 32-bit paging).

The base physical address of the paging-structure hierarchy is contained in control register CR3. The entries in the paging structures determine the physical address of the base of a page frame, access rights and memory management information.

To use this paging mechanism, a linear address is broken into parts. The parts provide separate offsets into the paging structures and the page frame. A system can have a single hierarchy of paging structures or several. For example, each task can have its own hierarchy.

2.1.5.1 Memory Management in IA-32e Mode

In IA-32e mode, physical memory pages are managed by a set of system data structures. In both compatibility mode and 64-bit mode, four or five levels of system data structures are used (see Chapter 5, "Paging"). These include the following:

- **The page map level 5 (PML5)** — An entry in the PML5 table contains the physical address of the base of a PML4 table, access rights, and memory management information. The base physical address of the PML5 table is stored in CR3. The PML5 table is used only with 5-level paging.
- **A page map level 4 (PML4)** — An entry in a PML4 table contains the physical address of the base of a page directory pointer table, access rights, and memory management information. With 4-level paging, there is only one PML4 table and its base physical address is stored in CR3.
- **A set of page directory pointer tables** — An entry in a page directory pointer table contains the physical address of the base of a page directory table, access rights, and memory management information.
- **Sets of page directories** — An entry in a page directory table contains the physical address of the base of a page table, access rights, and memory management information.
- **Sets of page tables** — An entry in a page table contains the physical address of a page frame, access rights, and memory management information.

2.1.6 System Registers

To assist in initializing the processor and controlling system operations, the system architecture provides system flags in the EFLAGS register and several system registers:

- The system flags and IOPL field in the EFLAGS register control task and mode switching, interrupt handling, instruction tracing, and access rights. See also: Section 2.3, "System Flags and Fields in the EFLAGS Register."
- The control registers (CR0, CR2, CR3, and CR4) contain a variety of flags and data fields for controlling system-level operations. Other flags in these registers are used to indicate support for specific processor capabilities within the operating system or executive. See also: Chapter 2, "Control Registers," and Section 2.6, "Extended Control Registers (Including XCR0)."
- The debug registers (not shown in Figure 2-1) allow the setting of breakpoints for use in debugging programs and systems software. See also: Chapter 20, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features."

- The GDTR, LDTR, and IDTR registers contain the linear addresses and sizes (limits) of their respective tables. See also: Section 2.4, “Memory-Management Registers.”
- The task register contains the linear address and size of the TSS for the current task. See also: Section 2.4, “Memory-Management Registers.”
- Model-specific registers (not shown in Figure 2-1).

The model-specific registers (MSRs) are a group of registers available primarily to operating-system or executive procedures (that is, code running at privilege level 0). These registers control items such as the debug extensions, the performance-monitoring counters, the machine-check architecture, and the memory type ranges (MTRRs).

The number and function of these registers varies among different members of the Intel 64 and IA-32 processor families. See also: Section 12.4, “Model-Specific Registers (MSRs),” and Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

Most systems restrict access to system registers (other than the EFLAGS register) by application programs. Systems can be designed, however, where all programs and procedures run at the most privileged level (privilege level 0). In such a case, application programs would be allowed to modify the system registers.

2.1.6.1 System Registers in IA-32e Mode

In IA-32e mode, the four system-descriptor-table registers (GDTR, IDTR, LDTR, and TR) are expanded in hardware to hold 64-bit base addresses. EFLAGS becomes the 64-bit RFLAGS register. CR0–CR4 are expanded to 64 bits. CR8 becomes available. CR8 provides read-write access to the task priority register (TPR) so that the operating system can control the priority classes of external interrupts.

In 64-bit mode, debug registers DR0–DR7 are 64 bits. In compatibility mode, address-matching in DR0–DR3 is also done at 64-bit granularity.

On systems that support IA-32e mode, the extended feature enable register (IA32_EFER) is available. This model-specific register controls activation of IA-32e mode and other IA-32e mode operations. In addition, there are several model-specific registers that govern IA-32e mode instructions:

- **IA32_KERNEL_GS_BASE** — Used by SWAPGS instruction.
- **IA32_LSTAR** — Used by SYSCALL instruction.
- **IA32_FMASK** — Used by SYSCALL instruction.
- **IA32_STAR** — Used by SYSCALL and SYSRET instruction.

Another set of MSRs control FRED transitions when they are enabled. These are described in Section 8.2.

2.1.7 Other System Resources

Besides the system registers and data structures described in the previous sections, system architecture provides the following additional resources:

- Operating system instructions (see also: Section 2.8, “System Instruction Summary”).
- Performance-monitoring counters (not shown in Figure 2-1).
- Internal caches and buffers (not shown in Figure 2-1).

Performance-monitoring counters are event counters that can be programmed to count processor events such as the number of instructions decoded, the number of interrupts received, or the number of cache loads.

The processor provides several internal caches and buffers. The caches are used to store both data and instructions. The buffers are used to store things like decoded addresses to system and application segments and write operations waiting to be performed. See also: Chapter 14, “Memory Cache Control.”

2.2 MODES OF OPERATION

The IA-32 architecture supports three operating modes and one quasi-operating mode:

- **Protected mode** — This is the native operating mode of the processor. It provides a rich set of architectural features, flexibility, high performance and backward compatibility to existing software base.
- **Real-address mode** — This operating mode provides the programming environment of the Intel 8086 processor, with a few extensions (such as the ability to switch to protected or system management mode).
- **System management mode (SMM)** — SMM is a standard architectural feature in all IA-32 processors, beginning with the Intel386 SL processor. This mode provides an operating system or executive with a transparent mechanism for implementing power management and OEM differentiation features. SMM is entered through activation of an external system interrupt pin (SMI#), which generates a system management interrupt (SMI). In SMM, the processor switches to a separate address space while saving the context of the currently running program or task. SMM-specific code may then be executed transparently. Upon returning from SMM, the processor is placed back into its state prior to the SMI.
- **Virtual-8086 mode** — In protected mode, the processor supports a quasi-operating mode known as virtual-8086 mode. This mode allows the processor execute 8086 software in a protected, multitasking environment.

Intel 64 architecture supports all operating modes of IA-32 architecture and IA-32e modes:

- **IA-32e mode** — In IA-32e mode, the processor supports two sub-modes: compatibility mode and 64-bit mode. 64-bit mode provides 64-bit linear addressing and support for physical address space larger than 64 GBytes. Compatibility mode allows most legacy protected-mode applications to run unchanged.

Figure 2-3 shows how the processor moves between operating modes.

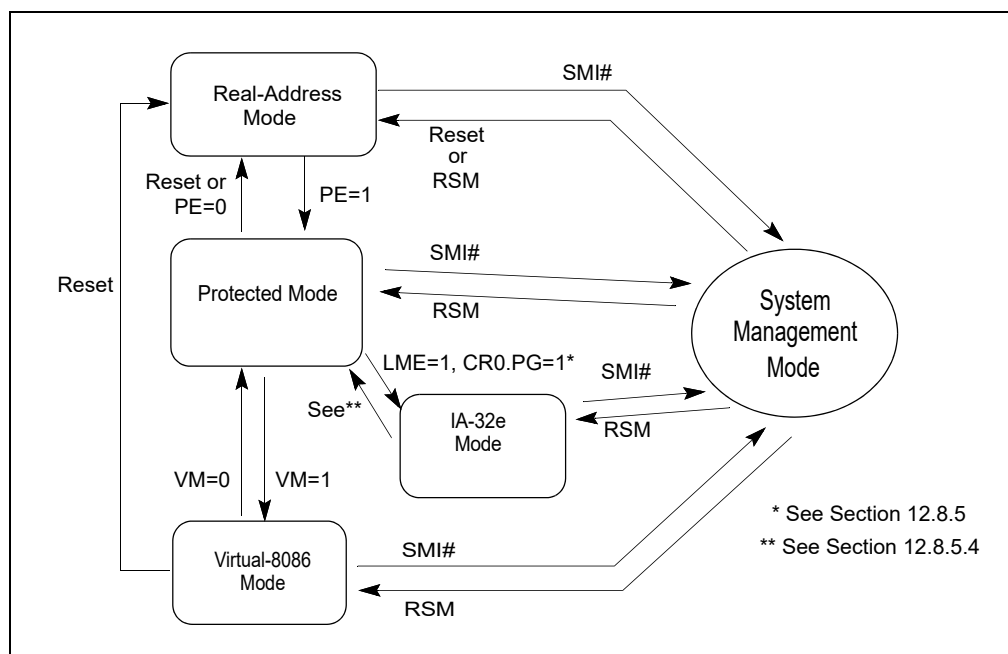


Figure 2-3. Transitions Among the Processor's Operating Modes

The processor is placed in real-address mode following power-up or a reset. The PE flag in control register CR0 then controls whether the processor is operating in real-address or protected mode. See also: Section 12.9, "Mode Switching," and Section 5.1.2, "Paging-Mode Enabling."

The VM flag in the EFLAGS register determines whether the processor is operating in protected mode or virtual-8086 mode. Transitions between protected mode and virtual-8086 mode are generally carried out as part of a task switch or a return from an interrupt or exception handler. See also: Section 23.2.5, "Entering Virtual-8086 Mode."

The LMA bit (IA32_EFER.LMA[10]) determines whether the processor is operating in IA-32e mode. When running in IA-32e mode, 64-bit or compatibility sub-mode operation is determined by CS.L bit of the code segment. The processor enters into IA-32e mode from protected mode by enabling paging and setting the LME bit (IA32_EFER.LME[8]). See also: Chapter 12, "Processor Management and Initialization."

The processor switches to SMM whenever it receives an SMI while the processor is in real-address, protected, virtual-8086, or IA-32e modes. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

2.2.1 Extended Feature Enable Register

The IA32_EFER MSR provides several fields related to IA-32e mode enabling and operation. It also provides one field that relates to page-access right modification (see Section 5.6, “Access Rights”). The layout of the IA32_EFER MSR is shown in Figure 2-4.

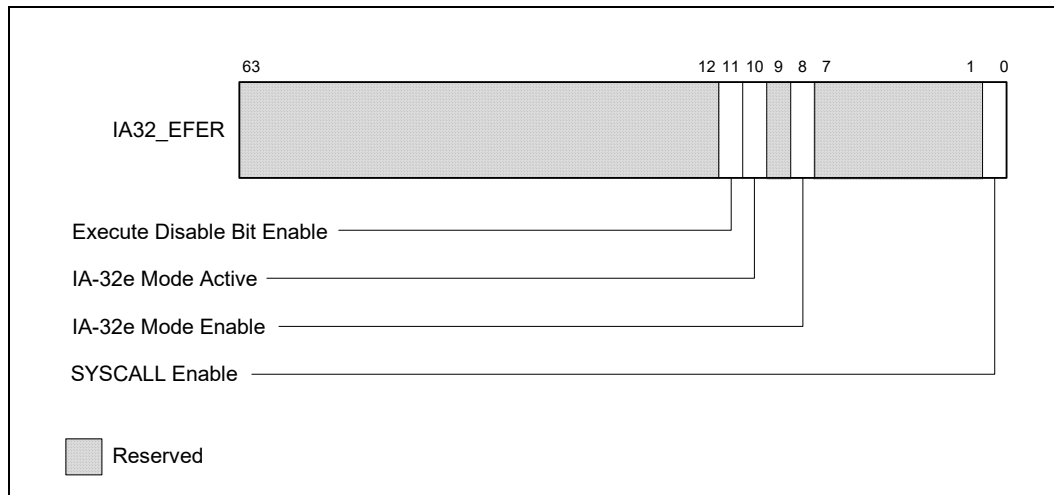


Figure 2-4. IA32_EFER MSR Layout

Table 2-1. IA32_EFER MSR Information

Bit	Description
0	SYSCALL Enable: IA32_EFER.SCE (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.
7:1	Reserved.
8	IA-32e Mode Enable: IA32_EFER.LME (R/W) Enables IA-32e mode operation.
9	Reserved.
10	IA-32e Mode Active: IA32_EFER.LMA (R) Indicates IA-32e mode is active when set.
11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W) Enables page access restriction by preventing instruction fetches from PAE pages with the XD bit set (See Section 5.6).
63:12	Reserved.

2.3 SYSTEM FLAGS AND FIELDS IN THE EFLAGS REGISTER

The system flags and IOPL field of the EFLAGS register control I/O, maskable hardware interrupts, debugging, task switching, and the virtual-8086 mode (see Figure 2-5). Only privileged code (typically operating system or executive code) should be allowed to modify these bits.

The system flags and IOPL are:

TF **Trap (bit 8)** — Set to enable single-step mode for debugging; clear to disable single-step mode. In single-step mode, the processor generates a debug exception after each instruction. This allows the execution state of a program to be inspected after each instruction. If an application program sets the TF flag using a POPF, POPFD, or IRET instruction, a debug exception is generated after the instruction that follows the POPF, POPFD, or IRET.

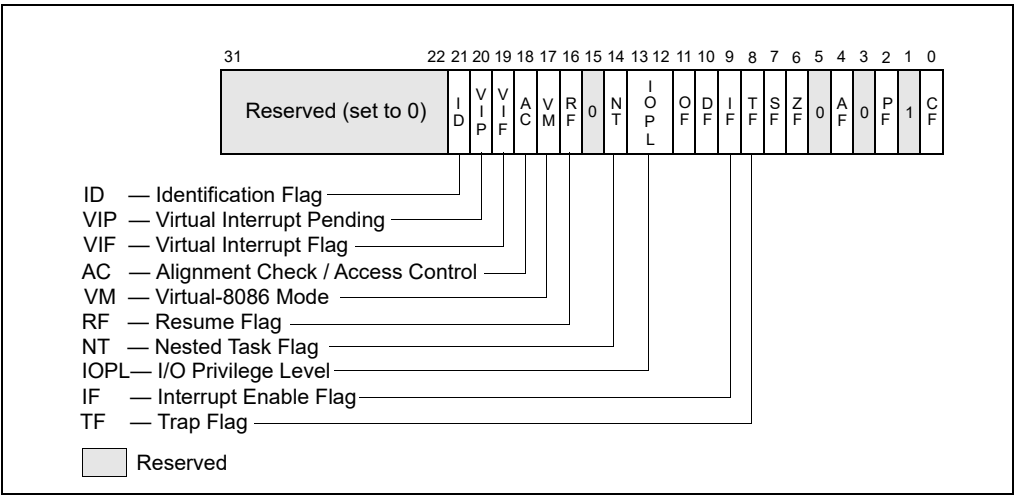


Figure 2-5. System Flags in the EFLAGS Register

IF **Interrupt enable (bit 9)** — Controls the response of the processor to maskable hardware interrupt requests (see also: Section 7.3.2, “Maskable Hardware Interrupts”). The flag is set to respond to maskable hardware interrupts; cleared to inhibit maskable hardware interrupts. The IF flag does not affect the generation of exceptions or nonmaskable interrupts (NMI interrupts). The CPL, IOPL, and the state of the VME flag in control register CR4 determine whether the IF flag can be modified by the CLI, STI, POPF, POPFD, and IRET.

IOPL **I/O privilege level field (bits 12 and 13)** — Indicates the I/O privilege level (IOPL) of the currently running program or task. The CPL of the currently running program or task must be less than or equal to the IOPL to access the I/O address space. The POPF and IRET instructions can modify this field only when operating at a CPL of 0.

The IOPL is also one of the mechanisms that controls the modification of the IF flag and the handling of interrupts in virtual-8086 mode when virtual mode extensions are in effect (when CR4.VME = 1). See also: Chapter 20, “Input/Output,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

NT **Nested task (bit 14)** — Controls the chaining of interrupted and called tasks. The processor sets this flag on calls to a task initiated with a CALL instruction, an interrupt, or an exception. It examines and modifies this flag on returns from a task initiated with the IRET instruction. The flag can be explicitly set or cleared with the POPF/POPF instructions; however, changing to the state of this flag can generate unexpected exceptions in application programs.

See also: Section 10.4, “Task Linking.”

RF **Resume (bit 16)** — Controls the processor’s response to instruction-breakpoint conditions. When set, this flag temporarily disables debug exceptions (#DB) from being generated for instruction breakpoints (although other exception conditions can cause an exception to be generated). When clear, instruction breakpoints will generate debug exceptions.

The primary function of the RF flag is to allow the restarting of an instruction following a debug exception that was caused by an instruction breakpoint condition. Here, debug software must set this flag in the EFLAGS image on the stack just prior to returning to the interrupted program with IRETD (to prevent the instruction breakpoint from causing another debug exception). The processor then automatically clears

this flag after the instruction returned to has been successfully executed, enabling instruction breakpoint faults again.

See also: Section 20.3.1.1, “Instruction-Breakpoint Exception Condition.”

VM **Virtual-8086 mode (bit 17)** — Set to enable virtual-8086 mode; clear to return to protected mode.

See also: Section 23.2.1, “Enabling Virtual-8086 Mode.”

AC **Alignment check or access control (bit 18)** — If the AM bit is set in the CR0 register, alignment checking of user-mode data accesses is enabled if and only if this flag is 1. An alignment-check exception is generated when reference is made to an unaligned operand, such as a word at an odd byte address or a doubleword at an address which is not an integral multiple of four. Alignment-check exceptions are generated only in user mode (privilege level 3). Memory references that default to privilege level 0, such as segment descriptor loads, do not generate this exception even when caused by instructions executed in user-mode.

The alignment-check exception can be used to check alignment of data. This is useful when exchanging data with processors which require all data to be aligned. The alignment-check exception can also be used by interpreters to flag some pointers as special by misaligning the pointer. This eliminates overhead of checking each pointer and only handles the special pointer when used.

If the SMAP bit is set in the CR4 register, explicit supervisor-mode data accesses to user-mode pages are allowed if and only if this bit is 1. See Section 5.6, “Access Rights.”

VIF **Virtual Interrupt (bit 19)** — Contains a virtual image of the IF flag. This flag is used in conjunction with the VIP flag. The processor only recognizes the VIF flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. (The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.)

See also: Section 23.3.3.5, “Method 6: Software Interrupt Handling,” and Section 23.4, “Protected-Mode Virtual Interrupts.”

VIP **Virtual interrupt pending (bit 20)** — Set by software to indicate that an interrupt is pending; cleared to indicate that no interrupt is pending. This flag is used in conjunction with the VIF flag. The processor reads this flag but never modifies it. The processor only recognizes the VIP flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.

See Section 23.3.3.5, “Method 6: Software Interrupt Handling,” and Section 23.4, “Protected-Mode Virtual Interrupts.”

ID **Identification (bit 21)** — The ability of a program or procedure to set or clear this flag indicates support for the CPUID instruction.

2.3.1 System Flags and Fields in IA-32e Mode

In 64-bit mode, the RFLAGS register expands to 64 bits with the upper 32 bits reserved. System flags in RFLAGS (64-bit mode) or EFLAGS (compatibility mode) are shown in Figure 2-5.

In IA-32e mode, the processor does not allow the VM bit to be set because virtual-8086 mode is not supported (attempts to set the bit are ignored). Also, the processor will not set the NT bit. The processor does, however, allow software to set the NT bit (note that an IRET causes a general protection fault in IA-32e mode if the NT bit is set).

In IA-32e mode, the SYSCALL/SYSRET instructions have a programmable method of specifying which bits are cleared in RFLAGS/EFLAGS. These instructions save/restore EFLAGS/RFLAGS.

2.4 MEMORY-MANAGEMENT REGISTERS

The processor provides four memory-management registers (GDTR, LDTR, IDTR, and TR) that specify the locations of the data structures which control segmented memory management (see Figure 2-6). Special instructions are provided for loading and storing these registers.

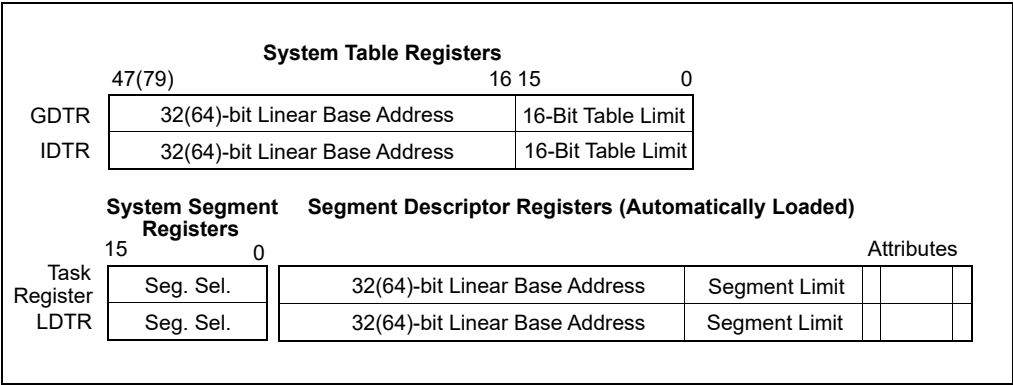


Figure 2-6. Memory Management Registers

2.4.1 Global Descriptor Table Register (GDTR)

The GDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and the 16-bit table limit for the GDT. The base address specifies the linear address of byte 0 of the GDT; the table limit specifies the number of bytes in the table.

The LGDT and SGDT instructions load and store the GDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. A new base address must be loaded into the GDTR as part of the processor initialization process for protected-mode operation.

See also: Section 3.5.1, "Segment Descriptor Tables."

2.4.2 Local Descriptor Table Register (LDTR)

The LDTR register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the LDT. The base address specifies the linear address of byte 0 of the LDT segment; the segment limit specifies the number of bytes in the segment. See also: Section 3.5.1, "Segment Descriptor Tables."

The LLDT and SLDT instructions load and store the segment selector part of the LDTR register, respectively. The segment that contains the LDT must have a segment descriptor in the GDT. When the LLDT instruction loads a segment selector in the LDTR: the base address, limit, and descriptor attributes from the LDT descriptor are automatically loaded in the LDTR.

When a task switch occurs, the LDTR is automatically loaded with the segment selector and descriptor for the LDT for the new task. The contents of the LDTR are not automatically saved prior to writing the new LDT information into the register.

On power up or reset of the processor, the segment selector and base address are set to the default value of 0 and the limit is set to 0FFFFH.

2.4.3 IDTR Interrupt Descriptor Table Register

The IDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and 16-bit table limit for the IDT. The base address specifies the linear address of byte 0 of the IDT; the table limit specifies the number of bytes in the table. The LIDT and SIDT instructions load and store the IDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. The base address and limit in the register can then be changed as part of the processor initialization process.

See also: Section 7.10, "Interrupt Descriptor Table (IDT)."

2.4.4 Task Register (TR)

The task register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the TSS of the current task. The selector references the TSS descriptor in the GDT. The base address specifies the linear address of byte 0 of the TSS; the segment limit specifies the number of bytes in the TSS. See also: Section 10.2.4, “Task Register.”

The LTR and STR instructions load and store the segment selector part of the task register, respectively. When the LTR instruction loads a segment selector in the task register, the base address, limit, and descriptor attributes from the TSS descriptor are automatically loaded into the task register. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH.

When a task switch occurs, the task register is automatically loaded with the segment selector and descriptor for the TSS for the new task. The contents of the task register are not automatically saved prior to writing the new TSS information into the register.

2.5 CONTROL REGISTERS

Control registers (CR0, CR1, CR2, CR3, and CR4; see Figure 2-7) determine operating mode of the processor and the characteristics of the currently executing task. These registers are 32 bits in all 32-bit modes and compatibility mode.

In 64-bit mode, control registers are expanded to 64 bits. The MOV CRn instructions are used to manipulate the register bits. Operand-size prefixes for these instructions are ignored. The following is also true:

- The control registers can be read and loaded (or modified) using the move-to-or-from-control-registers forms of the MOV instruction. In protected mode, the MOV instructions allow the control registers to be read or loaded (at privilege level 0 only). This restriction means that application programs or operating-system procedures (running at privilege levels 1, 2, or 3) are prevented from reading or loading the control registers.
- Some of the bits in CR0 and CR4 are reserved and must be written with zeros. Attempting to set any reserved bits in CR0[31:0] is ignored. Attempting to set any reserved bits in CR0[63:32] results in a general-protection exception, #GP(0). Attempting to set any reserved bits in CR4 results in a general-protection exception, #GP(0).
- All 64 bits of CR2 are writable by software.
- Bits in CR3 in the range 63:MAXPHYADDR that are reserved (see Figure 2-7) must be zero. Attempting to set any of them results in #GP(0).¹
- The MOV CR2 instruction does not check that address written to CR2 is canonical.
- A 64-bit capable processor will retain the upper 32 bits of each control register when transitioning out of IA-32e mode.
- On a 64-bit capable processor, an execution of MOV to CR outside of 64-bit mode zeros the upper 32 bits of the control register.
- Register CR8 is available in 64-bit mode only.

The control registers are summarized below, and each architecturally defined control field in these control registers is described individually. In Figure 2-7, the width of the register in 64-bit mode is indicated in parenthesis (except for CR0).

- **CR0** — Contains system control flags that control operating mode and states of the processor.
- **CR1** — Reserved.
- **CR2** — Contains the page-fault linear address (the linear address that caused a page fault).
- **CR3** — Contains the physical address of the base of the paging-structure hierarchy and four flags (PWT, PCD, LAM_U57, and LAM_U48). Only the most-significant bits (less the lower 12 bits) of the base address are specified; the lower 12 bits of the address are assumed to be 0. The first paging structure must thus be aligned

1. MAXPHYADDR is derived from the value enumerated by CPUID.80000008H:EAX[7:0]. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is out-side secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.

to a page (4-KByte) boundary. The PCD and PWT flags control caching of that paging structure in the processor's internal data caches (they do not control TLB caching of page-directory information). The LAM_U57 and LAM_U48 flags control linear-address masking for user addresses (see Section 4.4, "Linear-Address Masking").

When using the physical address extension, the CR3 register contains the base address of the page-directory-pointer table. With 4-level paging and 5-level paging, the CR3 register contains the base address of the PML4 table and PML5 table, respectively. If PCIDs are enabled, CR3 has a format different from that illustrated in Figure 2-7. See Section 5.5, "4-Level Paging and 5-Level Paging."

See also: Chapter 5, "Paging."

- **CR4** — Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities. Bits CR4[63:32] can only be used for IA-32e mode only features that are enabled after entering 64-bit mode. Bits CR4[31:0] do not have any effect outside of IA-32e mode.
- **CR8** — Provides (in CR8[3:0]) read and write access to bits 7:4 of the local APIC's Task-Priority Register (TPR). These bits are the **task-priority class**; all interrupts in this and lower priority classes are blocked. A value of 15 blocks interrupts in all priority classes, while a value of 0 blocks none. CR8 can be accessed only in 64-bit mode, but the value of the TPR blocks interrupts regardless of mode.

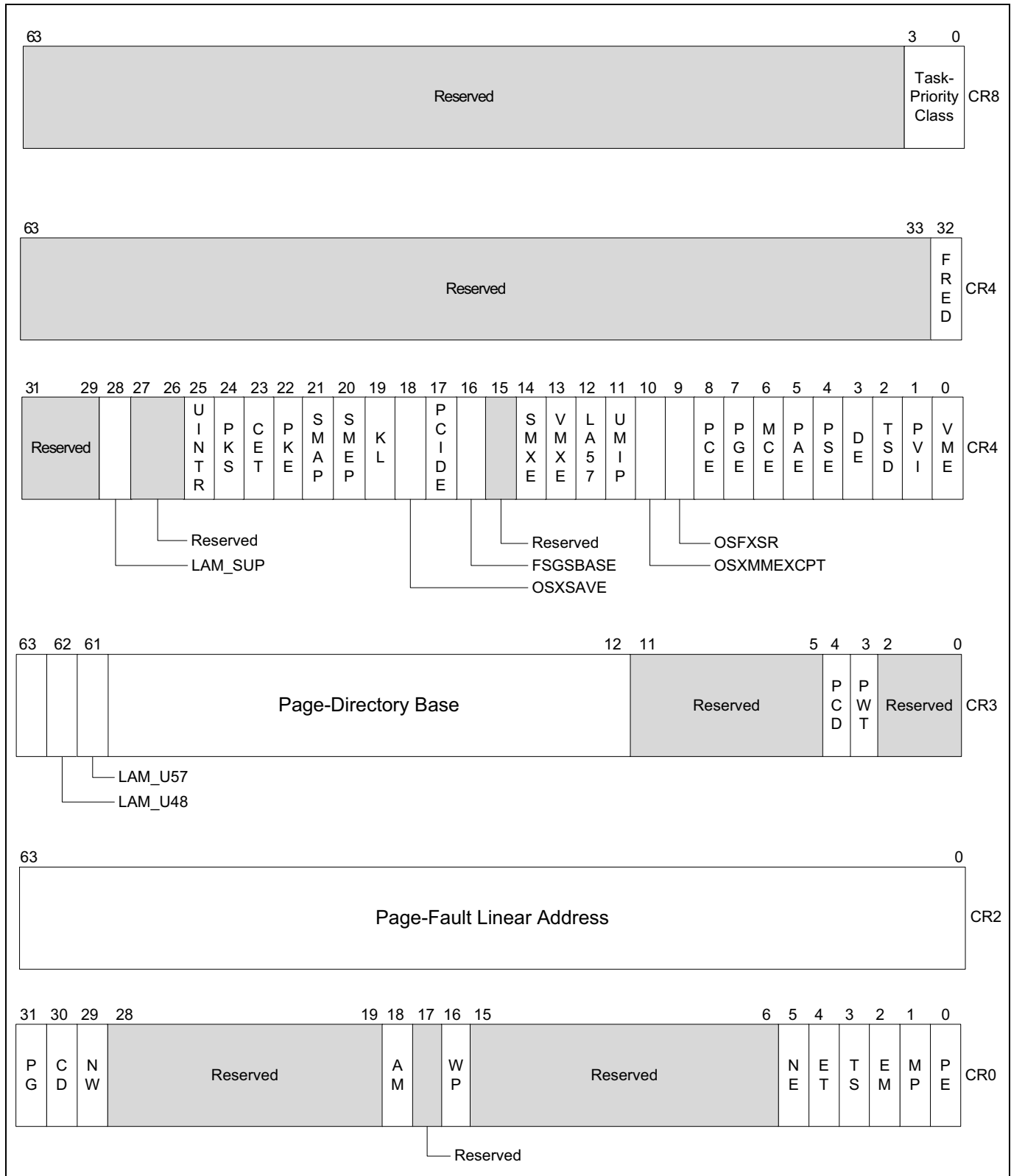


Figure 2-7. Control Registers

The flags in control registers are:

CR0.PG

Paging (bit 31 of CR0) — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Chapter 5, “Paging.”

On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.

CR0.CD

Cache Disable (bit 30 of CR0) — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor’s internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 14-5. To prevent the processor from accessing and updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.

See also: Section 14.5.3, “Preventing Caching,” and Section 14.5, “Cache Control.”

CR0.NW

Not Write-through (bit 29 of CR0) — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 14-5 for detailed information about the effect of the NW flag on caching for other settings of the CD and NW flags.

CR0.AM

Alignment Mask (bit 18 of CR0) — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

CR0.WP

Write Protect (bit 16 of CR0) — When set, inhibits supervisor-level procedures from writing into read-only pages; when clear, allows supervisor-level procedures to write into read-only pages (regardless of the U/S bit setting; see Section 5.1.3 and Section 5.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX. This flag must be set before software can set CR4.CET, and it cannot be cleared as long as CR4.CET = 1 (see below).

CR0.NE

Numeric Error (bit 5 of CR0) — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.

The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting. Using FERR# and IGNNE# to handle floating-point exceptions is deprecated by modern operating systems; this non-native approach also limits newer processors to operate with one logical processor active.

See also: Section 8.7, “Handling x87 FPU Exceptions in Software,” in Chapter 8, “Programming with the x87 FPU,” and Appendix A, “EFLAGS Cross-Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

CR0.ET

Extension Type (bit 4 of CR0) — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.

CR0.TS

Task Switched (bit 3 of CR0) — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context on a task switch to be delayed until an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction is

actually executed by the new task. The processor sets this flag on every task switch and tests it when executing x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

- If the TS flag is set and the EM flag (bit 2 of CR0) is clear, a device-not-available exception (#NM) is raised prior to the execution of any x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction; with the exception of PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. See the paragraph below for the special case of the WAIT/FWAIT instructions.
- If the TS flag is set and the MP flag (bit 1 of CR0) and EM flag are clear, an #NM exception is not raised prior to the execution of an x87 FPU WAIT/FWAIT instruction.
- If the EM flag is set, the setting of the TS flag has no effect on the execution of x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

Table 2-2 shows the actions taken when the processor encounters an x87 FPU instruction based on the settings of the TS, EM, and MP flags. Table 15-1 and 16-1 show the actions taken when the processor encounters an MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction.

The processor does not automatically save the context of the x87 FPU, XMM, and MXCSR registers on a task switch. Instead, it sets the TS flag, which causes the processor to raise an #NM exception whenever it encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction in the instruction stream for the new task (with the exception of the instructions listed above).

The fault handler for the #NM exception can then be used to clear the TS flag (with the CLTS instruction) and save the context of the x87 FPU, XMM, and MXCSR registers. If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction, the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context is never saved.

Table 2-2. Action Taken By x87 FPU Instructions for Different Combinations of EM, MP, and TS

CR0 Flags			x87 FPU Instruction Type	
EM	MP	TS	Floating-Point	WAIT/FWAIT
0	0	0	Execute	Execute.
0	0	1	#NM Exception	Execute.
0	1	0	Execute	Execute.
0	1	1	#NM Exception	#NM exception.
1	0	0	#NM Exception	Execute.
1	0	1	#NM Exception	Execute.
1	1	0	#NM Exception	Execute.
1	1	1	#NM Exception	#NM exception.

CR0.EM

Emulation (bit 2 of CR0) — Indicates that the processor does not have an internal or external x87 FPU when set; indicates an x87 FPU is present when clear. This flag also affects the execution of MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

When the EM flag is set, execution of an x87 FPU instruction generates a device-not-available exception (#NM). This flag must be set when the processor does not have an internal x87 FPU or is not connected to an external math coprocessor. Setting this flag forces all floating-point instructions to be handled by software emulation. Table 12-3 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the EM, MP, and TS flags.

Also, when the EM flag is set, execution of an MMX instruction causes an invalid-opcode exception (#UD) to be generated (see Table 15-1). Thus, if an IA-32 or Intel 64 processor incorporates MMX technology, the EM flag must be set to 0 to enable execution of MMX instructions.

Similarly for SSE/SSE2/SSE3/SSSE3/SSE4 extensions, when the EM flag is set, execution of most SSE/SSE2/SSE3/SSSE3/SSE4 instructions causes an invalid opcode exception (#UD) to be generated (see

Table 16-1). If an IA-32 or Intel 64 processor incorporates the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the EM flag must be set to 0 to enable execution of these extensions. SSE/SSE2/SSE3/SSSE3/SSE4 instructions not affected by the EM flag include: PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

CR0.MP

Monitor Coprocessor (bit 1 of CR0) — Controls the interaction of the WAIT (or FWAIT) instruction with the TS flag (bit 3 of CR0). If the MP flag is set, a WAIT instruction generates a device-not-available exception (#NM) if the TS flag is also set. If the MP flag is clear, the WAIT instruction ignores the setting of the TS flag. Table 12-3 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the MP, EM, and TS flags.

CR0.PE

Protection Enable (bit 0 of CR0) — Enables protected mode when set; enables real-address mode when clear. This flag does not enable paging directly. It only enables segment-level protection. To enable paging, both the PE and PG flags must be set.

See also: Section 12.9, “Mode Switching.”

CR3.LAM_U48

User LAM48 enable (bit 62 of CR3) — When set and CR3.LAM_U57 is clear, enables LAM48 (masking of linear-address bits 62:48) for user pointers. See Section 4.4, “Linear-Address Masking.”

CR3.LAM_U57

User LAM57 enable (bit 61 of CR3) — When set, enables LAM57 (masking of linear-address bits 62:57) for user pointers and overrides CR3.LAM_U48. See Section 4.4, “Linear-Address Masking.”

CR3.PCD

Page-level Cache Disable (bit 4 of CR3) — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 5.9, “Paging and Memory Typing.” This bit is not used if paging is disabled, with PAE paging, or with 4-level paging¹ or 5-level paging if CR4.PCIDE=1.

CR3.PWT

Page-level Write-Through (bit 3 of CR3) — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 5.9, “Paging and Memory Typing.” This bit is not used if paging is disabled, with PAE paging, or with 4-level paging or 5-level paging if CR4.PCIDE=1.

CR4.VME

Virtual-8086 Mode Extensions (bit 0 of CR4) — Enables interrupt- and exception-handling extensions in virtual-8086 mode when set; disables the extensions when clear. Use of the virtual mode extensions can improve the performance of virtual-8086 applications by eliminating the overhead of calling the virtual-8086 monitor to handle interrupts and exceptions that occur while executing an 8086 program and, instead, redirecting the interrupts and exceptions back to the 8086 program’s handlers. It also provides hardware support for a virtual interrupt flag (VIF) to improve reliability of running 8086 programs in multi-tasking and multiple-processor environments.

See also: Section 23.3, “Interrupt and Exception Handling in Virtual-8086 Mode.”

CR4.FRED

FRED enable (bit 32 of CR4) — When set, enables FRED transitions. See Chapter 8, “Flexible Return and Event Delivery (FRED).”

CR4.LAM_SUP

Supervisor LAM enable (bit 28 of CR4) — When set, enables LAM (linear-address masking) for supervisor pointers. See Section 4.4, “Linear-Address Masking.”

CR4.LASS

Linear-address-space Separation (bit 27 of CR4) — When set, enables LASS (linear-address-space separation). See Section 4.3, “Linear-Address-Space Separation (LASS).”

CR4.UINTR

User Interrupts Enable Bit (bit 25 of CR4) — Enables user interrupts when set, including user-interrupt delivery, user-interrupt notification identification, and the user-interrupt instructions.

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

CR4.PKS

Enable protection keys for supervisor-mode pages (bit 24 of CR4) — 4-level paging and 5-level paging associate each supervisor-mode linear address with a protection key. When set, this flag allows use of the IA32_PKRS MSR to specify, for each protection key, whether supervisor-mode linear addresses with that protection key can be read or written.

CR4.CET

Control-flow Enforcement Technology (bit 23 of CR4) — Enables control-flow enforcement technology when set. See Chapter 18, “Control-flow Enforcement Technology (CET),” of the *IA-32 Intel® Architecture Software Developer’s Manual, Volume 1*. This flag can be set only if CR0.WP is set, and it must be clear before CR0.WP can be cleared (see below).

CR4.PKE

Enable protection keys for user-mode pages (bit 22 of CR4) — 4-level paging and 5-level paging associate each user-mode linear address with a protection key. When set, this flag indicates (via CPUID.07H:ECX.OSPKE[4]) that the operating system supports use of the PKRU register to specify, for each protection key, whether user-mode linear addresses with that protection key can be read or written. This bit also enables access to the PKRU register using the RDPKRU and WRPKRU instructions.

CR4.SMAP

SMAP-Enable Bit (bit 21 of CR4) — Enables supervisor-mode access prevention (SMAP) when set. See Section 5.6, “Access Rights.”

CR4.SMEP

SMEP-Enable Bit (bit 20 of CR4) — Enables supervisor-mode execution prevention (SMEP) when set. See Section 5.6, “Access Rights.”

CR4.KL

Key-Locker-Enable Bit (bit 19 of CR4) — When set, the `LOADIWKEY` instruction is enabled; in addition, if support for the AES Key Locker instructions has been activated by system firmware, CPUID.19H:EBX.AESKLE[0] is enumerated as 1 and the AES Key Locker instructions are enabled.¹ When clear, CPUID.19H:EBX.AESKLE[0] is enumerated as 0 and execution of any Key Locker instruction causes an invalid-opcode exception (#UD).

CR4.OSXSAVE

XSAVE and Processor Extended States-Enable Bit (bit 18 of CR4) — When set, this flag: (1) indicates (via CPUID.01H:ECX.OSXSAVE[27]) that the operating system supports the use of the `XGETBV`, `XSAVE`, and `XRSTOR` instructions by general software; (2) enables the `XSAVE` and `XRSTOR` instructions to save and restore the x87 FPU state (including MMX registers), the SSE state (XMM registers and MXCSR), along with other processor extended states enabled in XCR0; (3) enables the processor to execute `XGETBV` and `XSETBV` instructions in order to read and write XCR0. See Section 2.6 and Chapter 16, “System Programming for Instruction Set Extensions and Processor Extended States.”

CR4.PCIDE

PCID-Enable Bit (bit 17 of CR4) — Enables process-context identifiers (PCIDs) when set. See Section 5.10.1, “Process-Context Identifiers (PCIDs).” Applies only in IA-32e mode (if IA32_EFER.LMA = 1).

CR4.FSGSBASE

FSGSBASE-Enable Bit (bit 16 of CR4) — Enables the instructions `RDFSBASE`, `RDGSBASE`, `WRFSBASE`, and `WRGSBASE`.

CR4.SMXE

SMX-Enable Bit (bit 14 of CR4) — Enables SMX operation when set. See Chapter 7, “Safer Mode Extensions Reference,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

See also: Chapter 5, “Paging.”

1. Software can check CPUID.19H:EBX.AESKLE[0] after setting CR4.KL to determine whether the AES Key Locker instructions have been enabled. Note that some processors may allow enabling of those instructions without activation by system firmware. Some processors may not support use of the AES Key Locker instructions in system-management mode (SMM). Those processors enumerate CPUID.19H:EBX.AESKLE[0] as 0 in SMM regardless of the setting of CR4.KL.

CR4.VMXE

VMX-Enable Bit (bit 13 of CR4) — Enables VMX operation when set. See Chapter 26, “Introduction to Virtual Machine Extensions.”

CR4.LA57

57-bit linear addresses (bit 12 of CR4) — When set in IA-32e mode, the processor uses 5-level paging to translate 57-bit linear addresses. When clear in IA-32e mode, the processor uses 4-level paging to translate 48-bit linear addresses. This bit cannot be modified in IA-32e mode.

CR4.UIMP

User-Mode Instruction Prevention (bit 11 of CR4) — When set, the following instructions cannot be executed if CPL > 0: SGDT, SIDT, SLDT, SMSW, and STR. An attempt at such execution causes a general-protection exception (#GP).

CR4.OSXMMEXCPT

Operating System Support for Unmasked SIMD Floating-Point Exceptions (bit 10 of CR4) — When set, indicates that the operating system supports the handling of unmasked SIMD floating-point exceptions through an exception handler that is invoked when a SIMD floating-point exception (#XM) is generated. SIMD floating-point exceptions are only generated by SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instructions.

The operating system or executive must explicitly set this flag. If this flag is not set, the processor will generate an invalid opcode exception (#UD) whenever it detects an unmasked SIMD floating-point exception.

CR4.OSFXSR

Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4) — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX registers, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3 instruction, with the exception of PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

NOTE

CPUID feature flag FXSR indicates availability of the FXSAVE/FXRSTOR instructions. The OSFXSR bit provides operating system software with a means of enabling FXSAVE/FXRSTOR to save/restore the contents of the X87 FPU, XMM, and MXCSR registers. Consequently OSFXSR bit indicates that the operating system provides context switch support for SSE/SSE2/SSE3/SSSE3/SSE4.

CR4.PCE

Performance-Monitoring Counter Enable (bit 8 of CR4) — Enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear.

CR4.PGE

Page Global Enable (bit 7 of CR4) — (Introduced in the P6 family processors.) Enables the global page feature when set; disables the global page feature when clear. The global page feature allows frequently used or shared pages to be marked as global to all users (done with the global flag, bit 8, in a page-directory-pointer-table entry, a page-directory entry, or a page-table entry). Global pages are not flushed from the translation-lookaside buffer (TLB) on a task switch or a write to register CR3.

When enabling the global page feature, paging must be enabled (by setting the PG flag in control register CR0) before the PGE flag is set. Reversing this sequence may affect program correctness, and processor performance will be impacted.

See also: Section 5.10, “Caching Translation Information.”

CR4.MCE

Machine-Check Enable (bit 6 of CR4) — Enables the machine-check exception when set; disables the machine-check exception when clear.

See also: Chapter 18, “Machine-Check Architecture.”

CR4.PAE

Physical Address Extension (bit 5 of CR4) — When set, enables paging to produce physical addresses with more than 32 bits. When clear, restricts physical addresses to 32 bits. PAE must be set before entering IA-32e mode.

See also: Chapter 5, “Paging.”

CR4.PSE

Page Size Extensions (bit 4 of CR4) — Enables 4-MByte pages with 32-bit paging when set; restricts 32-bit paging to pages of 4 KBytes when clear.

See also: Section 5.3, “32-Bit Paging.”

CR4.DE

Debugging Extensions (bit 3 of CR4) — References to debug registers DR4 and DR5 cause an undefined opcode (#UD) exception to be generated when set; when clear, processor aliases references to registers DR4 and DR5 for compatibility with software written to run on earlier IA-32 processors.

See also: Section 20.2.2, “Debug Registers DR4 and DR5.”

CR4.TSD

Time Stamp Disable (bit 2 of CR4) — Restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear. This bit also applies to the RDTSCP instruction if supported (if CPUID.80000001H:EDX[27] = 1).

CR4.PVI

Protected-Mode Virtual Interrupts (bit 1 of CR4) — Enables hardware support for a virtual interrupt flag (VIF) in protected mode when set; disables the VIF flag in protected mode when clear.

See also: Section 23.4, “Protected-Mode Virtual Interrupts.”

2.5.1 CPUID Qualification of Control Register Flags

Not all flags in control register CR4 are implemented on all processors. With the exception of the PCE flag, they can be qualified with the CPUID instruction to determine if they are implemented on the processor before they are used.

The CR8 register is available on processors that support Intel 64 architecture.

2.6 EXTENDED CONTROL REGISTERS (INCLUDING XCR0)

If CPUID.01H:ECX.XSAVE[26] is 1, the processor supports one or more **extended control registers** (XCRs). Currently, the only such register defined is XCR0. This register specifies the set of processor state components for which the operating system provides context management, e.g., x87 FPU state, SSE state, AVX state. The OS programs XCR0 to reflect the features for which it provides context management.

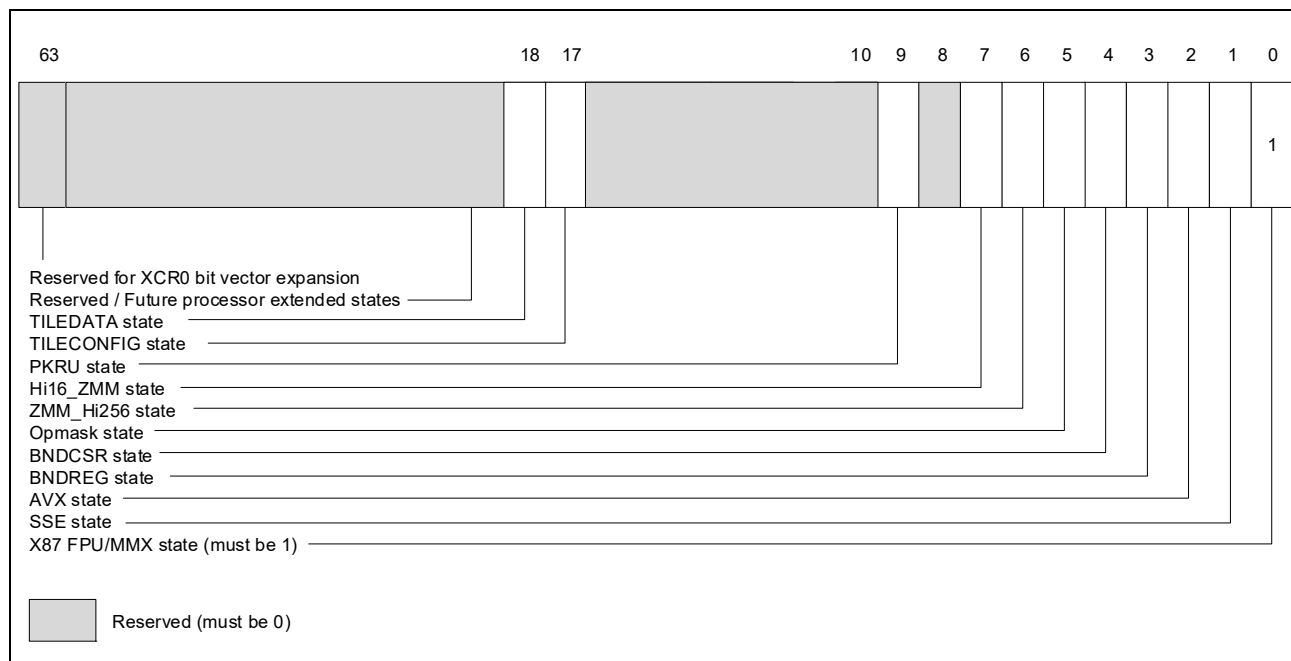


Figure 2-8. XCR0

Software can access XCR0 only if CR4.OSXSAVE[bit 18] = 1. (This bit is also readable as CPUID.01H:ECX.OSXSAVE[27].) Software can use CPUID.0DH to enumerate the bits in XCR0 that the processor supports (see CPUID instruction in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). Each supported state component is represented by a bit in XCR0. System software enables state components by loading an appropriate bit mask value into XCR0 using the XSETBV instruction.

As each bit in XCR0 (except bit 63) corresponds to a processor state component, XCR0 thus provides support for up to 63 sets of processor state components. Bit 63 of XCR0 is reserved for future expansion and will not represent a processor state component.

Currently, XCR0 defines support for the following state components:

- XCR0.X87 (bit 0): This bit 0 must be 1. An attempt to write 0 to this bit causes a #GP exception.
- XCR0.SSE (bit 1): If 1, the XSAVE feature set can be used to manage MXCSR and the XMM registers (XMM0-XMM15 in 64-bit mode; otherwise XMM0-XMM7).
- XCR0.AVX (bit 2): If 1, Intel AVX instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the YMM registers (YMM0-YMM15 in 64-bit mode; otherwise YMM0-YMM7).
- XCR0.BNDREG (bit 3): If 1, Intel MPX instructions can be executed and the XSAVE feature set can be used to manage the bounds registers BND0-BND3.
- XCR0.BNDCSR (bit 4): If 1, Intel MPX instructions can be executed and the XSAVE feature set can be used to manage the BNDCFGU and BNDSTATUS registers.
- XCR0.opmask (bit 5): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the opmask registers k0-k7.
- XCR0.ZMM_Hi256 (bit 6): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the lower ZMM registers (ZMM0-ZMM15 in 64-bit mode; otherwise ZMM0-ZMM7).
- XCR0.Hi16_ZMM (bit 7): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper ZMM registers (ZMM16-ZMM31, only in 64-bit mode).
- XCR0.PKRU (bit 9): If 1, the XSAVE feature set can be used to manage the PKRU register (see Section 2.7).
- XCR0.TILECFG (bit 17): If 1, and if XCR0.TILEDATA is also 1, Intel AMX instructions can be executed and the XSAVE feature set can be used to manage TILECFG.

- XCR0.TILEDATA (bit 18): If 1, and if XCR0.TILECFG is also 1, Intel AMX instructions can be executed and the XSAVE feature set can be used to manage TILEDATA.

An attempt to use XSETBV to write to XCR0 results in general-protection exceptions (#GP) if it would do any of the following:

- Set a bit reserved in XCR0 for a given processor (as determined by the contents of EAX and EDX after executing CPUID.0DH.00H).
- Clear XCR0.x87.
- Clear XCR0.SSE and set XCR0.AVX.
- Clear XCR0.AVX and set any of XCR0.opmask, XCR0.ZMM_Hi256, or XCR0.Hi16_ZMM.
- Set either XCR0.BNDREG or XCR0.BNDCSR while not setting the other.
- Set any of XCR0.opmask, XCR0.ZMM_Hi256, and XCR0.Hi16_ZMM while not setting all of them.
- Set either XCR0.TILECFG or XCR0.TILEDATA while not setting the other.

After reset, all bits (except bit 0) in XCR0 are cleared to zero; XCR0[0] is set to 1.

2.7 PROTECTION-KEY RIGHTS REGISTERS (PKRU AND IA32_PKRS)

Processors may support either or both of two protection-key rights registers: PKRU for user-mode pages and the IA32_PKRS MSR (MSR index 6E1H) for supervisor-mode pages. 4-level paging and 5-level paging associate a 4-bit **protection key** with each page. The protection-key rights registers determine accessibility based on a page's protection key.

If CPUID.07H.00H:ECX.PKU[3] = 1, the processor supports the protection-key feature for user-mode pages. When CR4.PKE = 1, software can use the **protection-key rights register for user pages** (PKRU) to specify the access rights for user-mode pages for each protection key.

If CPUID.07H.00H:ECX.PKS[31] = 1, the processor supports the protection-key feature for supervisor-mode pages. When CR4.PKS = 1, software can use the **protection-key rights register for supervisor pages** (the IA32_PKRS MSR) to specify the access rights for supervisor-mode pages for each protection key.

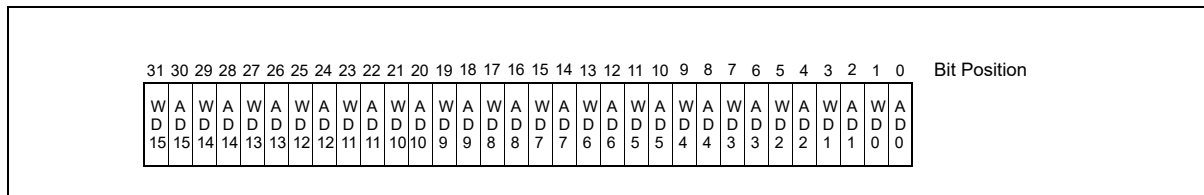


Figure 2-9. Format of Protection-Key Rights Registers

The format of each protection-key rights register is given in Figure 2-9. Each contains 16 pairs of disable controls to prevent data accesses to linear addresses (user-mode or supervisor-mode, depending on the register) based on their protection keys. Each protection key i ($0 \leq i \leq 15$) is associated with two bits in each protection-key rights register:

- Bit $2i$, shown as "AD i " (access disable): if set, the processor prevents any data accesses to linear addresses (user-mode or supervisor-mode, depending on the register) with protection key i .
- Bit $2i+1$, shown as "WD i " (write disable): if set, the processor prevents write accesses to linear addresses (user-mode or supervisor-mode, depending on the register) with protection key i .

(Bits 63:32 of the IA32_PKRS MSR are reserved and must be zero.)

See Section 5.6.2, "Protection Keys," for details of how the processor uses the protection-key rights registers to control accesses to linear addresses.

Software can read and write PKRU using the RDPKRU and WRPKRU instructions. The IA32_PKRS MSR can be read and written with the RDMSR and WRMSR instructions. Writes to the IA32_PKRS MSR using WRMSR are not serializing.

2.8 SYSTEM INSTRUCTION SUMMARY

System instructions handle system-level functions such as loading system registers, managing the cache, managing interrupts, or setting up the debug registers. Many of these instructions can be executed only by operating-system or executive procedures (that is, procedures running at privilege level 0). Others can be executed at any privilege level and are thus available to application programs.

Table 2-3 lists the system instructions and indicates whether they are available and useful for application programs. These instructions are described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D.

Table 2-3. Summary of System Instructions

Instruction	Description	Useful to Application?	Protected from Application?
LLDT	Load LDT Register	No	Yes
SLDT	Store LDT Register	No	If CR4.UMIP = 1
LGDT	Load GDT Register	No	Yes
SGDT	Store GDT Register	No	If CR4.UMIP = 1
LTR	Load Task Register	No	Yes
STR	Store Task Register	No	If CR4.UMIP = 1
LIDT	Load IDT Register	No	Yes
SIDT	Store IDT Register	No	If CR4.UMIP = 1
MOV CR _n	Load and store control registers	No	Yes
SMSW	Store MSW	Yes	If CR4.UMIP = 1
LMSW	Load MSW	No	Yes
CLTS	Clear TS flag in CRO	No	Yes
ARPL	Adjust RPL	Yes ^{1, 2}	No
LAR	Load Access Rights	Yes	No
LSL	Load Segment Limit	Yes	No
VERR	Verify for Reading	Yes	No
VERW	Verify for Writing	Yes	No
MOV DR _n	Load and store debug registers	No	Yes
INVD	Invalidate cache, no writeback	No	Yes
WBINVD	Invalidate cache, with writeback	No	Yes
WBNOINVD	Writeback cache, without invalidation	No	Yes
INVLPG	Invalidate TLB entry	No	Yes
INVPCID	Invalidate TLB entries for PCID	No	Yes
HLT	Halt Processor	No	Yes
LOCK (Prefix)	Bus Lock	Yes	No
RSM	Return from system management mode	No	Yes
RDMSR ³ , RDMSRLIST	Read Model-Specific Registers	No	Yes
WRMSR ³ , WRMSRLIST, WRMSRNS	Write Model-Specific Registers	No	Yes
RDPMS ⁴	Read Performance-Monitoring Counter	Yes	Yes ⁵
RDTSC ³	Read Time-Stamp Counter	Yes	Yes ⁵

Table 2-3. Summary of System Instructions (Contd.)

Instruction	Description	Useful to Application?	Protected from Application?
RDTSCP ⁶	Read Serialized Time-Stamp Counter	Yes	Yes ⁵
RDPID	Read Processor ID	Yes	No
RDPKRU	Read Protection Key Rights for User Pages	Yes	No
WRPKRU	Write Protection Key Rights for User Pages	Yes	No
SWAPGS	Swap GS Base Register	No	Yes
XGETBV	Return the state of XCRO	Yes	No
XSETBV	Enable one or more processor extended states	No	Yes
LKGS	Load Kernel GS Base	No	Yes
INT	Call to Interrupt Procedure	Yes	No ⁷
INT1	Generate debug trap	Yes	No
INT3	Generate breakpoint trap	Yes	No ⁷
INTO	Generate overflow trap if overflow flag is 1	Yes	No ⁷
IRET	Interrupt Return	Yes	No
SYSENTER, SYSCALL	Fast System Call	Yes	No
SYSEXIT, SYSRET	Return from Fast System Call	No	Yes
ERETS	Event Return to Supervisor	No	Yes
ERETU	Event Return to User	No	Yes

NOTES:

1. Useful to application programs running at a CPL of 1 or 2.
2. This instruction is not supported in 64-bit mode.
3. These instructions were introduced into the IA-32 Architecture with the Pentium processor.
4. This instruction was introduced into the IA-32 Architecture with the Pentium Pro processor and the Pentium processor with MMX technology.
5. The TSD and PCE flags in control register CR4 control access to these instructions by application programs running at a CPL of 3.
6. RDTSCP is introduced in Intel Core i7 processor.
7. With IDT event delivery, an operating system can configure the DPL in the relevant descriptor in the IDT to prevent application access.

2.8.1 Loading and Storing System Registers

The GDTR, LDTR, IDTR, and TR registers each have a load and store instruction for loading data into and storing data from the register:

- **LGDT (Load GDTR Register)** — Loads the GDT base address and limit from memory into the GDTR register.
- **SGDT (Store GDTR Register)** — Stores the GDT base address and limit from the GDTR register into memory.
- **LIDT (Load IDTR Register)** — Loads the IDT base address and limit from memory into the IDTR register.
- **SIDT (Store IDTR Register)** — Stores the IDT base address and limit from the IDTR register into memory.
- **LLDT (Load LDTR Register)** — Loads the LDT segment selector and segment descriptor from memory into the LDTR. (The segment selector operand can also be located in a general-purpose register.)
- **SLDT (Store LDTR Register)** — Stores the LDT segment selector from the LDTR register into memory or a general-purpose register.
- **LTR (Load Task Register)** — Loads segment selector and segment descriptor for a TSS from memory into the task register. (The segment selector operand can also be located in a general-purpose register.)

- **STR (Store Task Register)** — Stores the segment selector for the current task TSS from the task register into memory or a general-purpose register.

The LMSW (load machine status word) and SMSW (store machine status word) instructions operate on bits 0 through 15 of control register CR0. These instructions are provided for compatibility with the 16-bit Intel 286 processor. Programs written to run on 32-bit IA-32 processors should not use these instructions. Instead, they should access the control register CR0 using the MOV CR instruction.

The CLTS (clear TS flag in CR0) instruction is provided for use in handling a device-not-available exception (#NM) that occurs when the processor attempts to execute a floating-point instruction when the TS flag is set. This instruction allows the TS flag to be cleared after the x87 FPU context has been saved, preventing further #NM exceptions. See Section 2.5, “Control Registers,” for more information on the TS flag.

The control registers (CR0, CR1, CR2, CR3, CR4, and CR8) are loaded using the MOV instruction. The instruction loads a control register from a general-purpose register or stores the content of a control register in a general-purpose register.

2.8.2 Modifying Kernel GS

The Swap GS Base Register (SWAPGS) instruction exchanges the current GS base register value with the IA32_KERNEL_GS_BASE MSR (address C0000102H). The instruction cannot be executed when FRED transitions are enabled. FRED transitions perform the same swap when changing the CPL.

The LKGS (Load Kernel GS Base) instruction operates in the same way as MOV to GS except that the descriptor's base address is loaded into the IA32_KERNEL_GS_BASE MSR instead of the GS segment's descriptor cache. LKGS was introduced alongside FRED to allow system software to manage the GS segment more flexibly. Support is indicated by CPUID.07H.01H:EAX.LKGS[18] = 1.

2.8.3 Reading and Writing Protection Key Rights for User Pages

The Protection Key Rights User register (PKRU) can be read and written using the RDPKRU and WRPKRU instructions, respectively. Protection Keys User (PKU) feature support is indicated by CPUID.07H.00H:ECX.PKU[3] = 1. The OS must enable the PKU feature through CR4.PKE[22] = 1. Instruction support is indicated to software by CPUID.07H.00H:ECX.OSPKE[4] = 1. The 32-bit PKRU is read to or written from the EAX register.

See Section 2.7, “Protection-Key Rights Registers (PKRU and IA32_PKRS),” and Section 5.6.2, “Protection Keys,” for more information on the PKRU register and Protection Keys feature.

2.8.4 Verifying of Access Privileges

The processor provides several instructions for examining segment selectors and segment descriptors to determine if access to their associated segments is allowed. These instructions duplicate some of the automatic access rights and type checking done by the processor, thus allowing operating-system or executive software to prevent exceptions from being generated.

The ARPL (adjust RPL) instruction adjusts the RPL (requestor privilege level) of a segment selector to match that of the program or procedure that supplied the segment selector. See Section 6.10.4, “Checking Caller Access Privileges (ARPL Instruction),” for a detailed explanation of the function and use of this instruction. Note that ARPL is not supported in 64-bit mode.

The LAR (load access rights) instruction verifies the accessibility of a specified segment and loads access rights information from the segment's segment descriptor into a general-purpose register. Software can then examine the access rights to determine if the segment type is compatible with its intended use. See Section 6.10.1, “Checking Access Rights (LAR Instruction),” for a detailed explanation of the function and use of this instruction.

The LSL (load segment limit) instruction verifies the accessibility of a specified segment and loads the segment limit from the segment's segment descriptor into a general-purpose register. Software can then compare the segment limit with an offset into the segment to determine whether the offset lies within the segment. See Section 6.10.3, “Checking That the Pointer Offset Is Within Limits (LSL Instruction),” for a detailed explanation of the function and use of this instruction.

The VERR (verify for reading) and VERW (verify for writing) instructions verify if a selected segment is readable or writable, respectively, at a given CPL. See Section 6.10.2, “Checking Read/Write Rights (VERR and VERW Instructions),” for a detailed explanation of the function and use of these instructions.

2.8.5 Loading and Storing Debug Registers

Internal debugging facilities in the processor are controlled by a set of 8 debug registers (DR0-DR7). The MOV instruction allows setup data to be loaded to and stored from these registers.

On processors that support Intel 64 architecture, debug registers DR0-DR7 are 64 bits. In 32-bit modes and compatibility mode, writes to a debug register fill the upper 32 bits with zeros. Reads return the lower 32 bits. In 64-bit mode, the upper 32 bits of DR6-DR7 are reserved and must be written with zeros. Writing one to any of the upper 32 bits causes an exception, #GP(0).

In 64-bit mode, MOV DRn instructions read or write all 64 bits of a debug register (operand-size prefixes are ignored). All 64 bits of DR0-DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0-DR3 are in the limits of the implementation. Address matching is supported only on valid addresses generated by the processor implementation.

2.8.6 Invalidating Caches and TLBs

The processor provides several instructions for use in explicitly invalidating its caches and TLB entries. The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and sends a signal to the external caches indicating that they should also be invalidated.

The WBINVD (invalidate cache with writeback) instruction performs the same function as the INVD instruction, except that it writes back modified lines in its internal caches to memory before it invalidates the caches. After invalidating the caches local to the executing logical processor or processor core, WBINVD signals caches higher in the cache hierarchy (caches shared with the invalidating logical processor or core) to write back any data they have in modified state at the time of instruction execution and to invalidate their contents.

The WBNOINVD (writeback cache without invalidation) instruction performs similarly to the WBINVD instruction but does not invalidate (flush) the internal caches. Support is indicated by CPUID.80000008H:EBX.WBNOINVD[9] = 1.

Note, non-shared caches may not be written back nor invalidated. In Figure 2-10 below, if code executing on either LP0 or LP1 were to execute a WBINVD, the shared L1 and L2 for LP0/LP1 will be written back and invalidated as will the shared L3. However, the L1 and L2 caches not shared with LP0 and LP1 will not be written back nor invalidated.

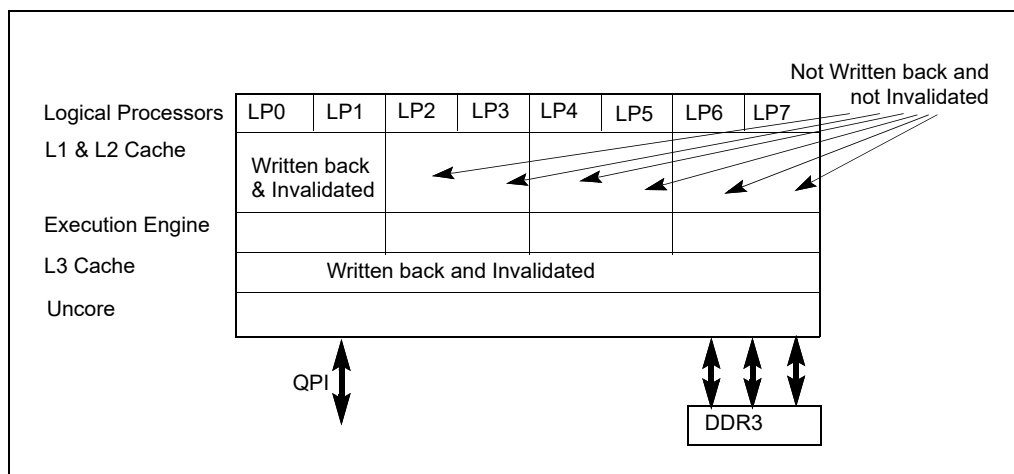


Figure 2-10. WBINVD Invalidation of Shared and Non-Shared Cache Hierarchy

The INVLPG (invalidate TLB entry) instruction invalidates (flushes) the TLB entry for a specified page.

The **INVPID** (Invalidate Process-Context Identifier) instruction invalidates (flushes) the TLB entries for the specified PCID. Support is indicated by `CPUID.07H.00H:EBX.INVPID[10] = 1`.

2.8.7 Controlling the Processor

The **HLT** (halt processor) instruction stops the processor until an enabled interrupt (such as NMI or SMI, which are normally enabled), a debug exception, the **BINIT#** signal, the **INIT#** signal, or the **RESET#** signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered.

Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked (note that the **BINIT#** pin was introduced with the Pentium Pro processor). If any non-wake events are pending during shutdown, they will be handled after the wake event from shutdown is processed (for example, **A20M#** interrupts).

The **LOCK** prefix invokes a locked (atomic) read-modify-write operation when modifying a memory operand. This mechanism is used to allow reliable communications between processors in multiprocessor systems, as described below:

- In the Pentium processor and earlier IA-32 processors, the **LOCK** prefix causes the processor to assert the **LOCK#** signal during the instruction. This always causes an explicit bus lock to occur.
- In the Pentium 4, Intel Xeon, and P6 family processors, the locking operation is handled with either a cache lock or bus lock. If a memory access is cacheable and affects only a single cache line, a cache lock is invoked and the system bus and the actual memory location in system memory are not locked during the operation. Here, other Pentium 4, Intel Xeon, or P6 family processors on the bus write-back any modified data and invalidate their caches as necessary to maintain system memory coherency. If the memory access is not cacheable and/or it crosses a cache line boundary, the processor's **LOCK#** signal is asserted and the processor does not respond to requests for bus control during the locked operation.

The **RSM** (return from SMM) instruction restores the processor (from a context dump) to the state it was in prior to a system management mode (SMM) interrupt.

2.8.8 Reading Performance-Monitoring Counters

The **RDPMC** (read performance-monitoring counter) instruction allows application programs to read the processor's performance-monitoring counters. Processors based on Intel NetBurst® microarchitecture have eighteen 40-bit performance-monitoring counters; P6 family processors have two 40-bit counters. Intel Atom® processors and most of the processors based on the Intel Core microarchitecture support two types of performance monitoring counters: programmable performance counters similar to those available in the P6 family, and three fixed-function performance monitoring counters. Details of programmable and fixed-function performance monitoring counters for each processor generation are described in Chapter 21, "Last Branch Records."

The programmable performance counters can support counting either the occurrence or duration of events. Events that can be monitored on programmable counters generally are model specific (except for architectural performance events enumerated by `CPUID.0AH`); they may include the number of instructions decoded, interrupts received, or the number of cache loads. Individual counters can be set up to monitor different events. Use the system instruction **WRMSR** to set up values in one of the `IA32_PERFECTSELx` MSR, in one of the 45 `ESCRs` and one of the 18 `CCCR` MSRs (for Pentium 4 and Intel Xeon processors); or in the `PerfEvtSel0` or the `PerfEvtSel1` MSR (for the P6 family processors). The **RDPMC** instruction loads the current count from the selected counter into the `EDX:EAX` registers.

Fixed-function performance counters record only specific events that are defined at: <https://perfmon-events.intel.com/>, and the width/number of fixed-function counters are enumerated by `CPUID.0AH`.

See Section 22.1, "Performance Monitoring Overview," for more information about the performance monitoring counters.

The **RDPMC** instruction was introduced into the IA-32 architecture with the Pentium Pro processor and the Pentium processor with MMX technology. Earlier Pentium processors have two performance-monitoring counters, but they can be read only with the **RDMR** instruction, and only at privilege level 0.

RDPMC requires an index to specify the offset of the performance-monitoring counter. In 64-bit mode for Pentium 4 or Intel Xeon processor families, the index is specified in ECX[30:0]. The current count of the performance-monitoring counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

2.8.9 Reading Time-Stamp Counter and IA32_TSC_AUX

The RDTSC/RDTSCP (read Time-Stamp Counter, Processor ID) instructions allow application programs to read the processor's time-stamp counter. The RDTSC instruction was introduced into the IA-32 architecture with the Pentium processor. Support is indicated by CPUID.01H:EDX.TSC[4] = 1. In 64-bit mode, RDTSC/RDTSCP operates the same as in protected mode.

The RDTSCP instruction operates similarly to the RDTSC instruction, but additionally returns the IA32_TSC_AUX MSR in ECX. RDTSCP waits until all previous instructions have executed and all previous loads are globally visible. Support is indicated by CPUID.80000001H:EDX.RDTSCP[27] = 1.

The RDPID (Read Processor ID) instruction simply reads the IA32_TSC_AUX MSR into the specified GPR. Support is indicated by CPUID.07H.00H:ECX.RDPID[22] = 1.

The time-stamp counter is a model-specific 64-bit counter that is reset to zero each time the processor is reset. If not reset, the counter will increment $\sim 9.5 \times 10^{16}$ times per year when the processor is operating at a clock rate of 3 GHz. At this clock frequency, it would take over 190 years for the counter to wrap around. The RDTSC instruction loads the current count of the time-stamp counter into the EDX:EAX registers.

See Section 20.17, "Time-Stamp Counter," for more information about the time-stamp counter.

2.8.10 Reading and Writing Model-Specific Registers

The RDMSR (read model-specific register) and WRMSR (write model-specific register) instructions allow a processor's 64-bit model-specific registers (MSRs) to be read and written, respectively. RDMSR and WRMSR were introduced into the IA-32 architecture with the Pentium processor. In 64-bit mode, the index specifying the MSR remains 32 bits.

WRMSRNS behaves like WRMSR, except that it is not a serializing instruction by default. Support is indicated by CPUID.07H.01H:EAX.WRMSRNS[19] = 1.

RDMSRLIST and WRMSRLIST allow up to 64 MSRs from a software-provided list to be read to or written from memory. WRMSRLIST is not a serializing instruction by default. Support is indicated by CPUID.07H.01H:EAX.MSRLIST[27] = 1.

The initial forms of RDMSR, WRMSR, and WRMSRNS use implicit 32-bit registers EDX:EAX for the value and ECX for the index.

RDMSRLIST and WRMSRLIST use implicit 64-bit registers. RSI contains the linear address of the memory table of MSR indexes, RDI contains the linear address of the memory table of MSR values. RCX is the bit mask of table entries.

See Section 12.4, "Model-Specific Registers (MSRs)," and the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B and Volume 2D for more information.

2.8.11 Enabling Processor Extended States

The XSETBV instruction is required to enable OS support of individual processor extended states in XCR0 (see Section 2.6).

5. Updates to Chapter 4, Volume 3A

Change bars and violet 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:

- Section 4.4, "Linear-Address Masking," was updated for the revised architecture where LAM applies to all 64-bit linear data accesses (not just those in 64-bit mode).

As described in Chapter 3, “Protected-Mode Memory Management,” software accesses to memory typically use logical addresses. The processor uses segmentation, as detailed in Section 3.4, to generate linear addresses from logical addresses. Linear addresses are then translated to physical addresses using paging, as described in Chapter 5, “Paging.”

In IA-32e mode (if `IA32_EFER.LMA = 1`), linear addresses may undergo some pre-processing before being translated through paging.¹ Some of this pre-processing is done only if enabled by software, but some occurs unconditionally. Specifically, linear addresses are subject to pre-processing in IA-32e mode as follows:

1. **Linear-address-space separation (LASS).** This is a feature that, when enabled by software, may limit the linear addresses that are accessible by software, generating faults for accesses out of range.
2. **Linear-address masking (LAM).** This is a feature that, when enabled by software, masks certain linear-address bits.
3. **Canonicity checking.** As will be detailed in Chapter 5, paging does not translate all 64 bits of a linear address. Each linear address must be **canonical**, meaning that the untranslated bits have a fixed value. Memory accesses using a non-canonical address generate faults.

Both LASS and canonicity checking can generate faults. For any specific memory access, the two features generate the same fault. For that reason, the relative order of that checking is not defined and cannot be determined by software.

4.1 ENABLING AND ENUMERATION

Software enables LASS by setting `CR4.LASS[bit 27]`. Enabling of LAM is based on three different bits: `CR3.LAM_U48[bit 62]`, `CR3.LAM_U57[bit 61]`, and `CR4.LAM_SUP[bit 28]`. The use of these bits is explained in Section 4.4. Canonicity checking is not enabled by software and is always performed in 64-bit mode.

The processor enumerates support for LASS with `CPUID.07H.01H:EAX.LASS[6]`. If this bit is enumerated as 1, software can set `CR4.LASS`.

The processor enumerates support for LAM with `CPUID.07H.01H:EAX.LAM[26]`. If this bit is enumerated as 1, software can set `CR3.LAM_U48`, `CR3.LAM_U57`, and `CR4.LAM_SUP`.

4.2 MODE-BASED ACCESSES AND LINEAR-ADDRESS-SPACE PARTITIONING

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. 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; accesses to the task-state segment (TSS) as part of a task switch or change of CPL; and accesses to a user posted-interrupt descriptor (UPID) during user-interrupt notification processing. Such accesses are called **implicit supervisor-mode accesses** regardless of CPL. Other accesses made while `CPL < 3` are called **explicit supervisor-mode accesses**.²

-
1. The presentation in this chapter focuses on 64-bit addresses. 32-bit and 16-bit addresses can also be used in IA-32e mode. For the purposes of this chapter, the upper bits of such addresses (32 bits and 48 bits, respectively) are treated as if they were all zero.
 2. The `WRUSS` instruction is an exception; although it can be executed only if `CPL = 0`, the processor treats its shadow-stack accesses as user-mode accesses.

Some 64-bit operating systems partition the 64-bit linear-address space into a supervisor portion and a user portion. Specifically, the upper half of the linear-address space (comprising addresses in which bit 63 is 1) is used for supervisor instructions and data, while the lower half (addresses in which bit 63 is 0) is for user instructions and data.

The LASS and LAM features are designed for operating systems that establish such linear-address-space partitioning. However, the features are defined and may be used even if such partitioning is not in effect.

4.3 LINEAR-ADDRESS-SPACE SEPARATION (LASS)

The access rights determined by paging (see Section 5.6) are based on whether a linear address is a supervisor-mode address or a user-mode address. Paging provides protection by preventing user-mode accesses to supervisor-mode addresses; in addition, there are paging features that can prevent supervisor-mode accesses to user-mode addresses.

These paging-based protections prevent malicious software from directly reading or writing memory inappropriately. However, they require the processor to traverse a hierarchy of paging structures in memory. Unprivileged software may be able to use the timing information resulting from this traversal to determine details about the paging structures, the layout of supervisor memory, or its use by supervisor software.

Linear-address-space separation (LASS) is an independent mechanism that can enforce mode-based protection without traversing the paging structures. Because LASS provides this protection as part of linear-address pre-processing, unprivileged software is denied paging-based timing information.

An operating system can use LASS to provide protections corresponding to the mode-based paging protections if it has established the linear-address-space partitioning outlined in Section 4.2.

4.3.1 Enumeration and Enabling

The processor enumerates support for LASS with CPUID.07H.01H:EAX.LASS[6].

Software enables LASS by setting CR4.LASS[bit 27]. CR4.LASS can be set to 1 if CPUID.07H.01H:EAX.LASS[6] is enumerated as 1.

The operation of LASS is also affected by the paging-mode bit CR4.SMAP[bit 21], which enables supervisor-access prevention. LASS enforces the equivalent of supervisor-mode execution prevention regardless of the setting of CR4.SMEP[bit 17]. See Section 4.3.2 for details.

4.3.2 Operation of Linear-Address-Space Separation (LASS)

This section describes the operation of linear-address-space separation (LASS). This operation applies only in IA-32e mode (if IA32_EFER.LMA = 1) and only if CR4.LASS = 1.

As indicated earlier, LASS enforces protections similar to those enforced by paging. Violations of these protections are called **LASS violations**.

LASS violations typically result in faults. In most cases, an access causing a LASS violation results in a general protection exception (#GP); for stack accesses (those due to stack-oriented instructions, as well as accesses that implicitly or explicitly use the SS segment register), a stack fault (#SS) is generated. In either case, a null error code is produced.

Some accesses are not subject to faults due to LASS violations. These include prefetches (e.g., those resulting from execution of one of the PREFETCHh instructions), executions of the CLDEMOT instruction, and accesses resulting from the speculative fetch or execution of an instruction. Such an access may cause a LASS violation; if it does, the access is not performed but no fault occurs.

The remainder of this section describes how LASS applies to different types of accesses to linear addresses. The items below discuss specific LASS violations based on bit 63 of a linear address. For a linear address with only 32 bits (or 16 bits), the processor treats bit 63 as if it were 0; this includes accesses in compatibility mode.

- A user-mode data access causes a LASS violation if it would access a linear address of which bit 63 is 1.

- A supervisor-mode data access causes a LASS violation if it would access a linear address of which bit 63 is 0, supervisor-mode access protection is enabled (by setting CR4.SMAP), and either RFLAGS.AC = 0 or the access is an implicit supervisor-mode access.
- A user-mode instruction fetch causes a LASS violation if it would fetch an instruction using a linear address of which bit 63 is 1.
- A supervisor-mode instruction fetch causes a LASS violation if it would access a linear address of which bit 63 is 0. (Unlike paging, this behavior of LASS applies regardless of the setting of CR4.SMEP.)

LASS for instruction fetches applies when the linear address in RIP is used to load an instruction from memory. Unlike canonicity checking (see Section 4.5.2), LASS does **not** apply to branch instructions that load RIP. A branch instruction can load RIP with an address that would violate LASS. Only when the address is used to fetch an instruction will a LASS violation occur, generating a #GP. (The return instruction pointer of the #GP handler is the address that incurred the LASS violation.)

4.4 LINEAR-ADDRESS MASKING

This section describes **linear-address masking (LAM)**. LAM modifies linear addresses before they are subject to canonicity checking as described in Section 4.5. Doing so allows untranslated linear-address bits to contain arbitrary values.

In IA-32e mode, linear addresses have 64 bits and are translated either with 4-level paging, which translates the low 48 bits of each linear address, or with 5-level paging, which translates 57 bits. The upper linear-address bits are reserved by canonicity checking (see Section 4.5).

Software usages that associate metadata with a pointer might benefit from being able to place metadata in the upper (untranslated) bits of the pointer itself. However, the canonicity enforcement mentioned earlier implies that software would have to mask the metadata bits in a pointer (making it canonical) before using it as a linear address to access memory. LAM allows software to use pointers with metadata without having to mask the metadata bits. With LAM enabled, the processor masks the metadata bits in a pointer before using it as a linear address to access memory.

LAM applies only to 64-bit linear addresses used for data accesses. It does not apply to addresses used for instruction fetches or to those being loaded into the RIP register (e.g., as targets of jump and call instructions).

4.4.1 Enumeration, Enabling, and Configuration

The processor enumerates support for LAM with CPUID.07H.01H:EAX.LAM[26].

Enabling and configuration of LAM is controlled by the following control-register bits: CR3.LAM_U48[bit 62], CR3.LAM_U57[bit 61], and CR4.LAM_SUP[bit 28]. The use of these control bits is explained below.

LAM supports configurations that differ regarding which linear-address bits are masked and can be used for metadata. With **LAM48**, linear-address bits 62:48 are masked (resulting in a **LAM width** of 15); with **LAM57**, linear-address bits 62:57 are masked (a LAM width of 6).

Like LASS, LAM was designed for operating systems that establish the linear-address-space partitioning outlined in Section 4.2: linear addresses that clear bit 63 are used for user memory, while those that set bit 63 are for supervisor memory. For LAM, the identification of an address as user or supervisor is based solely on the value of bit 63 and does not depend on the CPL.

LAM and the LAM width can be configured independently for user and supervisor addresses (as identified in the previous paragraph, using bit 63). CR3.LAM_U48 and CR3.LAM_U57 enable and configure LAM for user addresses:

- If CR3.LAM_U48 = CR3.LAM_U57 = 0, LAM is not enabled for user addresses.
- If CR3.LAM_U48 = 1 and CR3.LAM_U57 = 0, LAM48 is enabled for user addresses (a LAM width of 15).
- If CR3.LAM_U57 = 1, LAM57 is enabled for user addresses (a LAM width of 6; CR3.LAM_U48 is ignored in this case).

CR4.LAM_SUP enables and configures LAM for supervisor addresses:

- If CR4.LAM_SUP = 0, LAM is not enabled for supervisor addresses.

- If CR4.LAM_SUP = 1, LAM is enabled for supervisor addresses with a width determined by the paging mode (see Section 5.1.1):
 - If 4-level paging is enabled, LAM48 is enabled for supervisor addresses (a LAM width of 15).
 - If 5-level paging is enabled, LAM57 is enabled for supervisor addresses (a LAM width of 6).

4.4.2 Treatment of Data Accesses with LAM Active

When LAM is active, linear addresses used to access data are **masked** before they are subject to the canonicity checking identified in Section 4.5. Specifically, LAM modifies a linear address by extending the value of one address bit (depending on the LAM width) over others:

- When LAM48 is enabled (see Section 4.4.1), the processor modifies each linear address to replace each of bits 62:48 with the value of bit 47.
- When LAM57 is enabled, each of bits 62:57 is replaced by the value of bit 56 (bits 56:48 are not modified).

4.4.3 Paging Interactions

As explained in Section 4.4.2, LAM masks certain bits in a linear address before that address is translated by paging.

In most cases, the address bits in the masked positions are not used by address translation. However, if 5-level paging is active and LAM48 is enabled for user pointers, bit 47 of a user pointer is extended over bits 62:48 to form a linear address, and bits 56:48 are used by 5-level paging.

Page faults report the faulting linear address in CR2. Because LAM masking applies before paging, the faulting linear address recorded in CR2 reflects the result of that masking and does not contain any masked metadata.

The INVLPG, INVPCID, and INVVPID instructions can be used to invalidate any translation lookaside buffer (TLB) entries for specified linear addresses. LAM does not apply to those addresses, although those addresses are subject to canonicity checking (see Section 4.5.4).

4.5 CANONICALITY CHECKING

Memory accesses in IA-32e mode can use 64-bit linear addresses. As detailed in Section 5.5.4, 4-level paging translates the low 48 bits of each linear address, while 5-level paging translates the low 57 bits. The remaining upper bits (bits 63:48 with 4-level paging; bits 63:57 with 5-level paging) are not translated.

IA-32e mode accounts for the fact that address bits are not translated (and thus should be reserved) with the concept of **canonicity**. In general, a linear address is **canonical** if the untranslated bits are a sign-extension of the most significant translated bit. More specifically, there are two types of canonicity:

- A linear address is **paging canonical** if it is canonical for the current paging mode: a linear address is canonical for 4-level paging (**48-bit canonical**) if bits 63:47 of the address are identical; it is canonical for 5-level paging (**57-bit canonical**) if bits 63:56 are identical.
- A linear address is **CPU canonical** if it is canonical relative to the widest linear address supported by the processor: 48-bit canonical if the processor supports only 4-level paging and 57-bit canonical if the processor supports 5-level paging.

Unlike LASS and LAM, there is no control to enable canonicity checking. It always applies (as described in this section) when 64-bit linear addresses are used.

Section 4.5.1 and Section 4.5.2 explain canonicity checking for accesses to memory using linear addresses and for loads of the instruction pointer, respectively. Section 4.5.3 details how canonicity checking applies to certain system registers that contain linear addresses. Section 4.5.4 explains the role of canonicity checking by instructions that invalidate TLB entries.

NOTE

Section 4.5.2 and Section 4.5.3 discuss the canonicity checking performed by the WRMSR instruction. The WRMSRLIST and WRMSRNS instructions perform the same canonicity checking in corresponding situations. (Similarly, the characterization of RDMSR in Section 4.5.3 applies also to RDMSRLIST.)

4.5.1 Memory Accesses

An access to memory using a linear address is allowed only if the address is paging canonical; if it is not, a **canonicity violation** occurs. In most cases, an access causing a canonicity violation results in a general protection exception (#GP); for stack accesses (those due to stack-oriented instructions, as well as accesses that implicitly or explicitly use the SS segment register), a stack fault (#SS) is generated. In either case, a null error code is produced.

When LAM is enabled, canonicity checking is performed after masking of the linear address. This implies that the requirements of canonicity on an original (unmasked) linear address used to access data are effectively relaxed when LAM is enabled:

- With LAM48, bit 63 and bit 47 of the original linear address must be identical.
- With LAM57 and 4-level paging, bit 63 and bits 56:47 of the original linear address must be identical.
- With LAM57 and 5-level paging, bit 63 and bit 56 of the original linear address must be identical.

While LAM applies only to data accesses, canonicity checking applies both data accesses and instruction fetches.

4.5.2 Loads of the Instruction Pointer (RIP)

In 64-bit mode, the RIP register contains the linear address of the instruction pointer. Operations that load RIP (including both instructions such as JMP as well as control transfers through the IDT) check first whether the value to be loaded is paging canonical. If it is not, the operation does not modify RIP and instead causes a #GP. (This #GP is delivered as a fault, meaning that the return instruction pointer of the fault handler is the address of the faulting instruction and not the non-canonical address whose load was attempted.)

This treatment applies to the SYSRET and SYSEXIT instructions, which load RIP from RCX and RDX, respectively. The SYSCALL and SYSENTER instructions load RIP from the IA32_LSTAR and IA32_SYSENTER_EIP MSRs, respectively. On processors that support only 4-level paging, these instructions do not check explicitly that the values being loaded are paging canonical. This is because the WRMSR instruction ensures that these MSRs necessarily contain values that are CPU canonical, which is the same as being paging canonical on processors that support only 4-level paging. On processors that support 5-level paging, the checking by WRMSR is relaxed to ensure only 57-bit canonicity. (See Section 4.5.3 for the treatment of WRMSR.) On such processors, an execution of SYSCALL or SYSENTER with 4-level paging checks that the value being loaded into RIP is 48-bit canonical (paging canonical).

4.5.3 System Registers Containing Linear Addresses

In addition to RIP, the CPU maintains numerous other registers that hold linear addresses:

- GDTR and IDTR (in their base-address portions).
- LDTR, TR, FS, and GS (in the base-address portions of their hidden descriptor caches).
- Control register CR2, which holds the linear address causing a page fault.
- The debug-address registers (DR0 through DR3), which hold the linear addresses of breakpoints.
- The following MSRs: IA32_BNDCFGS, IA32_DS_AREA, IA32_FS_BASE, IA32_GS_BASE, IA32_INTERRUPT_SSP_TABLE_ADDR, IA32_KERNEL_GS_BASE, IA32_LSTAR, IA32_PL0_SSP, IA32_PL1_SSP, IA32_PL2_SSP, IA32_PL3_SSP, 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, IA32_S_CET, IA32_SYSENTER_EIP, IA32_SYSENTER_ESP, IA32_UINTR_HANDLER, IA32_UINTR_PD, IA32_UINTR_STACKADJUST, IA32_U_CET, and IA32_UINTR_TT.
- The x87 FPU instruction pointer (FIP).

- The user-mode configuration register BNDCFGU, used by Intel® MPX.

With a few exceptions, the processor ensures that the addresses in these registers are always canonical in the following ways:

- Some instructions fault on attempts to load a linear-address register with a non-canonical address:
 - An execution of the LGDT or LIDT instruction causes a general-protection exception (#GP) if the base address specified in the instruction's memory operand is not canonical.
 - An execution of the LLDT or LTR instruction causes a #GP if the base address to be loaded from the GDT is not canonical.
 - An execution of WRFSBASE or WRGSBASE causes a #GP if it would load the base address of either FS or GS with a non-canonical address.
 - An execution of WRMSR causes a #GP if it would load any of the following MSRs with a non-canonical address: IA32_BNDCFGS, IA32_DS_AREA, IA32_FS_BASE, IA32_GS_BASE, IA32_INTERRUPT_SSP_TABLE_ADDR, IA32_KERNEL_GS_BASE, IA32_LSTAR, IA32_PL0_SSP, IA32_PL1_SSP, IA32_PL2_SSP, IA32_PL3_SSP, 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, IA32_S_CET, IA32_SYSENTER_EIP, IA32_SYSENTER_ESP, IA32_UINTR_HANDLER, IA32_UINTR_PD, IA32_UINTR_STACKADJUST, IA32_U_CET, and IA32_UINTR_TT.¹
 - An execution of XRSTORS causes a #GP if it would load any of the following MSRs with a non-canonical address: IA32_PL0_SSP, IA32_PL1_SSP, IA32_PL2_SSP, IA32_PL3_SSP, 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, IA32_U_CET, IA32_UINTR_HANDLER, IA32_UINTR_PD, IA32_UINTR_STACKADJUST, or IA32_UINTR_TT.

With a small number of exceptions, this enforcement checks for CPU canonicity and is thus independent of the current paging mode. Thus, a processor that supports 5-level paging will allow the instructions mentioned above to load these registers with addresses that are 57-bit canonical but not 48-bit canonical, even if 4-level paging is active. (As a result, instructions that store these values — SGDT, SIDT, SLDT, STR, RDFSBASE, RDGSBASE, RDMSR, XSAVE, XSAVEC, XSAVEOPT, and XSAVES — may save addresses that are 57-bit canonical but not 48-bit canonical, even if 4-level paging is active.)

The WRFSBASE and WRGSBASE instructions, which load the base address of FS and GS, respectively, operate differently. An execution of either of these instructions causes a #GP if it would load a base address with an address that is not paging canonical. Thus, if 4-level paging is active, these instructions do not allow loading of addresses that are 57-bit canonical but not 48-bit canonical.

- The FXRSTOR, XRSTOR, and XRSTORS instructions ignore attempts to load some of these registers with non-canonical addresses:
 - Loads of FIP ignore any bits in the memory image beyond the enumerated maximum linear-address width. The processor sign-extends the most significant bit (e.g., bit 56 on processors that support 5-level paging) to ensure that FIP is always CPU canonical.
 - Loads of BNDCFGU (by XRSTOR or XRSTORS) ignore any bits in the memory image beyond the enumerated maximum linear-address width. The processor sign-extends the most significant bit to ensure that BNDCFGU is always CPU canonical.
- Every non-control x87 instruction loads FIP. The value loaded is always paging canonical.
- CR2 can be loaded with the MOV to CR instruction. The instruction allows that register to be loaded with a non-canonical address. The MOV from CR instruction will return for CR2 the value last loaded into that register by a page fault or with the MOV to CR instruction, even if (for the latter case) the address is not canonical. Page faults load CR2 only with linear addresses that are paging canonical.
- DR0 through DR3 can be loaded with the MOV to DR instruction. The instruction allows those registers to be loaded with non-canonical addresses. The MOV from DR instruction will return for a debug register the value

1. Such canonicity checking may apply also when the WRMSR instruction is used to load some non-architectural MSRs (not listed here) that hold a linear address.

last loaded into that register with the MOV to DR instruction, even if the address is not canonical. Breakpoint address matching is supported only for linear addresses that are paging canonical.

4.5.4 TLB-Invalidation Instructions

The Intel 64 architecture includes three instructions that may invalidate TLB entries for the linear address of an instruction operand: INVLPG, INVPCID, and INVVPID. The following items describe how they are affected by canonicity.

- The INVLPG instruction takes a memory operand. It invalidates any TLB entries that the logical processor is caching for the linear address of that operand for the current linear address space. The instruction does not fault if that address is not paging canonical. However, no invalidation is performed because the processor does not cache TLB entries for addresses that are not paging canonical.
- The INVPCID instruction takes a register operand (INVPCID type) and a memory operand (INVPCID descriptor). If the INVPCID type is 0, the instruction invalidates any TLB entries that the logical processor is caching for the linear address and PCID specified in the INVPCID descriptor. If the linear address is not CPU canonical, the instruction causes a #GP. If the processor supports 5-level paging, the instruction will not cause such a #GP for an address that is 57-bit canonical, regardless of paging mode, even if 4-level paging is active and the address is not 48-bit canonical.
- The INVVPID instruction takes a register operand (INVVPID type) and a memory operand (INVVPID descriptor). If the INVPCID type is 0, the instruction invalidates any TLB entries that the logical processor is caching for the linear address and VPID specified in the INVVPID descriptor. If the linear address is not CPU canonical, the instruction fails.¹ If the processor supports 5-level paging, the instruction will not fail for an address that is 57-bit canonical, regardless of paging mode, even if 4-level paging is active and the address is not 48-bit canonical.

LAM does not apply to the linear addresses that these instructions use to invalidate TLB entries.

1. INVVPID is a VMX instruction. In response to certain conditions, execution of a VMX instruction may **fail**, meaning that it does not complete its normal operation. When a VMX instruction fails, control passes to the next instruction (rather than to a fault handler) and a flag is set to report the failure.

6. Updates to Chapter 8, Volume 3A

Change bars and violet 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:

- Section 8.3.3 was updated removing reference to FRED event delivery of a non-maskable interrupt.
- The pseudo code in Section 8.3.5 was updated removing reference to an event delivery of a non-maskable interrupt.

CHAPTER 8

FLEXIBLE RETURN AND EVENT DELIVERY (FRED)

This chapter describes the architecture of **flexible return and event delivery (FRED)**.

The FRED architecture defines new transitions for invoking and returning from event handlers. These transitions are **FRED event delivery** and, for returning from events, two **FRED return instructions**.

FRED event delivery can change the current privilege level (CPL) from 3 to 0, but it is used also to deliver events that occur while CPL = 0. One FRED instruction (ERETU) effects a return that changes the CPL from 0 to 3, while the other (ERETS) returns while leaving the CPL at 0. Collectively, FRED event delivery and the FRED return instructions are **FRED transitions**.

The following is the organization of this chapter:

- Section 8.1 gives an overview of the FRED architecture.
- Section 8.2 gives details on enumeration and configuration of the FRED architecture, including the new state defined for it.
- Section 8.3 explains FRED event delivery.
- Section 8.4 presents the FRED return instructions.
- Section 8.5 describes how enabling FRED transitions changes the operation of existing instructions (including those use for system call and return).

8.1 OVERVIEW OF THE FRED ARCHITECTURE

The following items summarize the principle elements of the FRED architecture:

- **FRED event delivery.** Any event that would normally cause IDT event delivery (e.g., an interrupt or exception) instead establishes the new context with a process that does not use existing data structures such as the IDT. The SYSCALL and SYSENTER instructions also use FRED event delivery in place of their existing operation.
- **FRED return instructions.** Two return instructions, ERETS and ERETU, effect returns from event handling. ERETS (event return to supervisor) is used to return from events incident to CPL 0 and does not change CPL; ERETU (event return to user) returns to software operating with CPL 3.
- **GS segment management.** FRED transitions that change the CPL (FRED event delivery and ERETU) update the GS base address in a manner similar to the SWAPGS instruction. In addition, the LKGS instruction allows system software to manage the GS segment more flexibly.

When FRED transitions are enabled, they are the only transitions that change the CPL. Since FRED event delivery is always to CPL 0 and ERETU returns only to CPL 3, it is not possible for CPL to be 1 or 2 while FRED transitions are enabled. Enabling FRED transitions also prevents entry to compatibility mode with CPL 0.

The FRED architecture is defined for use by a 64-bit operating system. Except where noted, FRED event delivery and the FRED instructions (ERETS, ERETU, and LKGS) apply only in IA-32e mode (IA32_EFER.LMA = 1) and not in legacy protected mode (or real-address mode). The processor state defined by the FRED architecture is accessible with MSR-access instructions (e.g., RDMSR, WRMSR) regardless of mode.

The FRED architecture includes the concept of a **stack level**. The **current stack level (CSL)** is a value in the range 0–3 that a logical processor tracks when CPL = 0. FRED event delivery determines the stack level associated with the event being delivered and, if it is greater than the CSL (or if CPL had been 3), loads the stack pointer (RSP) from the **FRED RSP MSR** associated with the event's stack level. (If supervisor shadow stacks are enabled, the stack level applies also to the shadow-stack pointer, SSP, which would be loaded from a **FRED SSP MSR**.) The FRED return instruction ERETS restores the old stack level. (Details of the MSRs are given in Section 8.2.3.)

The existing shadow-stack architecture includes a token-management mechanism to ensure shadow-stack integrity when switching shadow stacks. This mechanism uses locked read-modify-write operations. FRED transitions do not use this token-management mechanism and instead protect shadow-stack integrity by having the FRED return instructions check values in the FRED SSP MSRs.

NOTE

Because FRED transitions do not use the shadow-stack token-management mechanism, FRED event delivery can never cause a fault or a VM exit after setting the busy bit in a shadow-stack token (unlike IDT event delivery). For that reason, an OS that enables FRED transitions need not consult CPUID.07H.1H:EDX.CET_SSS[18] before enabling supervisor shadow stacks.

8.2 ENUMERATION, ENABLING, AND CONFIGURATION

This section describes the enumeration, enabling, and configuration mechanisms for the FRED architecture.

8.2.1 Enumeration

The FRED architecture includes two related but independent elements: FRED transitions and the LKGS instruction for management of the GS segment register. Because 64-bit operating systems may benefit from the LKGS instruction without using FRED, the two elements are enumerated independently.

CPUID.07H.01H:EAX.FRED[17] is a feature flag that enumerates support for the FRED transitions. It also enumerates support for the architectural state (MSRs) defined by FRED.

CPUID.07H.01H:EAX.LKGS[18] is a feature flag that enumerates support for the LKGS instruction.

Any Intel® processor that enumerates support for FRED transitions will also enumerate support for LKGS.

8.2.2 Enabling in CR4

Software enables FRED transitions by setting CR4.FRED[32]. Specifically, setting CR4.FRED enables FRED event delivery (Section 8.3) as well as the FRED return instructions (Section 8.4), although those instructions can be used only in 64-bit mode (when CS.L = 1).

Execution of the MOV to CR4 instruction outside 64-bit mode clears CR4[63:32]. For that reason, software must first enter 64-bit mode before using MOV to CR4 to set CR4.FRED. The architecture ensures that CR4.FRED can be 1 only if the logical processor is in IA-32e mode (if IA32_EFER.LMA = 1):

- As just noted, MOV to CR4 can set CR4.FRED only in IA-32e mode.
- Software can leave IA-32e mode only by executing MOV to CR0 to clear CR0.PG, and this can be done only in compatibility mode while CPL = 0. (Attempts to clear CR0.PG in 64-bit mode will fault.)
- If CR4.FRED = 1, it is impossible to enter compatibility mode while CPL = 0 (see Section 8.5.2).
- RSM and VMX transitions cannot enter a state in which CR4.FRED = 1 outside IA-32e mode.

The value of CR4.FRED does not affect the accessibility of the FRED MSRs (Section 8.2.3) by MSR-access instructions (e.g., RDMSR, WRMSR), nor does it effect the operation of the LKGS instruction (Section 8.6).

Enabling FRED transitions changes the behavior of various existing instructions. See Section 8.5.

8.2.3 New MSRs for Configuration of FRED Transitions

FRED transitions are controlled by the following MSRs:

- IA32_FRED_CONFIG (MSR index 1D4H). This MSR is organized as follows:
 - Bits 1:0 of this MSR contain the **current stack level** (CSL). This 2-bit value is manipulated and used by FRED event delivery (Section 8.3) and the FRED return instructions (Section 8.4).
 - Software can modify the CSL with MSR-write instructions (e.g., WRMSR), but this is not an expected usage. It is recommended that, when using such an instruction to update IA32_FRED_CONFIG, software always write the existing CSL into bits 1:0 of the MSR (unless it specifically desires to change the CSL).
 - Bit 2 is reserved.

- Bit 3 indicates, if set, that, if FRED event delivery is not changing stacks, it should decrement the shadow-stack pointer (SSP) by 8.
- Bits 5:4 are reserved.
- Bits 8:6 identify the amount (measured in 64-byte cache lines) by which FRED event delivery decrements the regular stack pointer (RSP) when not changing stacks.
- Bits 10:9 identify the stack level that is used for maskable interrupts that occur while CPL = 0. See Section 8.3.1.2.
- Bit 11 is reserved.
- Bits 63:12 contain the upper bits of the linear address of a page in memory containing event handlers. FRED event delivery will load RIP to refer to an entry point on this page. See Section 8.3.1.1.

A write to this MSR (e.g., using WRMSR) causes a general-protection exception (#GP) if its operand sets any reserved bits or if it is not CPU canonical.

- IA32_FRED_RSP0, IA32_FRED_RSP1, IA32_FRED_RSP2, and IA32_FRED_RSP3 (MSR indices 1CCH–1CFH). These are the **FRED RSP MSRs**.

If FRED event delivery causes a change of the CPL (from 3 to 0) or a change to the CSL, it loads RSP from the FRED RSP MSR corresponding to the new stack level. See Section 8.3.1.2.

A write to any of these MSRs (e.g., using WRMSR) causes a general-protection exception (#GP) if its source operand is not 64-byte aligned (bits 5:0 are not all zero) or if it is not CPU canonical.

NOTE

The numbers 0–3 in the MSR names refer to the corresponding stack level and not to privilege level.

- IA32_FRED_STKLVL (MSR index 1D0H). This MSR is interpreted as an array of 32 2-bit values, one for each vector in the range 0–31. For an exception with vector v (or for a non-maskable interrupt, which has vector $v = 2$) that occurs CPL = 0, FRED event delivery ensures that the new stack level is at least the value of $\text{IA32\_FRED\_STKLVL}[2v+1:2v]$. See Section 8.3.1.2.
- IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3 (MSR indices 1D1H–1D3H). Together with the existing MSR IA32_PL0_SSP (MSR index 6A4H), these are the **FRED SSP MSRs**. References in this document to “IA32_FRED_SSP0” are to the IA32_PL0_SSP MSR.

If supervisor shadow stacks are enabled and FRED event delivery causes a change of the CPL (from 3 to 0) or a change to the CSL, it loads SSP from the FRED SSP MSR corresponding to the new stack level. See Section 8.3.1.2.

The address in each of IA32_FRED_SSPi ($1 \leq i \leq 3$) must be 8-byte aligned, so bits 2:0 are reserved. (IA32_PL0_SSP — also called IA32_FRED_SSP0 — is allowed to be 4-byte aligned, so only bits 1:0 are reserved.)

A write to any of these MSRs (e.g., using WRMSR) will cause a general-protection exception (#GP) if its source operand sets any reserved bits or if it is not CPU canonical. (The same is already true for executions of XRSTORS and MSR-write instructions that would load IA32_PL0_SSP.)

NOTES

As with the FRED RSP MSRs, the numbers 0–3 in the MSR names refer to the corresponding stack level **and not to privilege level**.

The FRED SSP MSRs are supported by any processor that enumerates CPUID.07H.01H:EAX.FRED[17] as 1. If such a processor does not support CET, FRED transitions will not use the MSRs (because shadow stacks are not enabled), but the MSRs would still be accessible using MSR-access instructions (e.g., RDMSR, WRMSR).

The RESET state of each of the new MSRs is zero. INIT does not change the value of the FRED MSRs.

8.2.4 FRED Use of Existing MSRs

In addition to the MSRs identified in Section 8.2.3, FRED transitions use the MSRs specified in the following items:

- IA32_STAR.
 - Use without FRED: this MSR is used by the existing operation of SYSCALL and SYSRET.
 - FRED use: FRED event delivery loads the CS and SS selectors with values derived from IA32_STAR[47:32] (see Section 8.3.3). ERETU uses the value of IA32_STAR[63:48] to determine how to load the CS and SS registers (see Section 8.4.2.1).

It is expected that operating systems will set IA32_STAR[33:32] to 00B and IA32_STAR[49:48] to 11B.
- IA32_KERNEL_GS_BASE.
 - Use without FRED: the SWAPGS instruction exchanges the value of this MSR with the base address of the GS segment register.
 - FRED use: executions of ERETU perform this same swapping operation, as does FRED event delivery of events that occur while CPL = 3.
- IA32_PL0_SSP.
 - Use without FRED: the shadow-stack feature of control-flow enforcement technology (CET) typically loads SSP from this MSR when the CPL changes to 0.
 - FRED use: FRED transitions use this MSR as IA32_FRED_SSP0 (Section 8.2.3).

8.3 FRED EVENT DELIVERY

When FRED transitions are enabled (CR4.FRED = 1), IDT event delivery of exceptions and interrupts is replaced with **FRED event delivery**. In addition, the existing operation of SYSCALL and SYSENTER is also replaced with FRED event delivery.

These changes do not affect the processor's handling of exceptions and interrupts prior to event delivery. For example, any determination that an event causes a VM exit or is converted into a double fault occurs normally. Similarly, page faults and debug exceptions update CR2 and DR6, respectively, in the normal way.

The principal functionality of FRED event delivery is to establish a new context, that of the event handler at CPL 0, while saving the old context for a subsequent return. Some parts of the new context have fixed values, while others depend on the old context, the nature of the event being delivered, and software configuration. Section 8.3.1 describes how FRED event delivery determines and establishes the new context. Section 8.3.2 specifies how FRED event delivery saves elements of the old context on the stack (and, when enabled, the shadow stack). Section 8.3.3 then describes how additional state is updated. Section 8.3.4 gives details about faults encountered during FRED event delivery.

Section 8.3.5 presents the detailed flow of the operation of FRED event delivery.

8.3.1 Determining and Establishing the New Context

The context of an event handler invoked by FRED event delivery includes the CS and SS segment registers, the instruction pointer (RIP), the flags register (RFLAGS), the stack pointer (RSP), and the base address of the GS segment (GS.base). The context also includes the shadow-stack pointer (SSP) if supervisor shadow stacks are enabled.

FRED event delivery establishes this context by loading these registers appropriately. It determines the values to be loaded into RIP, RSP, GS.base, and SSP based on the old context, the nature of the event being delivered, and software configuration. RFLAGS, CS, and SS are loaded with fixed values.

8.3.1.1 Determining the New RIP Value

FRED event delivery uses two entry points, depending on the CPL at the time the event occurred. This allows an event handler to identify the appropriate return instruction (ERETU or ERETS).

Specifically, the new RIP value that FRED event delivery establishes is `IA32_FRED_CONFIG & ~FFFH` for events that occur while `CPL = 3` and `(IA32_FRED_CONFIG & ~FFFH) + 256` for events that occur while `CPL = 0`.

If the new RIP value is not paging canonical, FRED event delivery causes a general-protection fault (`#GP`).¹

8.3.1.2 Determining the New Values for Stack Level, RSP, and SSP

FRED transitions support four different stacks for use while `CPL = 0`. A logical processor identifies the stack currently in use with a 2-bit value called the **current stack level (CSL)**.

FRED event delivery first determines the event's stack level and then uses that to determine whether the CSL should change. An event's stack level is based on the CPL, the nature and type of the event, the event's vector (for some event types), and MSRs configured by system software:

- If the event occurred while `CPL = 3`, was not a nested exception encountered during event delivery, and was not a double fault (`#DF`), the event's stack level is 0.
- If the event occurred while `CPL = 0`, was a nested exception encountered during event delivery, or was a `#DF`, the following items apply:
 - If the event is a maskable interrupt, the event's stack level is `IA32_FRED_CONFIG[10:9]`.
 - If the event is an exception or a non-maskable interrupt (NMI), the event's stack level is `IA32_FRED_STKLVL[2v+1:2v]`, where `v` is the event's vector (in the range 0–31).²
 - The stack level of all other events is 0.³

If the event occurred while `CPL = 3`, the new stack level is the event's stack level; otherwise, the new stack level is the maximum of the CSL and the event's stack level. (This implies that the CSL is always 0 following FRED event delivery of an event that occurred while `CPL = 3`, unless the event was a nested exception or a `#DF`.)

After determining the new stack level, FRED event delivery identifies the new RSP value as follows:

- If either the CPL or the CSL is changing, the new RSP value will be that of the FRED RSP MSR corresponding to the new stack level.
- Otherwise, the new RSP value will be the current RSP value decremented by `IA32_FRED_CONFIG & 1C0H` (the multiple of 64 in bits 8:6 of that MSR) and then aligned to a 64-byte boundary (by clearing `RSP[5:0]`).

If supervisor shadow stacks are enabled, the following items explain how the new SSP value is determined:

- If either the CPL or the CSL is changing, the new SSP value will be that of the FRED SSP MSR corresponding to the new stack level. If that new SSP value is not 8-byte aligned, FRED event delivery causes a general-protection fault (`#GP`).⁴
- Otherwise, the new SSP value will be the current SSP value decremented by `IA32_FRED_CONFIG & 8` (setting bit 3 of that MSR indicates that SSP should be decremented by 8).

8.3.1.3 Establishing the New Context

After determining the details of the new context, FRED event delivery loads registers to establish that context.

For events that occur while `CPL = 3`, FRED event delivery updates the CS, SS, and GS segments as well as the `IA32_KERNEL_GS_BASE` MSR:

- CS:
 - The selector is set to `IA32_STAR[47:32] AND FFFCH` (this forces `CS.RPL` to 0).

1. If this `#GP` does not cause a VM exit, FRED event delivery of the `#GP` will cause a second `#GP`, which will be converted to a double fault (`#DF`). If the `#DF` does not cause a VM exit, FRED event delivery of the `#DF` will cause a third `#GP`, resulting in a triple fault, which will either cause a VM exit or take the logical processor to shutdown.
2. Exceptions include the events generated by the `INT1`, `INT3`, and `INT0` instructions, but not those generated by the `INT n` instruction (for any value of `n`).
3. The other events are those generated by the instructions `INT n` (for any value of `n`), `SYSCALL`, and `SYSENTER`.
4. The value of each of `IA32_FRED_SSPI` ($1 \leq i \leq 3$) is always 8-byte aligned, so the check specified above is necessary only if SSP is being loaded from the `IA32_PLO_SSP` MSR (`IA32_FRED_SSPO`).

- The base address is set to 0. The limit is set to FFFFFFFH and the G bit is set to 1.
- The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
- The DPL is set to 0, the P and L bits are each set to 1, and the D bit is set to 0.
- CS is made usable regardless of the value of the selector.
- SS:
 - The selector is set to the new value of the CS selector (above) plus 8 (thus, SS.RPL is also 0).
 - The base address is set to 0. The limit is set to FFFFFFFH and the G bit is set to 1.
 - The type is set to 3 (read/write accessed data) and the S bit is set to 1.
 - The DPL is set to 0, and the P and B bits are each set to 1.
 - SS is made usable regardless of the value of the selector.
- GS: FRED event delivery swaps the value of the GS base address and that of the IA32_KERNEL_GS_BASE MSR.

(Note that the above updates occur without any access to the GDT or an LDT.)

For events that occur while CPL = 0, FRED event delivery does not modify CS, SS, or GS.

After updating the segment registers, FRED event delivery loads RIP, RSP, and CSL with the values determined in Section 8.3.1.1 and Section 8.3.1.2. If supervisor shadow stacks are enabled, SSP is loaded with the value determined in Section 8.3.1.2. RFLAGS is loaded with the value 2 (all bits clear except position 1, which has no function but is always set).

If FRED event delivery incurs a nested exception or VM exit after this point, the processor restores the values that were in these registers before FRED event delivery commenced and only then delivers the nested exception or VM exit.

8.3.2 Saving Information About the Event and the Old Context

Like IDT event delivery, FRED event delivery saves information about the old context on the stack of the event handler. This information includes the previous context saved in much the same format as that following IDT event delivery, additional information about the event being delivered, and auxiliary information that will guide a subsequent return instruction.

If supervisor shadow stacks are enabled, FRED event delivery also saves information on the event handler's shadow stack.

Since FRED event delivery results in CPL = 0, the memory accesses used to store information on the stacks are performed with supervisor privilege. Any of these memory accesses may result in nested exception. The nested exception (or a double fault to which it may be converted) would then be delivered with FRED event delivery.

8.3.2.1 Saving Information on the Regular Stack

The format of the stack frame is illustrated in Figure 8-1 and detailed in the following items.

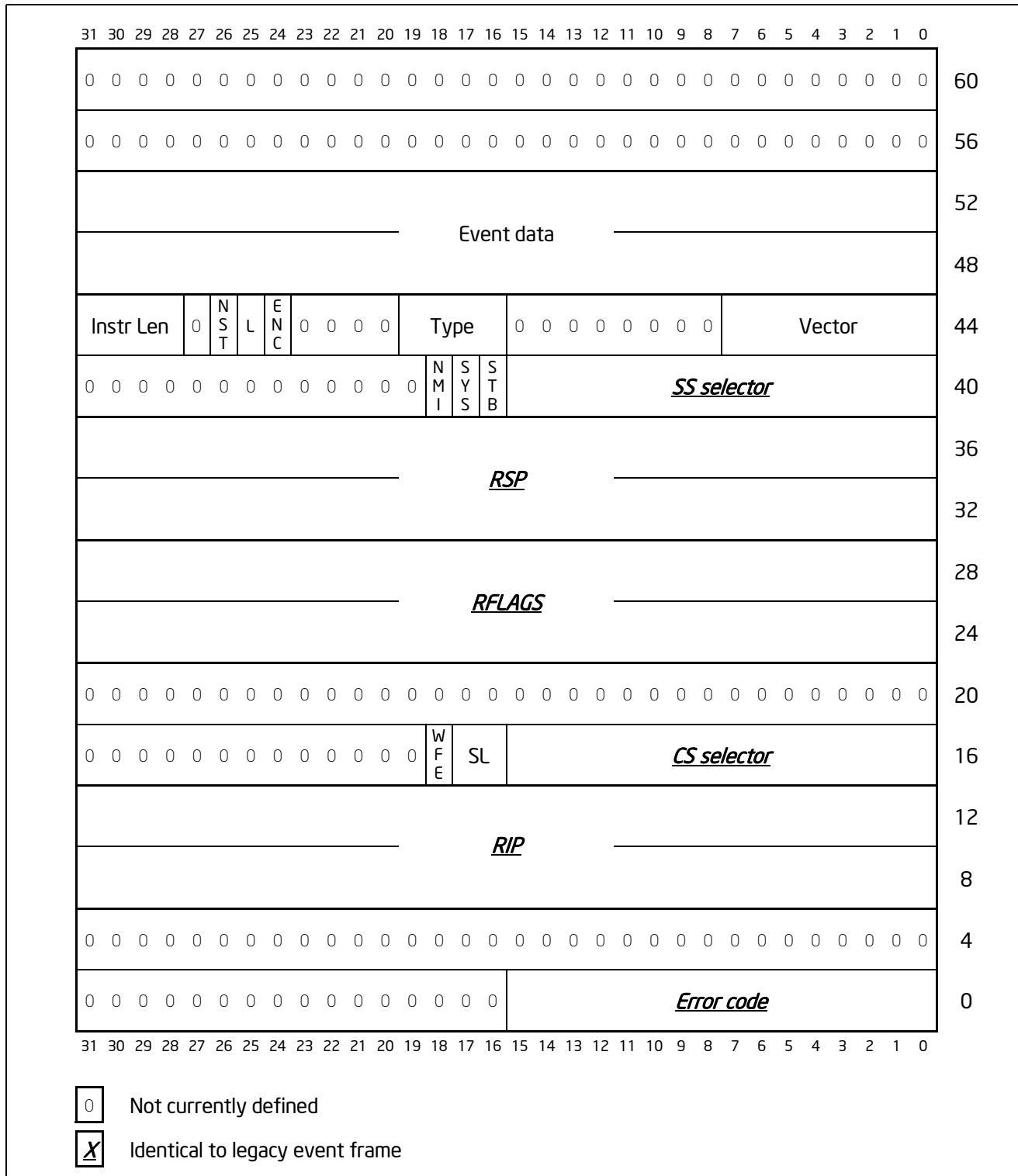


Figure 8-1. Flexible Return and Event Delivery Stack

- The first 8 bytes pushed (bytes 63:56 of the 64-byte stack frame) are not currently defined and will be zero until they are.

- The next 8 bytes pushed (bytes 55:48) contain **event data** and are defined as follows:
 - If the event being delivered is a page fault (#PF), the event data is the faulting linear address (this is the same value that the #PF loads into CR2).¹
 - If the event being delivered is a debug exception (#DB), the event data identifies the nature of the debug exception:
 - Bits 3:0 are B3–B0. When set, each of these bits indicates that the corresponding breakpoint condition was met (and may be set even if its corresponding enabling bit in DR7 is not set).
 - Bits 10:4 are not currently defined and will be zero until they are.
 - Bit 11 is BLD. When set, this bit indicates that the cause of the debug exception was acquisition of a bus lock (because IA32_DEBUGCTL[2] = 1).
 - Bit 12 is not currently defined and will be zero until it is.
 - Bit 13 is BD. When set, this bit indicates that the cause of the debug exception was “debug register access detected.”
 - Bit 14 is BS. When set, this bit indicates that the cause of the debug exception was the execution of a single instruction (because execution of that instruction commenced with RFLAGS.TF = 1).²
 - Bit 15 is not currently defined and will be zero until it is.
 - Bit 16 is RTM. When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled.
 - Bits 63:17 are not currently defined and will be zero until they are.

The event data is not exactly the same as that which will be in DR6 following delivery of the #DB. The polarity of bit 11 and bit 16 is inverted in DR6; in addition, delivery of #DB may leave unmodified in DR6 some bits in positions that are clear in the event data and that were already set in DR6.

- If the event being delivered is a device-not-available exception (#NM) caused by extended feature disable, the event data indicates the cause of the exception and is the same as that loaded into the IA32_XFD_ERR MSR. If the #NM was not caused by extended feature disable, the event data is zero.
- For any other event, the event data are not currently defined and will be zero until they are.
- The next 40 bytes pushed (bytes 47:8) are the **return state** and have generally the same format as that used by IDT event delivery. That format stores the 16-bit selectors for CS and SS in 64-bit fields. FRED event delivery augments those selector values with additional information saved in the upper 48 bits of those fields.

The following items detail the format of the return state on the stack from bottom (highest address) to top:

- The augmented SS of the previous context, 64 bits formatted as follows:
 - Bits 15:0 contain the SS selector.
 - Bit 16 is set to 1 for FRED event delivery of a hardware exception if interrupt blocking by STI (**STB**) was in effect at the time the exception occurred³ and is otherwise cleared to 0.⁴
 - Bit 17 is set to 1 for FRED event delivery of SYSCALL, SYSENTER, or INT n (**SYS**) and is otherwise cleared to 0.
 - Bit 18 is set to 1 if the event being delivered is a non-maskable interrupt (**NMI**) and is otherwise cleared to 0.
 - Bits 31:19 are not currently defined and will be zero until they are.

1. If the #PF occurred during execution of an instruction in enclave mode (but not during delivery of an event incident to enclave mode), bits 11:0 of the event data are cleared.

2. If in addition IA32_DEBUGCTL.BTF was 1 when the instruction commenced, a debug exception occurs only if the instruction was a taken branch.

3. Execution of STI with RFLAGS.IF = 0 blocks interrupts on the instruction boundary following its execution.

4. Delivery of SYSCALL, SYSENTER, INT1, INT3, or INT n (for any value of n) will always clear this bit, as will delivery of any exception encountered during event delivery for one of those instructions.

- Bits 63:32 contain **event information**. This provides software with additional information about the event. The FRED return instructions ignore these bits. The event information is formatted as follows:
 - Bits 39:32 contain the event's vector. The vector depends on the type of event:
 - For an external interrupt, the vector is provided by the local APIC.
 - For a non-maskable interrupt (NMI), the vector is 2.
 - For a hardware exception, the vector is determined by the exception.
 - For a software interrupt (INT n), the vector comes from the instruction's opcode (value n).
 - For a software exception, the vector is determined by the instruction opcode: INT1 has vector 1; INT3 has vector 3; INTO has vector 4.
 - For SYSCALL and SYSENTER (which use FRED event delivery but not IDT event delivery), vectors 1 and 2 are used, respectively.
 - Bits 47:40 are not currently defined and will be zero until they are.
 - Bit 51:48 encode the event type as follows: 0 = external interrupt; 2 = non-maskable interrupt; 3 = hardware exception (e.g., page fault); 4 = software interrupt (INT n); 5 = privileged software exception (INT1); 6 = software exception (INT3 or INTO); and 7 = other event (used for SYSCALL and SYSENTER). Other values are not used.¹
 - Bits 55:52 are not currently defined and will be zero until they are.
 - Bit 56 is set to 1 to indicate that the event was incident to enclave execution (**ENC**). Specifically, it is set if the event occurred while the logical processor was in enclave mode or was an exception encountered during delivery of such an event. Otherwise, the bit is cleared to 0.
 - Bit 57 (**L**) is set to 1 if the logical processor had been in 64-bit mode when the event occurred. (A value of 0 indicates an event incident to compatibility mode.)
 - Bit 58 is set to 1 if the event is nested exception (**NST**) encountered during FRED event delivery of another event. This bit is not set if the event is double fault (#DF).
 - Bit 59 is not currently defined and will be zero until it is.
 - Bits 63:60 contain the length of the instruction causing the event if the event type is software interrupt (INT n), privileged software exception (INT1), software exception (INT3 or INTO), or other event (when used for SYSCALL or SYSENTER). For other event types, these bits are cleared to zero.
 - RSP of the previous context (64 bits).
 - RFLAGS of the previous context (64 bits).
- Bit 16 of the RFLAGS field (corresponding to the RF bit) is saved as 1 when delivering events that do the same for IDT event delivery. These are faults other than instruction breakpoints as well as any traps or interrupts delivered following partial execution of an instruction (e.g., between iterations of a REP-prefixed string instruction). Delivery of other events save in bit 16 the value that RFLAGS.RF had at the time the event occurred.
- The augmented CS of the previous context. The FRED return instructions use the 64 bits of this field. It is formatted as follows:
 - Bits 15:0 contain the CS selector.
 - Bits 17:16:
 - For delivery of events that occur while CPL = 0, these bits report the stack level (**SL**) at the time the event occurred.
 - For delivery of events that occur while CPL = 3, these bits are cleared to 0.
 - Bit 18 is set to 1 if the event occurred while CPL = 0 and the indirect branch tracker was in the WAIT_FOR_ENDBRANCH (**WFE**) state. Specifically, the bit is set if and only if the following are all true: CPL = 0; CR4.CET = 1; IA32_S_CET.ENDBR_EN = 1; and IA32_S_CET.TRACKER = 1.

1. The event types used by FRED event delivery are the same as those already defined for VMX transitions. For SYSCALL or SYSENTER, FRED event delivery reports event type 7 (other event).

The following items document the implications of how bit 18 is saved based on the existing operation of indirect branch tracking (providing details regarding delivery of specific events occurring while CPL = 0):

- Delivery of an event occurring at an instruction boundary saves bit 18 with the setting of the tracker at that instruction boundary:
 - For an event that takes priority over a missing-ENDBRANCH #CP exceptions, this value may be 0 or 1, depending on the state of the tracker. These events include certain machine-check exceptions, trap-class debug exceptions, non-maskable interrupts, external interrupts, debug exceptions due to instruction breakpoints, and faults from fetching the next instruction. They may also include faults from decoding the next instruction.
 - For a missing-ENDBRANCH #CP exception, this value is 1.
 - For a fault from decoding the next instruction that is prioritized below missing-ENDBRANCH #CP exceptions, this value is 0.
- Delivery of INT1 or INT3 saves bit 18 with the tracker's setting at the time INT1 or INT3 was fetched. (Decode of these instructions does not clear the tracker.)
- Delivery of INT n (for any value of n), SYSCALL, or SYSENTER will save bit 18 with value 0.
- Delivery of an exception encountered during FRED event delivery (including that for INT1, INT3, INT n, SYSCALL, or SYSENTER) will save bit 18 with the value that would have been saved by delivery of the original event.
- Delivery of an exception encountered during execution of any instruction other than INT1 or INT3 will save bit 18 with value 0.
- Bits 63:19 are not currently defined and will be zero until they are.
- RIP of the previous context (64 bits).

If the event type is software interrupt (INT n), privileged software exception (INT1), software exception (INT3 or INTO), or other event (when used for SYSCALL or SYSENTER); the RIP value saved references the instruction after the one that caused the event being delivered. (If delivery of such an event encounters an exception, the RIP value saved by delivery of the exception will instead reference the instruction that caused the original event.)

- The last 8 bytes pushed (bytes 7:0) contain error code (defined only for certain exceptions; zero if none is defined) in bits 15:0. Bits 63:16 are not currently defined and will be zero until they are.

8.3.2.2 Saving Information on the Shadow Stack

If supervisor shadow stacks are enabled and the event being delivered occurred while CPL = 0, FRED event delivery saves information on the new shadow stack. The following items provide details:

- FRED event delivery writes four bytes of zeroes to the linear address SSP – 4 and then clears SSP[2:0]. This has the following effect:
 - If the value of SSP had not been 8-byte aligned (it is necessarily 4-byte aligned), 4 bytes of zeroes are stored (as padding) and subsequent shadow-stack pushes will immediately follow those 4 bytes of zeroes.
 - If the value of SSP had been 8-byte aligned, 4 bytes of zeroes are stored but will be overwritten by subsequent shadow-stack pushes.
- FRED event delivery then pushes the following values on the new shadow stack:
 - The same 64-bit augmented CS that was pushed on the regular stack (see Section 8.3.2.1), including any bits defined in positions 63:16.
 - The old linear instruction pointer.
 - The old SSP on the shadow stack.

Each of these values is pushed in a separate 8-byte field on the shadow stack.

The write of zeroes and the pushes are all performed as shadow-stack accesses; because FRED event delivery results in CPL = 0, they are performed with supervisor privilege.

8.3.3 Loading Additional State

After saving the old context and other information, FRED event delivery completes operation by loading additional registers and updating other processor state.

If the event occurred while $CPL = 3$ and user shadow stacks are enabled, the `IA32_PL3_SSP` MSR is loaded with the old value of `SSP`. The value loaded into the MSR is adjusted so that bits 63:N get the value of bit N-1, where N is the processor's maximum linear-address width.

If supervisor indirect branch tracking is enabled, the `IA32_S_CET` MSR is updated to set the `TRACKER` value to `IDLE` and to clear the `SUPPRESS` bit to 0. There is no reason for the instruction referenced by the new `RIP` value to be `ENDBR64`.

A debug trap (single-step trap or data or I/O breakpoint) may be pending at the time another event is delivered. FRED event delivery drops any debug traps that were pending at the time the original event occurred, regardless of the event being delivered. In particular, any pending data or I/O breakpoints (or single-step traps) are no longer pending after `INT n`, `INT3`, `INTO`, `SYSCALL`, or `SYSENTER` is delivered using FRED event delivery.

Data breakpoints detected during FRED event delivery might not be dropped and may be pending after completion of FRED event delivery. A single-step trap will never be pending following FRED event delivery (even if `INT n`, `INT1`, `INT3`, `INTO`, `SYSCALL`, or `SYSENTER` is executed while `RFLAGS.TF = 1`).

8.3.4 Faults During FRED Event Delivery

The earlier sections identify situations in which faults may occur during FRED event delivery. In addition, a memory access will cause a fault if it uses a non-canonical address or encounters a LASS violation¹ (`#SS` for an access to the regular stack; `#GP` for the shadow stack); or if it causes a page fault (`#PF`).

An exception encountered during FRED event delivery will be converted to a double fault (`#DF`) following the same principles under which this is done for IDT event delivery. As with IDT event delivery, the error code for the `#DF` is always zero.

For a `#GP` or `#SS` encountered during FRED event delivery that is not converted to `#DF`, the error code will be 0 if the event that was being delivered had type software interrupt (`INT n`), software exception (`INT3` or `INTO`), or other event (`SYSCALL` and `SYSENTER`); it will be 1 for event types external interrupt, non-maskable interrupt, hardware exception, or privileged software exception (`INT1`). (This aligns with the existing specification of the `EXT` bit in such error codes.)

For a `#PF` encountered during FRED event delivery that is not converted to `#DF`, the error code is defined exactly as it is for IDT event delivery.

If FRED event delivery causes a fault, any changes to register state are undone before delivering the fault.

8.3.5 Detailed Operation of FRED Event Delivery

The section presents detailed pseudocode for the operation of FRED event delivery. It does not include details of how event information is formatted or how event data is determined. These details are in Section 8.3.2.1.

```
// HOLD OLD STATE IN TEMPORARIES
oldRIP := RIP;
oldCS := CS;           // oldCS is 8 bytes; the upper 6 bytes are defined below
oldCPL := CPL;         // the same as oldCS[1:0]
oldRFLAGS := RFLAGS;
oldSS := SS;           // oldSS is 8 bytes; upper 6 bytes are zero
oldRSP := RSP;
IF CPL = 3
    THEN oldCSL := 0;
    ELSE oldCSL := CSL;
```

1. Since FRED event delivery clears both `CPL` and `RFLAGS.AC`, a LASS violation will occur if `CR4.LASS = CR4.SMAP = 1` and a memory access uses a linear address that clears bit 63.

FLEXIBLE RETURN AND EVENT DELIVERY (FRED)

```
FI;
oldGSB := GS.base;
oldSSP := SSP;                                // used only when shadow stacks are enabled
// ERETS and ERETU will restore the oldRFLAGS to RFLAGS
// Before saving, event delivery updates oldRFLAGS to set RF when appropriate
IF the event being delivered is a fault other than an instruction breakpoint OR
   the event being delivered is a trap or interrupt following partial execution of an instruction
   (e.g., between iterations of a REP-prefixed string instruction)
   THEN oldRFLAGS[16] := 1;
FI;
// Update bits above CS selector to hold additional information
// ERETS and ERETU will use these
oldCS[17:16] := oldCSL;
IF CPL = 3
   THEN oldCS[18] := 0;
   ELSE // See Section 8.3.2.1 for details of how oldCS[18] is determined
        oldCS[18] = CR4.CET AND IA32_S_CET.ENDBR_EN AND IA32_S_CET.TRACKER;
FI;
// Update bits above SS selector to hold additional information
IF STI blocking was in effect when the event occurred
   THEN oldSS[16] := 1;
FI;
IF event being delivered is either SYSCALL, SYSENTER, or INT n
   THEN oldSS[17] := 1;
FI;
IF event being delivered is an NMI                // includes VM-entry injection of NMI
   THEN oldSS[18] := 1;
FI;
// ERETS and ERETU will ignore the event information
oldSS[63:32] := event information as defined in Section 8.3.2.1;
// DETERMINE NEW CONTEXT
// determine new RIP
IF oldCPL = 3
   THEN newRIP := IA32_FRED_CONFIG & ~FFFH;
   ELSE newRIP := IA32_FRED_CONFIG & ~FFFH + 256;
FI;
IF newRIP is not paging canonical
   THEN #GP(0);                                // will lead eventually to shutdown or VM exit
FI;
// determine event's stack level and the new stack level for event handler
IF oldCPL = 3 AND event is not an exception nested on event delivery AND event is not #DF
   THEN eventSL := 0;
ELSE
   IF event type is external interrupt                // oldCPL must be 0
      THEN eventSL := IA32_FRED_CONFIG[10:9];
   ELSEIF event type is hardware exception, software exception, privileged software exception, or NMI
      // v = event vector (0–31); includes INT1, INT3, INTO
      THEN eventSL := IA32_FRED_STKLVL[2 v + 1:2 v];
      ELSE // SYSCALL, SYSENTER, INT n with CPL = 0; do not change CSL
           eventSL := 0;
FI;
FI;
newCSL := MAX[eventSL, oldCSL];
```

```

// determine new RSP
IF oldCPL = 3 OR newCSL > oldCSL
    THEN newRSP := IA32_FRED_RSPi, where i = newCSL;
    ELSE      // decrement RSP as specified and align to 64 bytes
        newRSP := (RSP - (IA32_FRED_CONFIG & 1COH)) & ~3FH;
FI;
// determine new RFLAGS; clear all bits except position 1
newRFLAGS := 2;
// if needed, determine new SSP
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
    THEN
        IF oldCPL = 3 OR newCSL > oldCSL
            THEN
                newSSP := IA32_FRED_SSPi, where i = newCSL;
                IF newSSP[2] = 1      // check for 8-byte alignment
                    THEN #GP(0);
                FI;
            ELSE      // decrement SSP if specified
                newSSP := SSP - (IA32_FRED_CONFIG & 8);
        FI;
FI;
// ESTABLISH NEW CONTEXT — OLD STATE WILL BE RESTORED IF THERE IS A SUBSEQUENT EXCEPTION
// update segment registers if event occurred while CPL = 3
IF oldCPL = 3
    THEN
        // set CS to standard values used by a 64-bit operating system
        CS.selector := IA32_STAR[47:32] & FFFCH;
        CS.base := 0;
        CS.limit := FFFFFFFH;
        CS.type := 11;
        CS.S := 1;
        CS.DPL := 0;
        CS.P := 1;
        CS.L := 1;
        CS.D := 0;
        CS.G := 1;
        CS.unusable := 0;
        // set SS to standard values used by a 64-bit operating system
        SS.selector := CS.selector + 8;
        SS.base := 0;
        SS.limit := FFFFFFFH;
        SS.type := 3;
        SS.S := 1;
        SS.DPL := 0;
        SS.P := 1;
        SS.B := 1;
        SS.G := 1;
        SS.unusable := 0;
        // swap in supervisor GS base address
        GS.base := IA32_KERNEL_GS_BASE;
        IA32_KERNEL_GS_BASE := oldGSB;
    FI;
// update registers defining context

```

```

RIP := newRIP;
RFLAGS := newRFLAGS;
RSP := newRSP;
CSL := newCSL;                                // reflected in IA32_FRED_CONFIG[1:0]
// Update SSP if supervisor shadow stacks enabled
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
    THEN SSP := newSSP;
FI;

// SAVE STATE ON STACKS
// Save return state on new regular stack; memory accesses here have supervisor privilege
push8B 00000000_00000000H;                    // First 8 bytes pushed are all zero
push8B event data as defined in Section 8.3.2.1;
push8B oldSS;
push8B oldRSP;
push8B oldRFLAGS;
push8B oldCS;
push8B oldRIP;
push8B errorcode;

// If supervisor shadow stacks enabled and old CPL was 0, save data on new shadow stack
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1 AND oldCPL = 0
    THEN
        store 4 bytes of zeroes to address SSP - 4; // shadow-stack access
        SSP := SSP & ~7;                          // aligns SSP to 8B boundary
        pushSS_8B oldCS;                          // full 64-bit field
        pushSS_8B oldRIP;
        pushSS_8B oldSSP;
    FI;

// UPDATE ADDITIONAL STATE
// update additional CET state as needed (SSP was updated earlier)
IF CR4.CET = 1
    THEN
        IF oldCPL = 3 AND IA32_U_CET.SH_STK_EN = 1
            THEN // adjust so bits 63:N get the value of bit N-1
                    // N = processor's maximum linear-address width
                    IA32_PL3_SSP := LA_adjust(oldSSP);
            FI;
        IF IA32_S_CET.ENDBR_EN = 1
            THEN IA32_S_CET[11:10] := 00b;        // IDLE with SUPPRESS = 0
            FI;
    FI;

// update event-related state
clear any debug traps that were pending prior to FRED event delivery;
CPL := 0;

```

8.4 FRED RETURN INSTRUCTIONS

FRED defines two instructions for returning from events delivered by FRED event delivery:

- **ERETS** (Section 8.4.1) is used to return from events that occur while $CPL = 0$; it does not modify CS, SS, or GS.
- **ERETU** (Section 8.4.2) is used to return from events that occur while $CPL = 3$; it loads CS and SS based on the stack image and also updates the GS base address.

Neither instruction takes explicit operands. These instructions can be executed only if FRED transitions are enabled and only if CPL = 0; in addition, ERETU can be executed only if CSL = 0.

NOTE

The following sections indicate situations in which one of the instructions may cause a control-protection exception (#CP). These exceptions will use the value 2 for an error code, indicating that the #CP was caused by a return instruction other than near RET.

8.4.1 ERETS (Event Return to Supervisor)

ERETS returns from an event handler without changing the CPL, establishing the **return state** that was in effect before FRED event delivery. Because it stays within the supervisor context, ERETS does not modify the segment registers CS, SS, or GS. Execution of ERETS causes an invalid-opcode exception (#UD) if FRED transitions are not enabled or if CPL > 0.

ERETS begins by reading and checking the return state from the stack (Section 8.4.1.1). If supervisor shadow stacks are enabled, it then checks the shadow stack to confirm the validity of this control-flow transfer (Section 8.4.1.2). Finally, ERETS establishes the return state by loading the appropriate registers (Section 8.4.1.3).

See “ERETS—Event Return to Supervisor” in Chapter 3 of Volume 2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual for the detailed flow of the operation of ERETS.

8.4.1.1 Loading and Checking the Return State

ERETS operates on the stack frame created by FRED event delivery (Section 8.3.2.1), principally to restore the state saved in that stack frame (the return state).

When FRED event delivery completes, the return state is located at offset 8 from the RSP value. (RSP references an 8-byte field containing the event’s error code.) For this reason, ERETS begins by adding 8 to RSP so that it can access the return state directly.

After adding 8 to RSP, ERETS pops from the regular stack (referenced by RSP) the return state that was saved by FRED event delivery. The state is checked and held by the processor to update register state when the instruction completes. The following items detail the state fields that are popped, from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The RIP of the return state (64 bits).
This field must be paging canonical; otherwise, a #GP with a zero error code occurs.
- The augmented CS for the return state (64 bits).
ERETS does not use bits 15:0 to load CS, but it does ensure that the value of these bits equals that of the current CS selector; otherwise, a #GP occurs.
Bit 18 of this field determines whether ERETS updates the state of the indirect branch tracker.
ERETS will establish the new stack level as the minimum of the CSL and the value of bits 17:16 of this field.
Bits 63:19 of this field must be zero; otherwise, a #GP occurs.
- The RFLAGS of the return state (64 bits).
Bit 1 of this field must be 1; bit 3, bit 5, bit 15, bit 17 (VM), and bits 63:22 of the field must be zero; otherwise, a #GP occurs.
- The RSP of the return state (64 bits). This field is not checked.
- The augmented SS of the return state (64 bits).
ERETS does not use bits 15:0 to load SS, but it does ensure that the value of these bits equals that of the current SS selector; otherwise, a #GP occurs.
Bits 18:16 of this field determine how ERETS manages certain events. See Section 8.4.1.3.
Bits 31:19 of this field must be zero; otherwise, a #GP occurs. ERETS ignores bits 63:32 of this field (the event information).

8.4.1.2 Checking the Shadow Stack

If supervisor shadow stacks are enabled, ERETS pops and checks values from the shadow stack (referenced by SSP) that were saved by FRED event delivery. The following items detail the state that is popped from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The SSP of the return state (64 bits).
This value must be 4-byte aligned (bits 1:0 must be zero); otherwise, a control-protection exception (#CP) occurs.
This value must be CPU canonical; otherwise, a #GP occurs with a zero error code. (This #GP has priority below the #CP exceptions specified in the following items.)
- The linear instruction pointer of the return state (64 bits).
This value must equal the RIP of the return context that was popped from the regular stack; otherwise, a #CP occurs.
- The augmented CS for the return state (64 bits).
This value must equal the code-segment value that was popped from the regular stack; otherwise, a #CP occurs.

If supervisor shadow stacks are enabled and the new stack level is less than the CSL, ERETS compares the value of SSP (after popping the above fields) and the value of the FRED SSP MSR for the CSL (not the new stack level). (For example, if ERETS is executing with CSL = 2 and is returning to stack level 1, the comparison is with IA32_FRED_SSP2.) If the values are not equal, a #CP occurs.

8.4.1.3 Establishing the Return State

After the stack operations described in Section 8.4.1.1 and Section 8.4.1.2, ERETS then loads RIP, RFLAGS, RSP, and CSL with the values that were popped earlier from the regular stack. If supervisor shadow stacks are enabled, SSP is loaded with the value that was popped earlier from the shadow stack.

Additional steps are performed based on the value of the popped fields for CS and SS:

- If bit 18 of the CS field is 1 and ERETS will complete with indirect branch tracking enabled and not suppressed, the indirect branch tracker will be in the WAIT_FOR_ENDBRANCH state upon completion of ERETS.¹
- If bit 16 of the SS field is 1, bit 9 of the RFLAGS field (corresponding to RFLAGS.IF) is 1, and there was no blocking by STI prior to ERETS, blocking by STI is in effect upon completion of ERETS.
- If bit 17 of the SS field is 1 and ERETS will result in RFLAGS.TF = 1, a single-step trap will be pending upon completion of ERETS.²
- If bit 18 of the SS field is 1, ERETS unblocks NMIs.

8.4.2 ERETU (Event Return to User)

ERETU returns from an event handler while changing the CPL to 3, establishing the **return context** that was in effect before FRED event delivery. The change of context includes updates to the segment registers CS, SS, or GS. Execution of ERETU causes an invalid-opcode exception (#UD) if FRED transitions are not enabled or if CPL > 0, and a general-protection exception (#GP) if CSL > 0.

ERETU begins by reading and checking the return state from the stack (Section 8.4.2.1). If supervisor shadow stacks are enabled, it then checks the validity of the old and new shadow-stack pointers (Section 8.4.2.2). Finally, ERETU establishes the return state by loading the appropriate registers (Section 8.4.2.3).

-
1. ERETS will complete with indirect branch tracking enabled and not suppressed if CR4.CET = 1, IA32_S_CET.ENDBR_EN = 1, and IA32_S_CET.SUPPRESS = 0. If ERETS puts the indirect branch tracker in the WAIT_FOR_ENDBRANCH state, it will result in IA32_S_CET.TRACKER = 1.
 2. If ERETS begins execution with RFLAGS.TF = 1, there will always be a single-step debug exception pending after ERETS, regardless of the values on the stack for RFLAGS.TF and CS.

See “ERETU—Event Return to User” in Chapter 3 of Volume 2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual for the detailed flow of the operation of ERETU.

8.4.2.1 Loading and Checking the Return State

ERETU operates on the stack frame created by FRED event delivery (Section 8.3.2.1), principally to restore the state saved in that stack frame (the return state).

When FRED event delivery completes, the return state is located at offset 8 from the RSP value. (RSP references an 8-byte field containing the event’s error code.) For this reason, ERETU begins by adding 8 to RSP so that it can access the return state directly.

After adding 8 to RSP, ERETU first pops from the regular stack (referenced by RSP) the return state that was saved by FRED event delivery. The state is checked and held by the processor to update register state when the instruction completes. The following items detail the state fields that are popped, from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The RIP of the return state (64 bits).
This field is checked once the new CS configuration is determined (see below).
- The augmented CS of the return state (64 bits).
Bits 15:0 contain the new CS selector itself. ERETU ensures that the value of bits 1:0 is 3; otherwise, a #GP occurs.
Bits 63:16 of this field must be zero; otherwise, a #GP occurs.
- The RFLAGS of the return state (64 bits).
Bit 1 of this field must be 1; bit 3, bit 5, bits 13:12 (IOPL), bit 15, bit 17 (VM), and bits 63:22 of the field must be zero; otherwise, a #GP occurs.

NOTE

Checking IOPL ensures that, when FRED transitions are enabled, IOPL is always 0 when CPL = 3.

- The RSP of the return state (64 bits). This field is not checked.
- The augmented SS of the return state (64 bits).
Bits 15:0 contain the new SS selector itself. ERETU ensures that the value of bits 1:0 is 3; otherwise, a #GP occurs.
Bit 17 and bit 18 of this field determine how ERETU manages certain events. See Section 8.4.2.3.
Bits 31:19 of this field must be zero; otherwise, a #GP occurs. ERETU ignores bit 16 of this field, as well as bits 63:32 (the event information).

After popping these fields, ERETU determines the configuration of the CS and SS segment registers for the return state according to one of these three cases:

1. If the selector popped for CS is `IA32_STAR[63:48] + 16` and the selector popped for SS is `IA32_STAR[63:48] + 8`, ERETU establishes CS and SS in a standard configuration for CPL 3 in 64-bit mode:
 - CS:
 - The selector is set to the selector popped for CS.
 - The base address is set to 0. The limit is set to `FFFFFH` and the G bit is set to 1.
 - The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
 - The DPL is set to 3, the P and L bits are each set to 1, and the D bit is set to 0.
 - CS is made usable regardless of the value of the selector.
 - SS:
 - The selector is set to the selector popped for SS.
 - The base address is set to 0. The limit is set to `FFFFFH` and the G bit is set to 1.
 - The type is set to 3 (read/write accessed data) and the S bit is set to 1.

- The DPL is set to 3, and the P and B bits are each set to 1.
 - SS is made usable regardless of the value of the selector.
2. If the selector popped for CS is `IA32_STAR[63:48]` and the selector popped for SS is `IA32_STAR[63:48] + 8`, ERETU will establish CS and SS in a standard configuration for CPL s3 in compatibility mode:
- CS:
 - The selector is set to the selector popped for CS.
 - The base address is set to 0. The limit is set to `FFFFFH` and the G bit is set to 1.
 - The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
 - The DPL is set to 3, the P and D bits are set to 1, and the L bit is set to 0.
 - CS is made usable regardless of the value of the selector.
 - SS:
 - The selector is set to the selector popped for SS.
 - The base address is set to 0. The limit is set to `FFFFFH` and the G bit is set to 1.
 - The type is set to 3 (read/write accessed data) and the S bit is set to 1.
 - The DPL is set to 3, and the P and B bits are each set to 1.
 - SS is made usable regardless of the value of the selector.
3. Otherwise, the selectors popped for CS and SS are used to load descriptors from the GDT or an LDT as would be done by an execution of IRET. These descriptor loads may cause ERETU to fault as would occur with IRET.

The RIP of the return state is then checked, depending on the mode to which ERETU is returning:

- If ERETU is returning to 64-bit mode (either case #1 above, or case #3, where the descriptor loaded for CS sets the L bit), a #GP with a zero error code occurs if the return RIP is not paging canonical.
- If ERETU is returning to compatibility mode (either case #2 above, or case #3, where the descriptor loaded for CS clears the L bit), the upper 32 bits of the return RIP are cleared. Following this, a #GP with a zero error code occurs if the truncated RIP value exceeds the new CS segment limit (taking into account that segment's G bit). (Such a #GP will never occur for case #2, as it establishes a limit of `FFFFFFFFH`.)

8.4.2.2 Checking the Shadow-Stack Pointers

If user shadow stacks are enabled, the SSP of the return state is the value of the `IA32_PL3_SSP` MSR. If the return is to compatibility mode, a #GP occurs if `IA32_PL3_SSP[63:32]` are not all zero.

If supervisor shadow stacks are enabled, ERETU compares the current (supervisor) SSP value and the value of the `IA32_FRED_SSP0` MSR. If the values are not equal, a #CP occurs.

NOTE

Execution of ERETU does not access any shadow stack.

8.4.2.3 Establishing the Return State

After the stack operations described earlier, ERETU loads RIP, RFLAGS, RSP, CS, and SS with the values identified in Section 8.4.2.1. If the return is to compatibility mode, the upper 32 bits of each of RIP and RSP are cleared to zero. If user shadow stacks are enabled, SSP is loaded with the value for the return state as identified in Section 8.4.2.2 (from the `IA32_PL3_SSP` MSR).

ERETU swaps the value of the GS base address and that of the `IA32_KERNEL_GS_BASE` MSR.

Additional steps are performed based on the value of the popped SS field:

- If bit 17 of the field is 1 and ERETU would result in `RFLAGS.TF = 1`, a single-step trap will be pending upon completion of ERETU.¹
- If bit 18 of the field (above the selector) is 1, ERETU unblocks NMIs.

NOTE

Unlike IRET, ERETU does not make any of DS, ES, FS, or GS null if it is found to have DPL < 3. It is expected that a FRED-enabled operating system will return to CPL 3 (in compatibility mode) only when those segments all have DPL = 3.

8.4.2.4 Interactions Between ERETU and Other Instructions

ERETU provides the same instruction-ordering guarantees as SYSRET. Specifically, instructions following an execution of ERETU may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the ERETU have completed execution. (The later instructions may execute before data stored by the earlier instructions have become globally visible.)

Execution of the UMWAIT instruction will not wait (will not enter an implementation-dependent optimized state) if ERETU was executed before UMWAIT and after the most recent execution of the UMONITOR instruction.

8.5 FRED AND EXISTING INSTRUCTIONS

The Intel® 64 architecture defines numerous instructions that change the CPL. This section considers those instructions (and others) and describes how enabling FRED transitions affects either their operation or their usage. (Recall that FRED transitions are enabled if CR4.FRED = 1.)

- Section 8.5.1 identifies instructions that cannot be executed when FRED transitions are enabled.
- Section 8.5.2 considers far CALL, far JMP, far RET, and IRET. Enabling FRED transitions modifies the operation of these instructions. A FRED-enabled operating system cannot use them to change the CPL or to enter compatibility mode while CPL = 0.
- Section 8.5.3 discusses software interrupts and related instructions (INT n, INT3, INTO, and INT1). When FRED transitions are enabled, these instructions use FRED event delivery.
- Section 8.5.4 describes changes to the instructions SYSCALL and SYSENTER. When FRED transitions are enabled, these instructions also use FRED event delivery.
- Section 8.5.5 discusses interactions with the GETSEC instructions that enter or exit authenticated code execution mode.

8.5.1 Disallowed Instructions

When FRED transitions are enabled, execution of any of the following instructions causes an invalid-opcode exception (#UD): CLRSSBSY, SETSSBSY, SWAPGS, SYSEXIT, and SYSRET.

NOTE

CLRSSBSY, SETSSBSY, and SWAPGS account for anomalous situations that do not exist with FRED. In addition, CLRSSBSY and SETSSBSY operate on supervisor shadow-stack tokens, which are not used when FRED transitions are enabled.

8.5.2 Far CALL, IRET, Far JMP, and Far RET

Enabling FRED transitions disables the ability of certain existing instructions to change the CPL or to effect certain other mode changes.

The far CALL instruction provides a mechanism by which user software can change to more privilege. The IRET and far RET instructions allow returns to the calling context, and this can change the CPL. The operation of these instructions (as well as that of far JMP) is modified when FRED transitions are enabled.

1. If ERETU began execution with RFLAGS.TF = 1, there will always be a single-step debug exception pending after ERETU, regardless of the values on the stack for RFLAGS.TF and CS.

Far CALL can change CPL only if it references a call gate in the GDT or an LDT. (Far JMP can also reference a call gate, but an execution that does so cannot change the CPL.) When FRED transitions are enabled, any execution of far CALL or far JMP that references a call gate causes a general-protection exception (#GP). (The call-gate descriptor will not be modified.) The exception's error code will include the selector that referenced the call gate. This exception has the same priority as that of existing checks on the type of the target descriptor.

An execution of IRET or far RET changes the CPL if the RPL value of selector referencing the target code segment is greater than the CPL. When FRED transitions are enabled, such an execution causes #GP. (The code-segment descriptor will be accessed but not modified.) The exception's error code will include the selector that referenced the code segment. This exception has the same priority as that of the existing check that the RPL cannot be less than the CPL.

These restrictions imply that, when they are enabled, FRED transitions are the only ways to change the CPL and that it is impossible to change the CPL to 1 or to 2.

In addition, enabling FRED transitions disables the ability of software to enter compatibility mode while CPL = 0. An execution of far CALL, IRET, far JMP, or far RET causes a transition to compatibility mode if it loads CS from a code-segment descriptor in which the L bit is 0. (The L bit is bit 53 of the 64-bit descriptor.) When FRED transitions are enabled, such an execution with CPL = 0 causes #GP. The exception's error code will include the selector that referenced the code segment. This exception has the same priority as that of existing checks on the type of the target descriptor.

NOTE

The restrictions described in this section do not prevent use of far CALL, IRET, far JMP, or far RET to enter compatibility mode when the instruction is executed with CPL = 3 and does not reference a call gate. They can be used with CPL = 0 only if their execution does not change the CPL and stays in 64-bit mode.

8.5.3 Software Interrupts and Related Instructions

The Intel 64 architecture supports the following instructions that unprivileged software can use to invoke the operating system using the IDT:

- INT n (opcode CD followed by an immediate byte). There are 256 such software-interrupt instructions, one for each value n of the immediate byte (0–255).
- INT3 (opcode CC). This instruction generates a breakpoint exception (#BP) as a trap.
- INTO (opcode CE). If RFLAGS.OF = 1, this instruction generates an overflow exception (#OF) as a trap. If RFLAGS.OF = 0, this instruction does not generate an exception and control passes to the next instruction. Executing INTO in 64-bit mode causes an invalid-opcode exception (#UD), but the instruction can be used to generate #OF in compatibility mode.
- INT1 (opcode F1). This instruction generates a debug exception (#DB) as a trap. Hardware vendors may use INT1 for hardware debug.

When FRED transitions are enabled, an execution of any of these instructions (except for INTO if RFLAGS.OF = 0) results in FRED event delivery. Section 8.3.2.1 describes how the resulting FRED event delivery saves event information for these instructions (including the length of the instruction).

NOTE

IDT event delivery of INT n, INT3, and INTO checks the DPL field of the IDT gate and generates a general-protection exception (#GP) if it is less than the CPL. (Delivery of INT1 does not do this.) FRED event delivery does not use the IDT and thus does not perform this check. The event handlers of a FRED-enabled operating system should check the event type and vector to identify situations that would have resulted in a fault with IDT event delivery.

8.5.4 SYSCALL and SYSENTER

The Intel 64 architecture supports two instructions that unprivileged software can use to invoke the operating system without using the IDT: SYSCALL and SYSENTER.

When FRED transitions are enabled, the operation of SYSCALL is modified to use FRED event delivery (Section 8.3) in place of its existing operation. A FRED-enabled operating system should use ERETU (Section 8.4.2) instead of SYSRET to return after handling a system call invoked by an execution of SYSCALL with CPL = 3. (Recall that any execution of SYSRET causes #UD when FRED transitions are enabled. A FRED-enabled operating system will normally use ERETU to return from any event that occurred when CPL = 3.)

(Since the SYSRET instruction always changes the CPL to 3, use of SYSCALL has previously been effectively limited to CPL 3. When SYSCALL uses FRED event delivery, that instruction can be used effectively when CPL = 0. If this is done, a FRED-enabled operating system would naturally return with ERETS.)

The operation of SYSENTER is also modified to use FRED event delivery when FRED transitions are enabled. A FRED-enabled operating system should use ERETU instead of SYSEXIT to return after handling a system call invoked by an execution of SYSENTER with CPL = 3. (Recall that any execution of SYSEXIT causes #UD when FRED transitions are enabled.)

(Since the SYSEXIT instruction always changes the CPL to 3, use of SYSENTER has previously been effectively limited to CPL 3. When SYSENTER uses FRED event delivery, that instruction can be used effectively when CPL = 0. If this is done, a FRED-enabled operating system would naturally return with ERETS.)

8.5.5 GETSEC and Authenticated-Code Execution Mode

Software can invoke an authenticated-code module that operates in authenticated-code execution mode using the ENTERACCS and SENTER functions of the GETSEC instruction. The authenticated-code module leaves authenticated-code execution mode by executing the GETSEC function EXITAC.

After an authenticated-code module is invoked, events should not be delivered using FRED event delivery using a configuration that was established before entry to authenticated-code execution mode. Processors ensure that by disabling FRED transitions when entering authenticated-code execution mode. A subsequent execution of GETSEC[EXITAC] will re-enable FRED transitions if they had been enabled earlier.

To support this functionality, processors that support FRED transitions implement the features described in the remainder of this section.

A four-byte field called Cr4High is defined at offset 92 in the header of an authenticated-code module (ACM). This field is used by entry to and exit from authenticated-code execution mode.

Execution of either GETSEC[ENTERACCS] or GETSEC[SENDER] saves the current value of CR4[63:32] into the Cr4High field in the ACM header and then clears CR4.FRED to 0. This ensures that FRED event delivery will not be used in authenticated-code execution mode until the ACM has had the opportunity to configure and enable FRED.

If the logical processor entered authenticated-code execution mode using GETSEC[ENTERACCS], execution of GETSEC[EXITAC] reads the Cr4High field in the ACM header. If bit 0 of the Cr4High field in the ACM header (corresponding to CR4.FRED) is 1, a general-protection fault (#GP) occurs if GETSEC[EXITAC] is executed (and would thus end) outside IA-32e mode (IA32_EFER.LMA = 0). Otherwise, GETSEC[EXITAC] loads CR4.FRED with the value of that bit. (If the logical processor entered authenticated-code execution mode in some other manner, GETSEC[EXITAC] does not read the Cr4High field in the ACM header and does not modify CR4.)

8.6 LKGS: SUPPORT FOR MANAGING GS

64-bit operating systems and their applications use the GS segment for thread-local storage (TLS). Because the operating system and applications use the TLS at different addresses, they use different base addresses for that segment.

FRED transitions ensure that an operating system can always operate with its own GS base address:

- For events that occur while CPL = 3, FRED event delivery swaps the GS base address with the IA32_KERNEL_GS_BASE MSR.

FLEXIBLE RETURN AND EVENT DELIVERY (FRED)

- ERETU (the FRED transition that returns to CPL 3) also swaps the GS base address with the IA32_KERNEL_GS_BASE MSR.

An operating system can modify the GS base address of a user thread (e.g., as part of a context switch) by updating the IA32_KERNEL_GS_BASE MSR. However, the MOV GS instruction does not allow an operating system to modify other attributes of the GS segment without compromising its ability always to operate with its own GS base address. This is because that instruction updates the GS base address as well as those attributes.

The FRED architecture addresses this deficiency with the **LKGS** instruction (abbreviating “load into IA32_KERNEL_GS_BASE”). LKGS behaves like the MOV to GS instruction except that it loads the base address into the IA32_KERNEL_GS_BASE MSR instead of the GS segment’s descriptor cache.

Support for LKGS is enumerated with the feature flag CPUID.07H.01H:EAX.LKGS[18].

7. Updates to Chapter 10, Volume 3A

Change bars and violet 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:

- In Figure 10-2, the offset between 88 and 96 was corrected to 92.

This chapter describes the IA-32 architecture's task management facilities. These facilities are only available when the processor is running in protected mode.

This chapter focuses on 32-bit tasks and the 32-bit TSS structure. For information on 16-bit tasks and the 16-bit TSS structure, see Section 10.6, "16-Bit Task-State Segment (TSS)." For information specific to task management in 64-bit mode, see Section 10.7, "Task Management in 64-bit Mode."

10.1 TASK MANAGEMENT OVERVIEW

A task is a unit of work that a processor can dispatch, execute, and suspend. It can be used to execute a program, a task or process, an operating-system service utility, an interrupt or exception handler, or a kernel or executive utility.

The IA-32 architecture provides a mechanism for saving the state of a task, for dispatching tasks for execution, and for switching from one task to another. When operating in protected mode, all processor execution takes place from within a task. Even simple systems must define at least one task. More complex systems can use the processor's task management facilities to support multitasking applications.

10.1.1 Task Structure

A task is made up of two parts: a task execution space and a task-state segment (TSS). The task execution space consists of a code segment, a stack segment, and one or more data segments (see Figure 10-1). If an operating system or executive uses the processor's privilege-level protection mechanism, the task execution space also provides a separate stack for each privilege level.

The TSS specifies the segments that make up the task execution space and provides a storage place for task state information. In multitasking systems, the TSS also provides a mechanism for linking tasks.

A task is identified by the segment selector for its TSS. When a task is loaded into the processor for execution, the segment selector, base address, limit, and segment descriptor attributes for the TSS are loaded into the task register (see Section 2.4.4, "Task Register (TR)").

If paging is implemented for the task, the base address of the page directory used by the task is loaded into control register CR3.

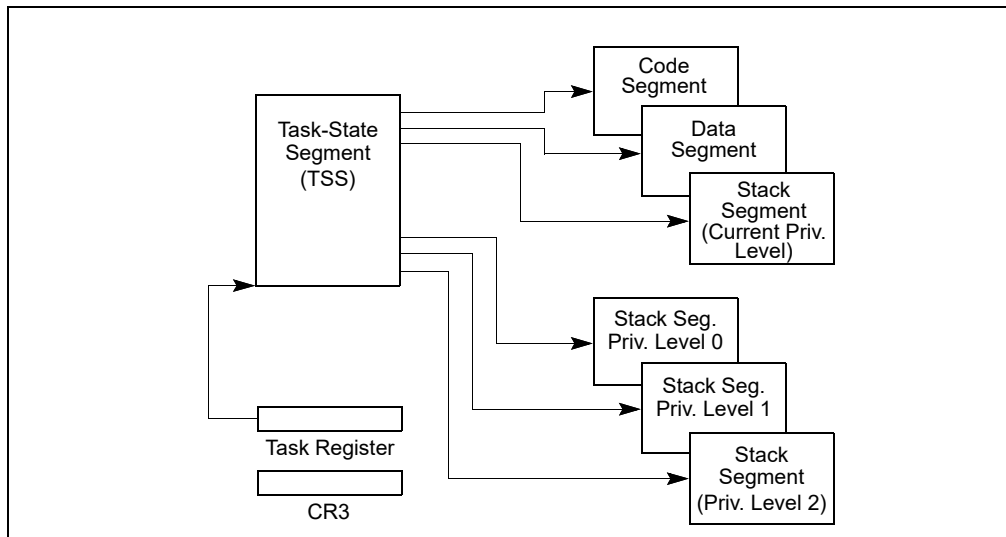


Figure 10-1. Structure of a Task

10.1.2 Task State

The following items define the state of the currently executing task:

- The task's current execution space, defined by the segment selectors in the segment registers (CS, DS, SS, ES, FS, and GS).
- The state of the general-purpose registers.
- The state of the EFLAGS register.
- The state of the EIP register.
- The state of control register CR3.
- The state of the task register.
- The state of the LDTR register.
- The I/O map base address and I/O map (contained in the TSS).
- Stack pointers to the privilege 0, 1, and 2 stacks (contained in the TSS).
- Link to previously executed task (contained in the TSS).
- The state of the shadow stack pointer (SSP).

Prior to dispatching a task, all of these items are contained in the task's TSS, except the state of the task register. Also, the complete contents of the LDTR register are not contained in the TSS, only the segment selector for the LDT.

10.1.3 Executing a Task

Software or the processor can dispatch a task for execution in one of the following ways:

- A explicit call to a task with the CALL instruction.
- A explicit jump to a task with the JMP instruction.
- An implicit call (by the processor) to an interrupt-handler task.
- An implicit call to an exception-handler task.
- A return (initiated with an IRET instruction) when the NT flag in the EFLAGS register is set.

All of these methods for dispatching a task identify the task to be dispatched with a segment selector that points to a task gate or the TSS for the task. When dispatching a task with a CALL or JMP instruction, the selector in the instruction may select the TSS directly or a task gate that holds the selector for the TSS. When dispatching a task

to handle an interrupt or exception, the IDT entry for the interrupt or exception must contain a task gate that holds the selector for the interrupt- or exception-handler TSS.

When a task is dispatched for execution, a task switch occurs between the currently running task and the dispatched task. During a task switch, the execution environment of the currently executing task (called the task's state or **context**) is saved in its TSS and execution of the task is suspended. The context for the dispatched task is then loaded into the processor and execution of that task begins with the instruction pointed to by the newly loaded EIP register. If the task has not been run since the system was last initialized, the EIP will point to the first instruction of the task's code; otherwise, it will point to the next instruction after the last instruction that the task executed when it was last active.

If the currently executing task (the calling task) called the task being dispatched (the called task), the TSS segment selector for the calling task is stored in the TSS of the called task to provide a link back to the calling task.

For all IA-32 processors, tasks are not recursive. A task cannot call or jump to itself.

Interrupts and exceptions can be handled with a task switch to a handler task. Here, the processor performs a task switch to handle the interrupt or exception and automatically switches back to the interrupted task upon returning from the interrupt-handler task or exception-handler task. This mechanism can also handle interrupts that occur during interrupt tasks.

As part of a task switch, the processor can also switch to another LDT, allowing each task to have a different logical-to-physical address mapping for LDT-based segments. The page-directory base register (CR3) also is reloaded on a task switch, allowing each task to have its own set of page tables. These protection facilities help isolate tasks and prevent them from interfering with one another.

If protection mechanisms are not used, the processor provides no protection between tasks. This is true even with operating systems that use multiple privilege levels for protection. A task running at privilege level 3 that uses the same LDT and page tables as other privilege-level-3 tasks can access code and corrupt data and the stack of other tasks.

Use of task management facilities for handling multitasking applications is optional. Multitasking can be handled in software, with each software defined task executed in the context of a single IA-32 architecture task.

If shadow stack is enabled, then the SSP of the task is located at the 4 bytes at offset 104 in the 32-bit TSS and is used by the processor to establish the SSP when a task switch occurs from a task associated with this TSS. Note that the processor does not write the SSP of the task initiating the task switch to the TSS of that task, and instead the SSP of the previous task is pushed onto the shadow stack of the new task.

10.2 TASK MANAGEMENT DATA STRUCTURES

The processor defines five data structures for handling task-related activities:

- Task-state segment (TSS).
- Task-gate descriptor.
- TSS descriptor.
- Task register.
- NT flag in the EFLAGS register.

When operating in protected mode, a TSS and TSS descriptor must be created for at least one task, and the segment selector for the TSS must be loaded into the task register (using the LTR instruction).

10.2.1 Task-State Segment (TSS)

The processor state information needed to restore a task is saved in a system segment called the task-state segment (TSS). Figure 10-2 shows the format of a TSS for tasks designed for 32-bit CPUs. The fields of a TSS are divided into two main categories: dynamic fields and static fields.

For information about 16-bit Intel 286 processor task structures, see Section 10.6, "16-Bit Task-State Segment (TSS)." For information about 64-bit mode task structures, see Section 10.7, "Task Management in 64-bit Mode."

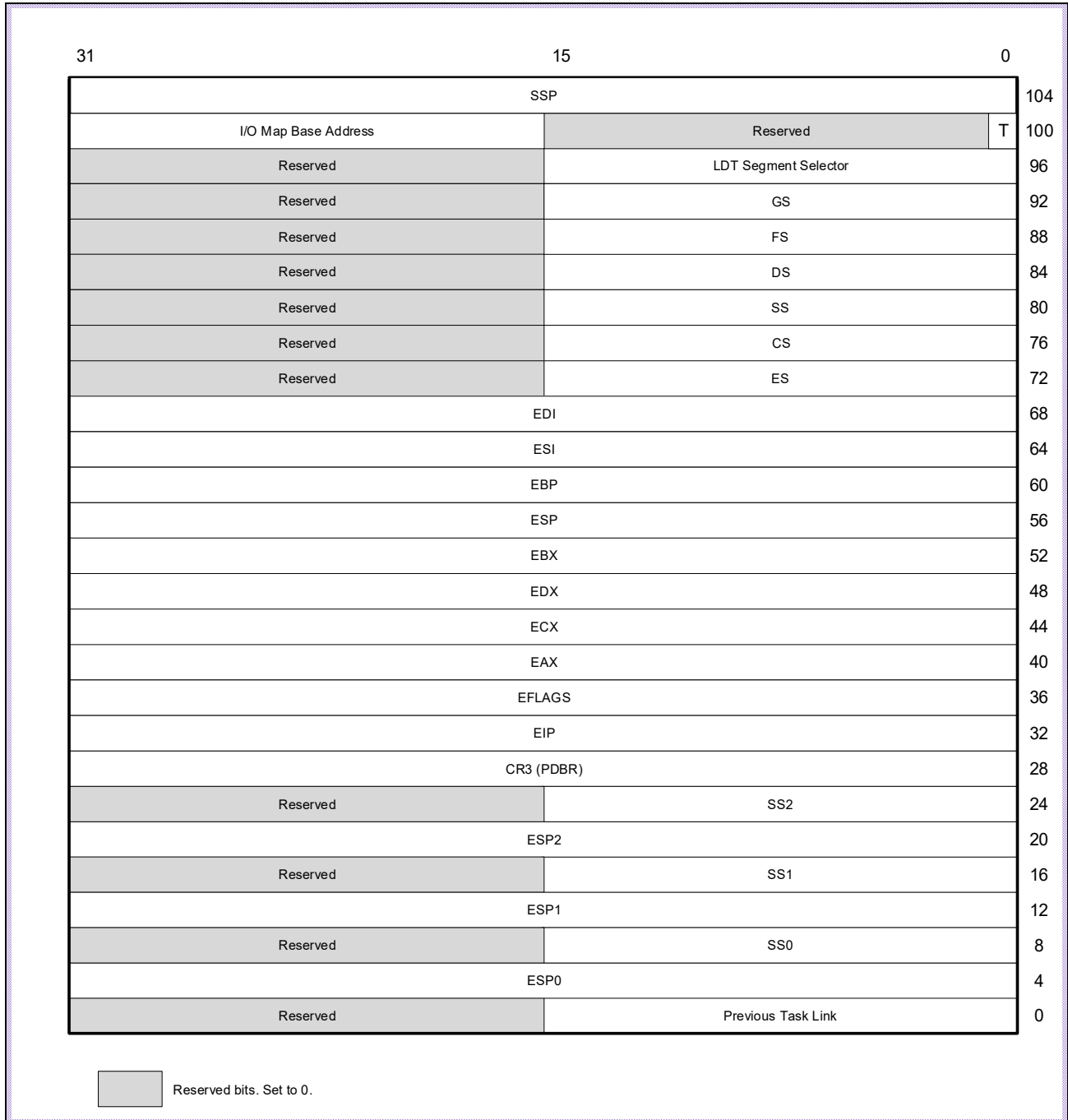


Figure 10-2. 32-Bit Task-State Segment (TSS)

The processor updates dynamic fields when a task is suspended during a task switch. The following are dynamic fields:

- **General-purpose register fields** — State of the EAX, ECX, EDX, EBX, ESP, EBP, ESI, and EDI registers prior to the task switch.
- **Segment selector fields** — Segment selectors stored in the ES, CS, SS, DS, FS, and GS registers prior to the task switch.

- **EFLAGS register field** — State of the EFLAGS register prior to the task switch.
- **EIP (instruction pointer) field** — State of the EIP register prior to the task switch.
- **Previous task link field** — Contains the segment selector for the TSS of the previous task (updated on a task switch that was initiated by a call, interrupt, or exception). This field (which is sometimes called the back link field) permits a task switch back to the previous task by using the IRET instruction.

The processor reads the static fields, but does not normally change them. These fields are set up when a task is created. The following are static fields:

- **LDT segment selector field** — Contains the segment selector for the task's LDT.
- **CR3 control register field** — Contains the base physical address of the page directory to be used by the task. Control register CR3 is also known as the page-directory base register (PDBR).
- **Privilege level-0, -1, and -2 stack pointer fields** — These stack pointers consist of a logical address made up of the segment selector for the stack segment (SS0, SS1, and SS2) and an offset into the stack (ESP0, ESP1, and ESP2). Note that the values in these fields are static for a particular task; whereas, the SS and ESP values will change if stack switching occurs within the task.
- **T (debug trap) flag (byte 100, bit 0)** — When set, the T flag causes the processor to raise a debug exception when a task switch to this task occurs (see Section 20.3.1.5, "Task-Switch Exception Condition").
- **I/O map base address field** — Contains a 16-bit offset from the base of the TSS to the I/O permission bit map and interrupt redirection bitmap. When present, these maps are stored in the TSS at higher addresses. The I/O map base address points to the beginning of the I/O permission bit map and the end of the interrupt redirection bit map. See Chapter 20, "Input/Output," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for more information about the I/O permission bit map. See Section 23.3, "Interrupt and Exception Handling in Virtual-8086 Mode," for a detailed description of the interrupt redirection bit map.
- **Shadow Stack Pointer (SSP)** — Contains task's shadow stack pointer. The shadow stack of the task should have a supervisor shadow stack token at the address pointed to by the task SSP (offset 104). This token will be verified and made busy when switching to that shadow stack using a CALL/JMP instruction, and made free when switching out of that task using an IRET instruction.

If paging is used:

- Pages corresponding to the previous task's TSS, the current task's TSS, and the descriptor table entries for each all should be marked as read/write.
- Task switches are carried out faster if the pages containing these structures are present in memory before the task switch is initiated.

10.2.2 TSS Descriptor

The TSS, like all other segments, is defined by a segment descriptor. Figure 10-3 shows the format of a TSS descriptor. TSS descriptors may only be placed in the GDT; they cannot be placed in an LDT or the IDT.

An attempt to access a TSS using a segment selector with its TI flag set (which indicates the current LDT) causes a general-protection exception (#GP) to be generated during CALLs and JMPs; it causes an invalid TSS exception (#TS) during IRETs. A general-protection exception is also generated if an attempt is made to load a segment selector for a TSS into a segment register.

The busy flag (B) in the type field indicates whether the task is busy. A busy task is currently running or suspended. A type field with a value of 1001B indicates an inactive task; a value of 1011B indicates a busy task. Tasks are not recursive. The processor uses the busy flag to detect an attempt to call a task whose execution has been interrupted. To ensure that there is only one busy flag is associated with a task, each TSS should have only one TSS descriptor that points to it.

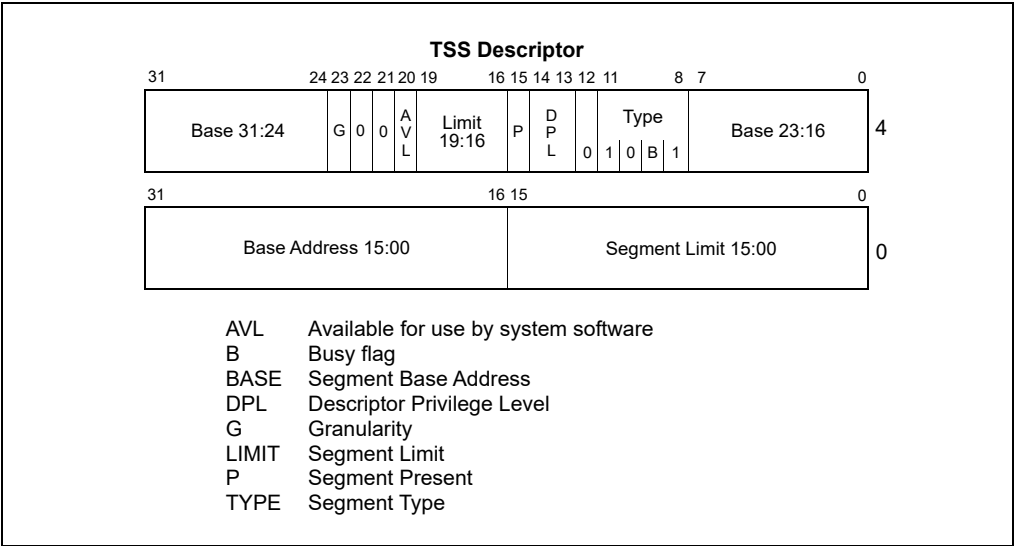


Figure 10-3. TSS Descriptor

The base, limit, and DPL fields and the granularity and present flags have functions similar to their use in data-segment descriptors (see Section 3.4.5, “Segment Descriptors”). When the G flag is 0 in a TSS descriptor for a 32-bit TSS, the limit field must have a value equal to or greater than 67H, one byte less than the minimum size of a TSS. Attempting to switch to a task whose TSS descriptor has a limit less than 67H generates an invalid-TSS exception (#TS). A larger limit is required if an I/O permission bit map is included or if the operating system stores additional data. The processor does not check for a limit greater than 67H on a task switch; however, it does check when accessing the I/O permission bit map or interrupt redirection bit map.

Any program or procedure with access to a TSS descriptor (that is, whose CPL is numerically equal to or less than the DPL of the TSS descriptor) can dispatch the task with a call or a jump.

In most systems, the DPLs of TSS descriptors are set to values less than 3, so that only privileged software can perform task switching. However, in multitasking applications, DPLs for some TSS descriptors may be set to 3 to allow task switching at the application (or user) privilege level.

10.2.3 TSS Descriptor in 64-bit mode

In 64-bit mode, task switching is not supported, but TSS descriptors still exist. The format of a 64-bit TSS is described in Section 10.7.

In 64-bit mode, the TSS descriptor is expanded to 16 bytes (see Figure 10-4). This expansion also applies to an LDT descriptor in 64-bit mode. Table 3-2 provides the encoding information for the segment type field.

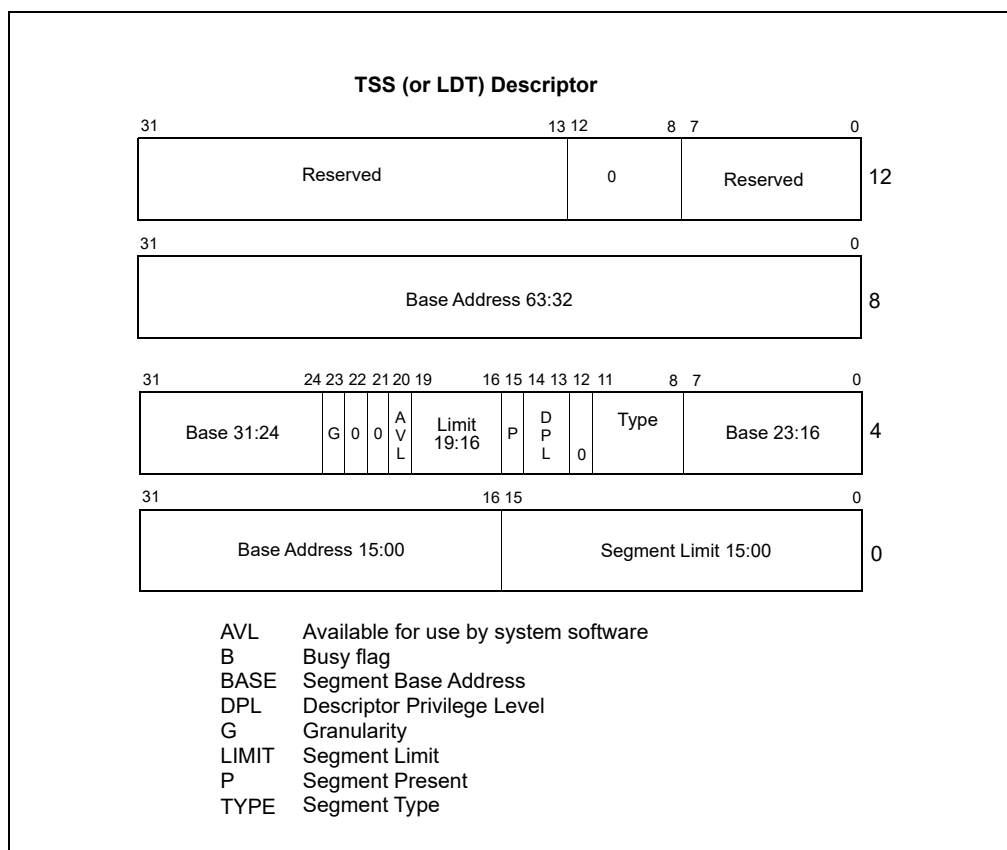


Figure 10-4. Format of TSS and LDT Descriptors in 64-bit Mode

10.2.4 Task Register

The task register holds the 16-bit segment selector and the entire segment descriptor (32-bit base address (64 bits in IA-32e mode), 16-bit segment limit, and descriptor attributes) for the TSS of the current task (see Figure 2-6). This information is copied from the TSS descriptor in the GDT for the current task. Figure 10-5 shows the path the processor uses to access the TSS (using the information in the task register).

The task register has a visible part (that can be read and changed by software) and an invisible part (maintained by the processor and is inaccessible by software). The segment selector in the visible portion points to a TSS descriptor in the GDT. The processor uses the invisible portion of the task register to cache the segment descriptor for the TSS. Caching these values in a register makes execution of the task more efficient. The LTR (load task register) and STR (store task register) instructions load and read the visible portion of the task register:

The LTR instruction loads a segment selector (source operand) into the task register that points to a TSS descriptor in the GDT. It then loads the invisible portion of the task register with information from the TSS descriptor. LTR is a privileged instruction that may be executed only when the CPL is 0. It's used during system initialization to put an initial value in the task register. Afterwards, the contents of the task register are changed implicitly when a task switch occurs.

The STR (store task register) instruction stores the visible portion of the task register in a general-purpose register or memory. This instruction can be executed by code running at any privilege level in order to identify the currently running task. However, it is normally used only by operating system software. (If CR4.UMIP = 1, STR can be executed only when CPL = 0.)

On power up or reset of the processor, segment selector and base address are set to the default value of 0; the limit is set to FFFFH.

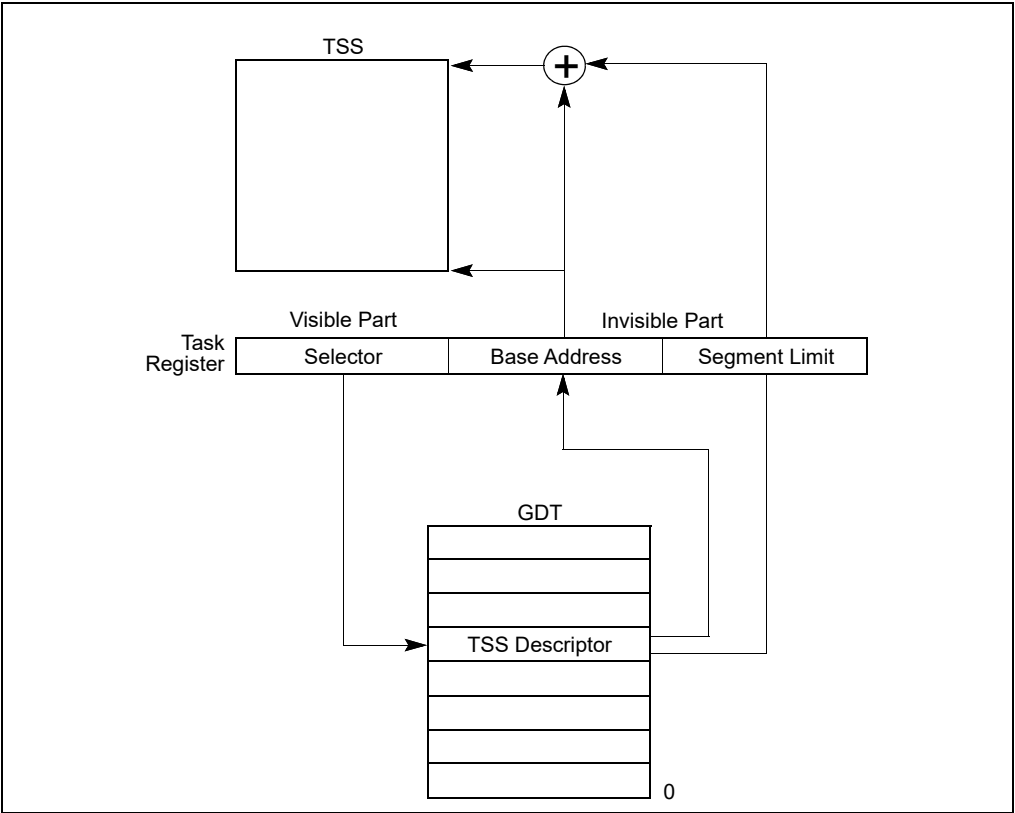


Figure 10-5. Task Register

10.2.5 Task-Gate Descriptor

A task-gate descriptor provides an indirect, protected reference to a task (see Figure 10-6). It can be placed in the GDT, an LDT, or the IDT. The TSS segment selector field in a task-gate descriptor points to a TSS descriptor in the GDT. The RPL in this segment selector is not used.

The DPL of a task-gate descriptor controls access to the TSS descriptor during a task switch. When a program or procedure makes a call or jump to a task through a task gate, the CPL and the RPL field of the gate selector pointing to the task gate must be less than or equal to the DPL of the task-gate descriptor. Note that when a task gate is used, the DPL of the destination TSS descriptor is not used.

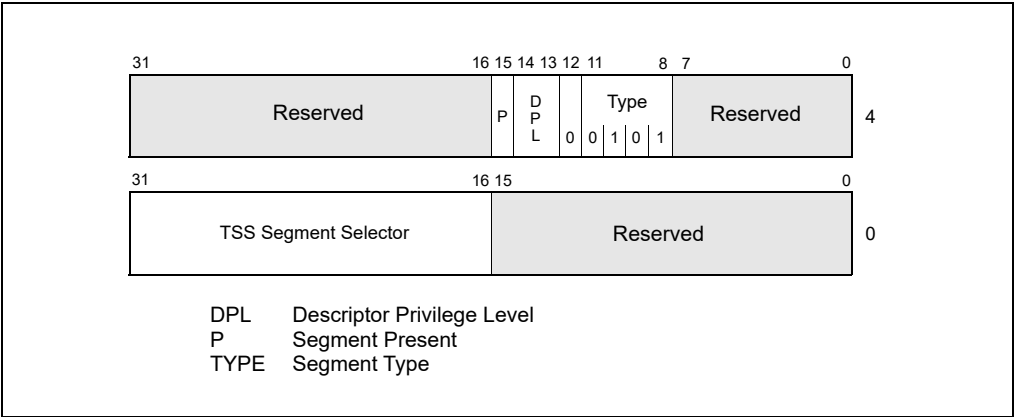


Figure 10-6. Task-Gate Descriptor

A task can be accessed either through a task-gate descriptor or a TSS descriptor. Both of these structures satisfy the following needs:

- **Need for a task to have only one busy flag** — Because the busy flag for a task is stored in the TSS descriptor, each task should have only one TSS descriptor. There may, however, be several task gates that reference the same TSS descriptor.
- **Need to provide selective access to tasks** — Task gates fill this need, because they can reside in an LDT and can have a DPL that is different from the TSS descriptor's DPL. A program or procedure that does not have sufficient privilege to access the TSS descriptor for a task in the GDT (which usually has a DPL of 0) may be allowed access to the task through a task gate with a higher DPL. Task gates give the operating system greater latitude for limiting access to specific tasks.
- **Need for an interrupt or exception to be handled by an independent task** — Task gates may also reside in the IDT, which allows interrupts and exceptions to be handled by handler tasks. When an interrupt or exception vector points to a task gate, the processor switches to the specified task.

Figure 10-7 illustrates how a task gate in an LDT, a task gate in the GDT, and a task gate in the IDT can all point to the same task.

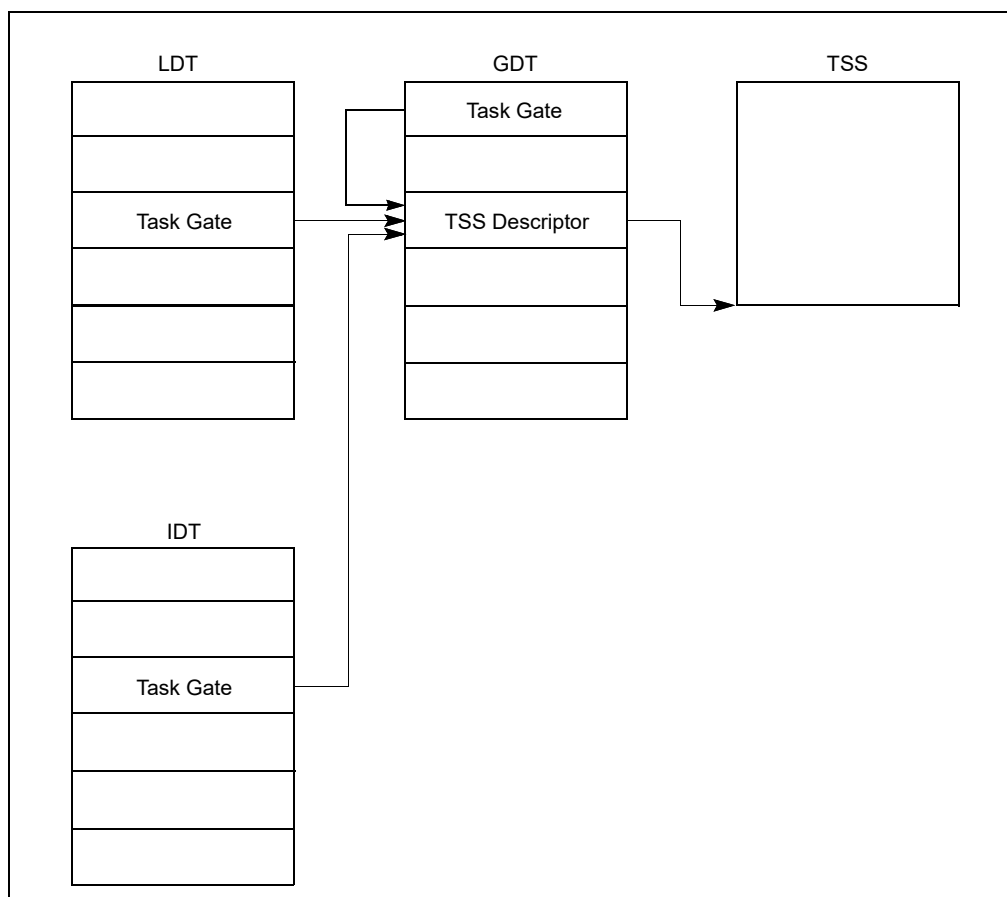


Figure 10-7. Task Gates Referencing the Same Task

10.3 TASK SWITCHING

The processor transfers execution to another task in one of four cases:

- The current program, task, or procedure executes a JMP or CALL instruction to a TSS descriptor in the GDT.
- The current program, task, or procedure executes a JMP or CALL instruction to a task-gate descriptor in the GDT or the current LDT.

- An interrupt or exception vector points to a task-gate descriptor in the IDT.
- The current task executes an IRET when the NT flag in the EFLAGS register is set.

JMP, CALL, and IRET instructions, as well as interrupts and exceptions, are all mechanisms for redirecting a program. The referencing of a TSS descriptor or a task gate (when calling or jumping to a task) or the state of the NT flag (when executing an IRET instruction) determines whether a task switch occurs.

The processor performs the following operations when switching to a new task:

1. Obtains the TSS segment selector for the new task as the operand of the JMP or CALL instruction, from a task gate, or from the previous task link field (for a task switch initiated with an IRET instruction).
2. Checks that the current (old) task is allowed to switch to the new task. Data-access privilege rules apply to JMP and CALL instructions. The CPL of the current (old) task and the RPL of the segment selector for the new task must be less than or equal to the DPL of the TSS descriptor or task gate being referenced. Exceptions, interrupts (except for those identified in the next sentence), and the IRET and INT1 instructions are permitted to switch tasks regardless of the DPL of the destination task-gate or TSS descriptor. For interrupts generated by the INT n , INT3, and INTO instructions, the DPL is checked and a general-protection exception (#GP) results if it is less than the CPL.¹
3. Checks that the TSS descriptor of the new task is marked present and has a valid limit (greater than or equal to 67H). If the task switch was initiated by IRET and shadow stacks are enabled at the current CPL, then the SSP must be aligned to 8 bytes, else a #TS(current task TSS) fault is generated. If CR4.CET is 1, then the TSS must be a 32 bit TSS and the limit of the new task's TSS must be greater than or equal to 107 bytes, else a #TS(new task TSS) fault is generated.
4. Checks that the new task is available (call, jump, exception, or interrupt) or busy (IRET return).
5. Checks that the current (old) TSS, new TSS, and all segment descriptors used in the task switch are paged into system memory.
6. Saves the state of the current (old) task in the current task's TSS. The processor finds the base address of the current TSS in the task register and then copies the states of the following registers into the current TSS: all the general-purpose registers, segment selectors from the segment registers, the temporarily saved image of the EFLAGS register, and the instruction pointer register (EIP).
7. Loads the task register with the segment selector and descriptor for the new task's TSS.
8. If CET is enabled, the processor performs following shadow stack actions:

Read CS of new task from new task TSS

Read EFLAGS of new task from new task TSS

IF EFLAGS.VM = 1

THEN

new task CPL = 3;

ELSE

new task CPL = CS.RPL;

FI;

pushCsLipSsp = 0

IF task switch was initiated by CALL instruction, exception or interrupt

IF shadow stack enabled at current CPL

IF new task CPL < CPL and current task CPL = 3

THEN

IA32_PL3_SSP = SSP (* user → supervisor *)

ELSE

pushCsLipSsp = 1 (* no privilege change; supv → supv; supv → user *) tempSSP = SSP

1. The INT1 has opcode F1; the INT n instruction with $n=1$ has opcode CD 01.

```

        tempSsLIP = CSBASE + EIP
        tempSsCS = CS
    FI;
FI;
verifyCsLIP = 0
IF task switch was initiated by IRET
    IF shadow stacks enabled at current CPL
        IF (CPL of new Task = CPL of current Task) OR
            (CPL of new Task < 3 AND CPL of current Task < 3) OR
            (CPL of new Task < 3 AND CPL of current task = 3)
            (* no privilege change or supervisor → supervisor or user → supervisor IRET *)
            tempSsCS = shadow_stack_load 8 bytes from SSP+16;
            tempSsLIP = shadow_stack_load 8 bytes from SSP+8;
            tempSSP = shadow_stack_load 8 bytes from SSP;
            SSP = SSP + 24;
            verifyCsLIP = 1
        FI;
        // Clear busy flag on current shadow stack
        IF ( SSP & 0x07 == 0 )                (* SSP must be aligned to 8B *)
            THEN
                expected_token_value = (SSP & ~0x07) | BUSY_BIT; (* busy - bit 0 - must be set*)
                new_token_value      = SSP                        (* clear the busy bit *)
                shadow_stack_lock_cmpxchg8b(SSP, new_token_value, expected_token_value)
            FI;
            SSP = 0
        FI;
    FI;
FI;

```

9. The TSS state is loaded into the processor. This includes the LDTR register, the PDBR (control register CR3), the EFLAGS register, the EIP register, the general-purpose registers, and the segment selectors. A fault during the load of this state may corrupt architectural state. (If paging is not enabled, a PDBR value is read from the new task's TSS, but it is not loaded into CR3.)
10. If the task switch was initiated with a JMP or IRET instruction, the processor clears the busy (B) flag in the current (old) task's TSS descriptor; if initiated with a CALL instruction, an exception, or an interrupt: the busy (B) flag is left set. (See Table 10-2.)
11. If the task switch was initiated with an IRET instruction, the processor clears the NT flag in a temporarily saved image of the EFLAGS register; if initiated with a CALL or JMP instruction, an exception, or an interrupt, the NT flag is left unchanged in the saved EFLAGS image.
12. If the task switch was initiated with a CALL instruction, an exception, or an interrupt, the processor will set the NT flag in the EFLAGS loaded from the new task. If initiated with an IRET instruction or JMP instruction, the NT flag will reflect the state of NT in the EFLAGS loaded from the new task (see Table 10-2).
13. If the task switch was initiated with a CALL instruction, JMP instruction, an exception, or an interrupt, the processor sets the busy (B) flag in the new task's TSS descriptor; if initiated with an IRET instruction, the busy (B) flag is left set.
14. The descriptors associated with the segment selectors are loaded and qualified. Any errors associated with this loading and qualification occur in the context of the new task and may corrupt architectural state.

15. If CET is enabled, the processor performs following shadow stack actions:

IF shadow stack enabled at current CPL OR indirect branch tracking at current CPL

THEN

IF EFLAGS.VM = 1

THEN #TSS(new-Task-TSS); FI;

FI;

IF shadow stack enabled at current CPL

IF task switch initiated by CALL instruction, JMP instruction, interrupt or exception (* switch stack *)

new_SSP ← Load the 4 byte from offset 104 in the TSS

// Verify new SSP to be legal

IF new_SSP & 0x07 != 0

THEN #TSS(New-Task-TSS); FI;

expected_token_value = SSP; (* busy - bit 0 - must be clear *)

new_token_value = SSP | BUSY_BIT (* set the busy bit - bit 0*)

IF shadow_stack_lock_cmpxchg8b(SSP, new_token_value,
expected_token_value) != expected_token_value

THEN #TSS(New-Task-TSS); FI;

SSP = new_SSP

IF pushCsLipSsp = 1 (* call, int, exception from user → user or supv → supv or supv → user *)

Push tempSsCS, tempSsLip, tempSsSSP on shadow stack using 8B pushes¹

FI;

FI;

FI;

IF task switch initiated by IRET

IF verifyCsLIP = 1

(* do 64 bit comparisons; CS zero padded to 64 bit; CSBASE+EIP zero padded to 64 bit *)

IF tempSsCS and tempSsLIP do not match CS and CSBASE+EIP

THEN #CP(FAR-RET/IRET); FI;

FI;

IF ShadowStackEnabled(CPL)

THEN

IF (verifyCsLIP == 0) tempSSP = IA32_PL3_SSP;

IF tempSSP & 0x03 != 0 THEN #CP(FAR-RET/IRET) // verify aligned to 4 bytes

IF tempSSP[63:32] != 0 THEN # CP(FAR-RET/IRET)

SSP = tempSSP

FI;

FI;

IF EndbranchEnabled(CPL)

IF task switch initiated by CALL instruction, JMP instruction, interrupt or exception

IF CPL = 3

THEN

IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH

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

```

        IA32_U_CET.SUPPRESS = 0
    ELSE
        IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
        IA32_S_CET.SUPPRESS = 0
    FI;
FI;
FI;

```

16. Begins executing the new task. (To an exception handler, the first instruction of the new task appears not to have been executed.)

NOTES

If all checks and saves have been carried out successfully, the processor commits to the task switch. If an unrecoverable error occurs in steps 1 through 8, the processor does not complete the task switch and ensures that the processor is returned to its state prior to the execution of the instruction that initiated the task switch.

If an unrecoverable error occurs in step 9, architectural state may be corrupted, but an attempt will be made to handle the error in the prior execution environment. If an unrecoverable error occurs after the commit point (in step 13), the processor completes the task switch (without performing additional access and segment availability checks) and generates the appropriate exception prior to beginning execution of the new task.

If exceptions occur after the commit point, the exception handler must finish the task switch itself before allowing the processor to begin executing the new task. See Chapter 7, “Interrupt 10—Invalid TSS Exception (#TS),” for more information about the affect of exceptions on a task when they occur after the commit point of a task switch.

The state of the currently executing task is always saved when a successful task switch occurs. If the task is resumed, execution starts with the instruction pointed to by the saved EIP value, and the registers are restored to the values they held when the task was suspended.

When switching tasks, the privilege level of the new task does not inherit its privilege level from the suspended task. The new task begins executing at the privilege level specified in the CPL field of the CS register, which is loaded from the TSS. Because tasks are isolated by their separate address spaces and TSSs and because privilege rules control access to a TSS, software does not need to perform explicit privilege checks on a task switch.

Table 10-1 shows the exception conditions that the processor checks for when switching tasks. It also shows the exception that is generated for each check if an error is detected and the segment that the error code references. (The order of the checks in the table is the order used in the P6 family processors. The exact order is model specific and may be different for other IA-32 processors.) Exception handlers designed to handle these exceptions may be subject to recursive calls if they attempt to reload the segment selector that generated the exception. The cause of the exception (or the first of multiple causes) should be fixed before reloading the selector.

Table 10-1. Exception Conditions Checked During a Task Switch

Condition Checked	Exception ¹	Error Code Reference ²
Segment selector for a TSS descriptor references the GDT and is within the limits of the table.	#GP	New Task's TSS
P bit is set in TSS descriptor.	#TS (for IRET)	
TSS descriptor is not busy (for task switch initiated by a call, interrupt, or exception).	#NP	New Task's TSS
TSS descriptor is not busy (for task switch initiated by an IRET instruction).	#GP (for JMP, CALL, INT)	Task's back-link TSS
TSS segment limit greater than or equal to 108 (for 32-bit TSS) or 44 (for 16-bit TSS).	#TS (for IRET)	New Task's TSS
	#TS	New Task's TSS

Table 10-1. Exception Conditions Checked During a Task Switch (Contd.)

Condition Checked	Exception ¹	Error Code Reference ²
TSS segment limit greater than or equal to 108 (for 32-bit TSS) if CR4.CET = 1. ³	#TS	New Task's TSS
If shadow stack enabled and SSP not aligned to 8 bytes (for task switch initiated by an IRET instruction). ³	#TS	Current Task's TSS
Registers are loaded from the values in the TSS.		
LDT segment selector of new task is valid ⁴ .	#TS	New Task's LDT
If code segment is non-conforming, its DPL should equal its RPL.	#TS	New Code Segment
If code segment is conforming, its DPL should be less than or equal to its RPL.	#TS	New Code Segment
SS segment selector is valid ² .	#TS	New Stack Segment
P bit is set in stack segment descriptor.	#SS	New Stack Segment
Stack segment DPL should equal CPL.	#TS	New stack segment
P bit is set in new task's LDT descriptor.	#TS	New Task's LDT
CS segment selector is valid ⁴ .	#TS	New Code Segment
P bit is set in code segment descriptor.	#NP	New Code Segment
Stack segment DPL should equal its RPL.	#TS	New Stack Segment
DS, ES, FS, and GS segment selectors are valid ⁴ .	#TS	New Data Segment
DS, ES, FS, and GS segments are readable.	#TS	New Data Segment
P bits are set in descriptors of DS, ES, FS, and GS segments.	#NP	New Data Segment
DS, ES, FS, and GS segment DPL greater than or equal to CPL (unless these are conforming segments).	#TS	New Data Segment
Shadow Stack Pointer in a task not aligned to 8 bytes (for task switch initiated by a call, interrupt, or exception). ³	#TS	New Task's TSS
If EFLAGS.VM=1 and shadow stacks are enabled. ³	#TS	New Task's TSS
Supervisor Shadow Stack Token verification failures (for task switch initiated by a call, interrupt, jump, or exception): ³	#TS	New Task's TSS
- Busy bit already set.		
- Address in Shadow stack token does not match SSP value from TSS.		
If task switch initiated by IRET, CS and LIP stored on old task shadow stack does not match CS and LIP of new task. ³	#CP	FAR-RET/IRET
If task switch 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) fails the following checks: ³	#CP	FAR-RET/IRET
- Not aligned to 4 bytes.		
- Is beyond 4G.		

NOTES:

1. #NP is segment-not-present exception, #GP is general-protection exception, #TS is invalid-TSS exception, and #SS is stack-fault exception.
2. The error code contains an index to the segment descriptor referenced in this column.
3. Valid when CET is enabled.
4. A segment selector is valid if it is in a compatible type of table (GDT or LDT), occupies an address within the table's segment limit, and refers to a compatible type of descriptor (for example, a segment selector in the CS register only is valid when it points to a code-segment descriptor).

The TS (task switched) flag in the control register CR0 is set every time a task switch occurs. System software uses the TS flag to coordinate the actions of floating-point unit when generating floating-point exceptions with the rest of the processor. The TS flag indicates that the context of the floating-point unit may be different from that of the current task. See Section 2.5, "Control Registers," for a detailed description of the function and use of the TS flag.

10.4 TASK LINKING

The previous task link field of the TSS (sometimes called the “backlink”) and the NT flag in the EFLAGS register are used to return execution to the previous task. EFLAGS.NT = 1 indicates that the currently executing task is nested within the execution of another task.

When a CALL instruction, an interrupt, or an exception causes a task switch: the processor copies the segment selector for the current TSS to the previous task link field of the TSS for the new task; it then sets EFLAGS.NT = 1. If software uses an IRET instruction to suspend the new task, the processor checks for EFLAGS.NT = 1; it then uses the value in the previous task link field to return to the previous task. See Figures 10-8.

When a JMP instruction causes a task switch, the new task is not nested. The previous task link field is not used and EFLAGS.NT = 0. Use a JMP instruction to dispatch a new task when nesting is not desired.

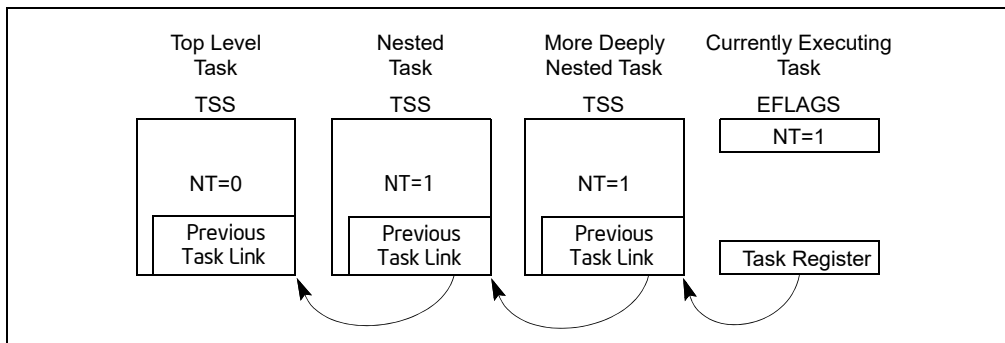


Figure 10-8. Nested Tasks

Table 10-2 shows the busy flag (in the TSS segment descriptor), the NT flag, the previous task link field, and TS flag (in control register CR0) during a task switch.

The NT flag may be modified by software executing at any privilege level. It is possible for a program to set the NT flag and execute an IRET instruction. This might randomly invoke the task specified in the previous link field of the current task's TSS. To keep such spurious task switches from succeeding, the operating system should initialize the previous task link field in every TSS that it creates to 0.

Table 10-2. Effect of a Task Switch on Busy Flag, NT Flag, Previous Task Link Field, and TS Flag

Flag or Field	Effect of JMP instruction	Effect of CALL Instruction or Interrupt	Effect of IRET Instruction
Busy (B) flag of new task.	Flag is set. Must have been clear before.	Flag is set. Must have been clear before.	No change. Must have been set.
Busy flag of old task.	Flag is cleared.	No change. Flag is currently set.	Flag is cleared.
NT flag of new task.	Set to value from TSS of new task.	Flag is set.	Set to value from TSS of new task.
NT flag of old task.	No change.	No change.	Flag is cleared.
Previous task link field of new task.	No change.	Loaded with selector for old task's TSS.	No change.
Previous task link field of old task.	No change.	No change.	No change.
TS flag in control register CR0.	Flag is set.	Flag is set.	Flag is set.

10.4.1 Use of Busy Flag To Prevent Recursive Task Switching

A TSS allows only one context to be saved for a task; therefore, once a task is called (dispatched), a recursive (or re-entrant) call to the task would cause the current state of the task to be lost. The busy flag in the TSS segment descriptor is provided to prevent re-entrant task switching and a subsequent loss of task state information. The processor manages the busy flag as follows:

1. When dispatching a task, the processor sets the busy flag of the new task.
2. If during a task switch, the current task is placed in a nested chain (the task switch is being generated by a CALL instruction, an interrupt, or an exception), the busy flag for the current task remains set.
3. When switching to the new task (initiated by a CALL instruction, interrupt, or exception), the processor generates a general-protection exception (#GP) if the busy flag of the new task is already set. If the task switch is initiated with an IRET instruction, the exception is not raised because the processor expects the busy flag to be set.
4. When a task is terminated by a jump to a new task (initiated with a JMP instruction in the task code) or by an IRET instruction in the task code, the processor clears the busy flag, returning the task to the "not busy" state.

The processor prevents recursive task switching by preventing a task from switching to itself or to any task in a nested chain of tasks. The chain of nested suspended tasks may grow to any length, due to multiple calls, interrupts, or exceptions. The busy flag prevents a task from being invoked if it is in this chain.

The busy flag may be used in multiprocessor configurations, because the processor follows a LOCK protocol (on the bus or in the cache) when it sets or clears the busy flag. This lock keeps two processors from invoking the same task at the same time. See Section 11.1.2.1, "Automatic Locking," for more information about setting the busy flag in a multiprocessor applications.

10.4.2 Modifying Task Linkages

In a uniprocessor system, in situations where it is necessary to remove a task from a chain of linked tasks, use the following procedure to remove the task:

1. Disable interrupts.
2. Change the previous task link field in the TSS of the pre-empting task (the task that suspended the task to be removed). It is assumed that the pre-empting task is the next task (newer task) in the chain from the task to be removed. Change the previous task link field to point to the TSS of the next oldest task in the chain or to an even older task in the chain.
3. Clear the busy (B) flag in the TSS segment descriptor for the task being removed from the chain. If more than one task is being removed from the chain, the busy flag for each task being removed must be cleared.
4. Enable interrupts.

In a multiprocessing system, additional synchronization and serialization operations must be added to this procedure to ensure that the TSS and its segment descriptor are both locked when the previous task link field is changed and the busy flag is cleared.

10.5 TASK ADDRESS SPACE

The address space for a task consists of the segments that the task can access. These segments include the code, data, stack, and system segments referenced in the TSS and any other segments accessed by the task code. The segments are mapped into the processor's linear address space, which is in turn mapped into the processor's physical address space (either directly or through paging).

The LDT segment field in the TSS can be used to give each task its own LDT. Giving a task its own LDT allows the task address space to be isolated from other tasks by placing the segment descriptors for all the segments associated with the task in the task's LDT.

It also is possible for several tasks to use the same LDT. This is a memory-efficient way to allow specific tasks to communicate with or control each other, without dropping the protection barriers for the entire system.

Because all tasks have access to the GDT, it also is possible to create shared segments accessed through segment descriptors in this table.

If paging is enabled, the CR3 register (PDBR) field in the TSS allows each task to have its own set of page tables for mapping linear addresses to physical addresses. Or, several tasks can share the same set of page tables.

10.5.1 Mapping Tasks to the Linear and Physical Address Spaces

Tasks can be mapped to the linear address space and physical address space in one of two ways:

- **One linear-to-physical address space mapping is shared among all tasks.** — When paging is not enabled, this is the only choice. Without paging, all linear addresses map to the same physical addresses. When paging is enabled, this form of linear-to-physical address space mapping is obtained by using one page directory for all tasks. The linear address space may exceed the available physical space if demand-paged virtual memory is supported.
- **Each task has its own linear address space that is mapped to the physical address space.** — This form of mapping is accomplished by using a different page directory for each task. Because the PDBR (control register CR3) is loaded on task switches, each task may have a different page directory.

The linear address spaces of different tasks may map to completely distinct physical addresses. If the entries of different page directories point to different page tables and the page tables point to different pages of physical memory, then the tasks do not share physical addresses.

With either method of mapping task linear address spaces, the TSSs for all tasks must lie in a shared area of the physical space, which is accessible to all tasks. This mapping is required so that the mapping of TSS addresses does not change while the processor is reading and updating the TSSs during a task switch. The linear address space mapped by the GDT also should be mapped to a shared area of the physical space; otherwise, the purpose of the GDT is defeated. Figure 10-9 shows how the linear address spaces of two tasks can overlap in the physical space by sharing page tables.

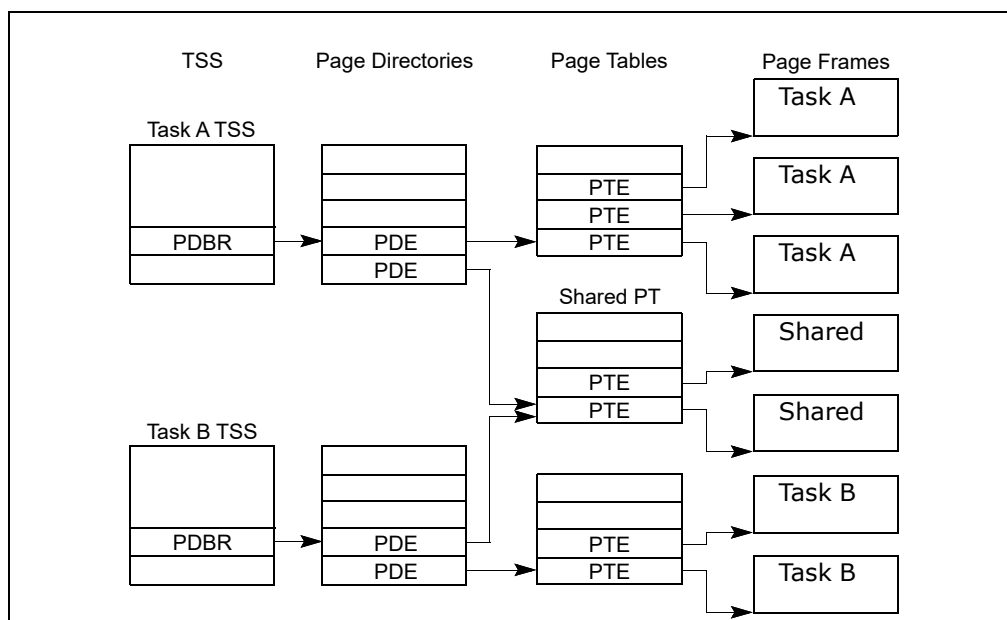


Figure 10-9. Overlapping Linear-to-Physical Mappings

10.5.2 Task Logical Address Space

To allow the sharing of data among tasks, use the following techniques to create shared logical-to-physical address-space mappings for data segments:

- **Through the segment descriptors in the GDT** — All tasks must have access to the segment descriptors in the GDT. If some segment descriptors in the GDT point to segments in the linear-address space that are

mapped into an area of the physical-address space common to all tasks, then all tasks can share the data and code in those segments.

- **Through a shared LDT** — Two or more tasks can use the same LDT if the LDT fields in their TSSs point to the same LDT. If some segment descriptors in a shared LDT point to segments that are mapped to a common area of the physical address space, the data and code in those segments can be shared among the tasks that share the LDT. This method of sharing is more selective than sharing through the GDT, because the sharing can be limited to specific tasks. Other tasks in the system may have different LDTs that do not give them access to the shared segments.
- **Through segment descriptors in distinct LDTs that are mapped to common addresses in linear address space** — If this common area of the linear address space is mapped to the same area of the physical address space for each task, these segment descriptors permit the tasks to share segments. Such segment descriptors are commonly called aliases. This method of sharing is even more selective than those listed above, because, other segment descriptors in the LDTs may point to independent linear addresses which are not shared.

10.6 16-BIT TASK-STATE SEGMENT (TSS)

The 32-bit IA-32 processors also recognize a 16-bit TSS format like the one used in Intel 286 processors (see Figure 10-10). This format is supported for compatibility with software written to run on earlier IA-32 processors.

The following information is important to know about the 16-bit TSS.

- Do not use a 16-bit TSS to implement a virtual-8086 task.
- The valid segment limit for a 16-bit TSS is 2CH.
- The 16-bit TSS does not contain a field for the base address of the page directory, which is loaded into control register CR3. A separate set of page tables for each task is not supported for 16-bit tasks. If a 16-bit task is dispatched, the page-table structure for the previous task is used.
- The I/O base address is not included in the 16-bit TSS. None of the functions of the I/O map are supported.
- When task state is saved in a 16-bit TSS, the upper 16 bits of the EFLAGS register and the EIP register are lost.
- When the general-purpose registers are loaded or saved from a 16-bit TSS, the upper 16 bits of the registers are modified and not maintained.

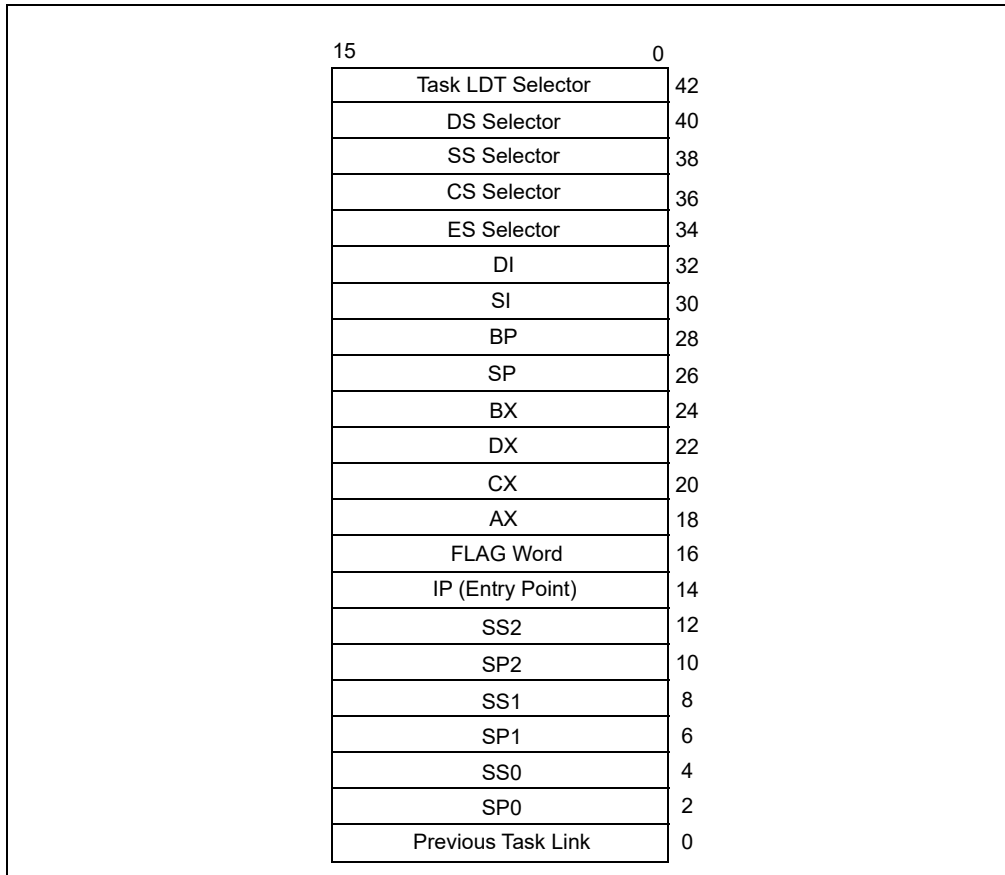


Figure 10-10. 16-Bit TSS Format

10.7 TASK MANAGEMENT IN 64-BIT MODE

In 64-bit mode, task structure and task state are similar to those in protected mode. However, the task switching mechanism available in protected mode is not supported in 64-bit mode. Task management and switching must be performed by software. The processor issues a general-protection exception (#GP) if the following is attempted in 64-bit mode:

- Control transfer to a TSS or a task gate using `JMP`, `CALL`, `INT n`, `INT3`, `INTO`, `INT1`, or interrupt.
- An `IRET` with `EFLAGS.NT` (nested task) set to 1.

Although hardware task-switching is not supported in 64-bit mode, a 64-bit task state segment (TSS) must exist. Figure 10-11 shows the format of a 64-bit TSS. The TSS holds information important to 64-bit mode and that is not directly related to the task-switch mechanism. This information includes:

- **RSPn** — The full 64-bit canonical forms of the stack pointers (RSP) for privilege levels 0-2.
- **ISTn** — The full 64-bit canonical forms of the interrupt stack table (IST) pointers.
- **I/O map base address** — The 16-bit offset to the I/O permission bit map from the 64-bit TSS base.

The operating system must create at least one 64-bit TSS after activating IA-32e mode. It must execute the `LTR` instruction (in 64-bit mode) to load the `TR` register with a pointer to the 64-bit TSS responsible for both 64-bit-mode programs and compatibility-mode programs.

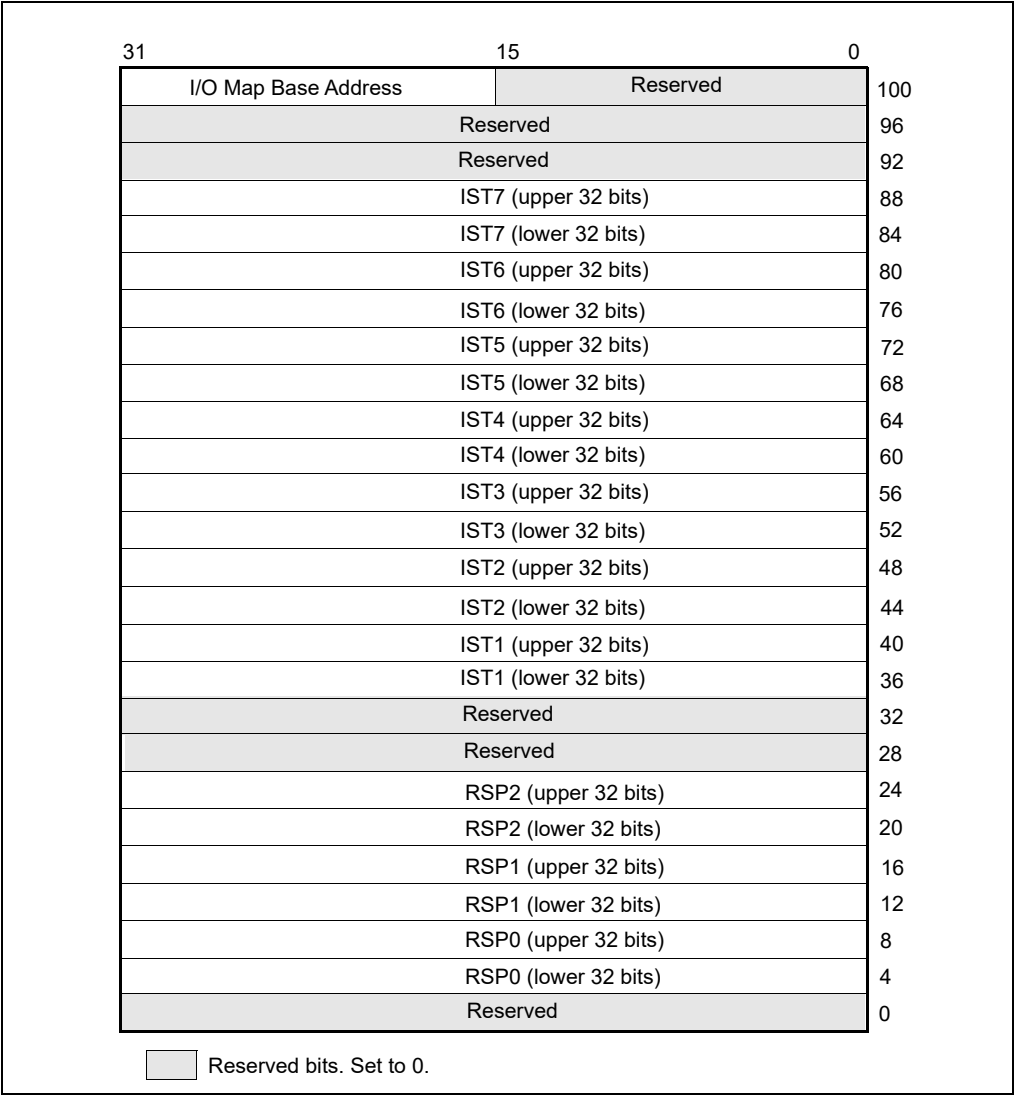


Figure 10-11. 64-Bit TSS Format

8. Updates to Chapter 17, Volume 3B

Change bars and violet 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:

- Pseudo code in Section 17.2 was updated to clarify that MPERF and APERF should be read together to avoid underflow or overflow in the calculation of the ratio between their values.

This chapter describes facilities of Intel 64 and IA-32 architecture used for power management and thermal monitoring.

17.1 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 in 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.

17.1.1 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 17-2. 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.

17.2 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 17-1 for an overview of the two 64-bit MSRs and the bullets below for a detailed description.

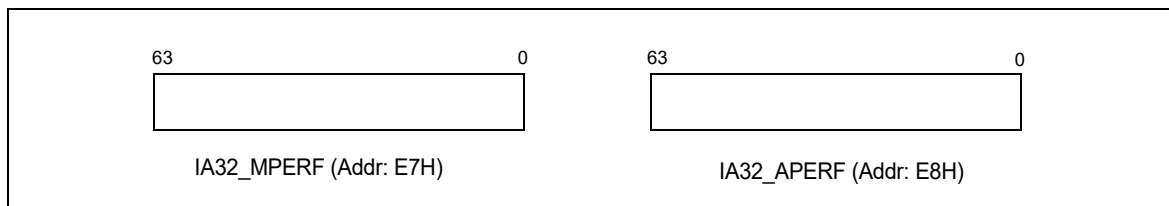


Figure 17-1. IA32_MPERF MSR and IA32_APERF MSR for P-state Coordination

- Use CPUID to check the P-State hardware coordination feedback capability bit. CPUID.06H:ECX[0] = 1 indicates IA32_MPERF MSR and IA32_APERF MSR are present.
- The IA32_MPERF MSR (E7H) increments in proportion to a fixed frequency, which is configured when the processor is booted.
- The 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 bits of the 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 17-1.

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 17-1 demonstrates steps for using the hardware feedback mechanism provided by IA32_APERF MSR and IA32_MPERF MSR to determine a target P-state.

Example 17-1. 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
//
// Software should minimize delay between RDMSR of ACNT and MCNT.
// If a delay occurs, it can lead to overflow or underflow in the division or multiplication
// by ignoring the sample.
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.
MARatio = MCNT/ACNT
IF MARatio = 0 OR __builtin_mul_overflow(PercentPerformance, MARatio) THEN
    // ignore sample
FI;
PercentBusy = PercentPerformance * MARatio

// 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 should minimize delay between the two WRMSRs below,
// but if a long delay happens the next sample will likely be rejected by the over/underflow checks above.
WRMSR(IA32_MPERF, 0);
WRMSR(IA32_APERF, 0);

```

17.3 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.

17.3.1 Intel® Dynamic Acceleration Technology

The Intel Core 2 Duo processor T7700 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.

17.3.2 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.

17.3.2.1 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 17.3.2.2. 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.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.

17.3.2.2 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 DISENGAGE 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 17.3.2.1.
- Expose the operating points associated with Intel Dynamic Acceleration Technology/Turbo mode to the OS.

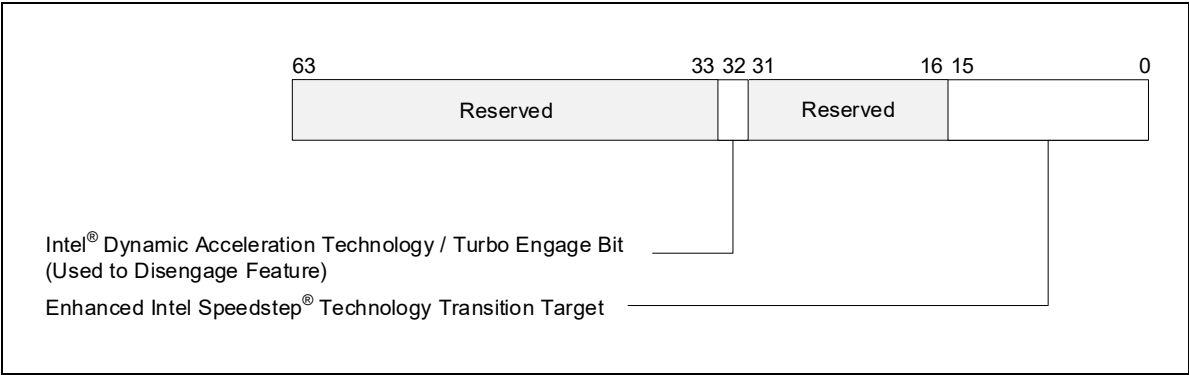


Figure 17-2. IA32_PERF_CTL Register

17.3.2.3 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. Soft-

were must query for the presence of IA32_APERF and IA32_MPERF (see Section 17.2). 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.

17.3.3 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 17.3.2 also applies to Intel Turbo Boost Technology.

17.3.4 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.06H. The processor supports this capability if CPUID.06H:ECX.ENERGY_PERF_BIAS[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.

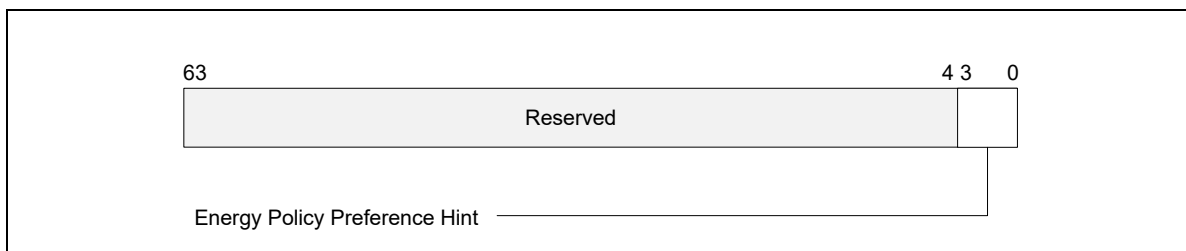


Figure 17-3. IA32_ENERGY_PERF_BIAS Register

The layout of IA32_ENERGY_PERF_BIAS is shown in Figure 17-3. 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.

17.4 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.

17.4.1 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.06H 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[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[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[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[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[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 17-1. 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.

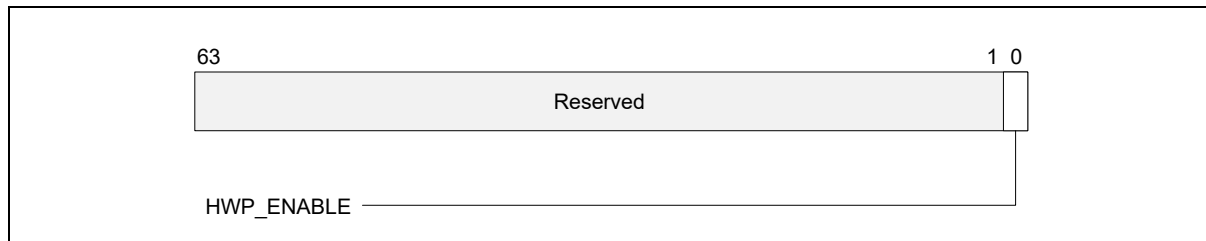
Table 17-1. Architectural and Non-Architectural MSRs Related to HWP

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.

17.4.2 Enabling HWP

The layout of the IA32_PM_ENABLE MSR is shown in Figure 17-4. The bit fields are described below:

**Figure 17-4. 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 that 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.

17.4.3 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 17-5. The bit fields are described below this figure.

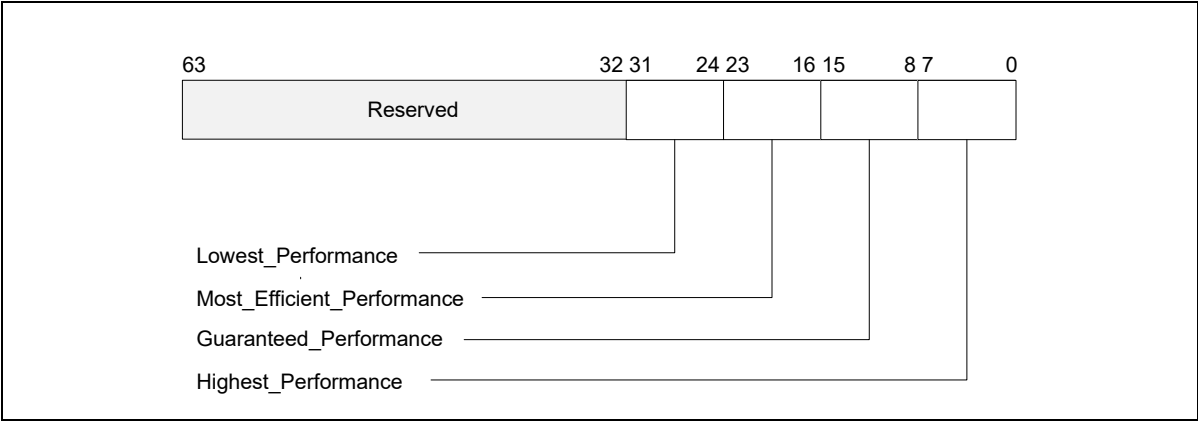


Figure 17-5. IA32_HWP_CAPABILITIES Register

- **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 17.10.1), 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.

17.4.4 Managing HWP

17.4.4.1 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 17-6. The bit fields are described below Figure 17-6.

Operating systems can control HWP by writing both IA32_HWP_REQUEST and IA32_HWP_REQUEST_PKG MSRs (see Section 17.4.4.2). 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.06H: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.

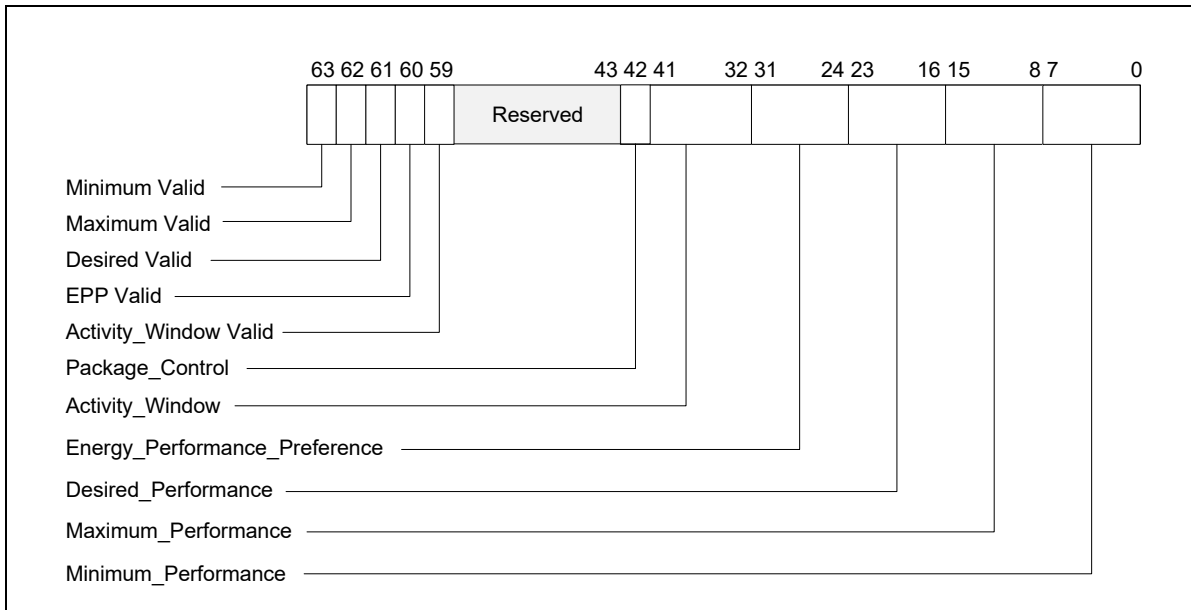


Figure 17-6. IA32_HWP_REQUEST Register

- **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[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/maximal 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 17.5) 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 17.4.8 for additional details).

17.4.4.2 IA32_HWP_REQUEST_PKG MSR (Address: 772H Package Scope)

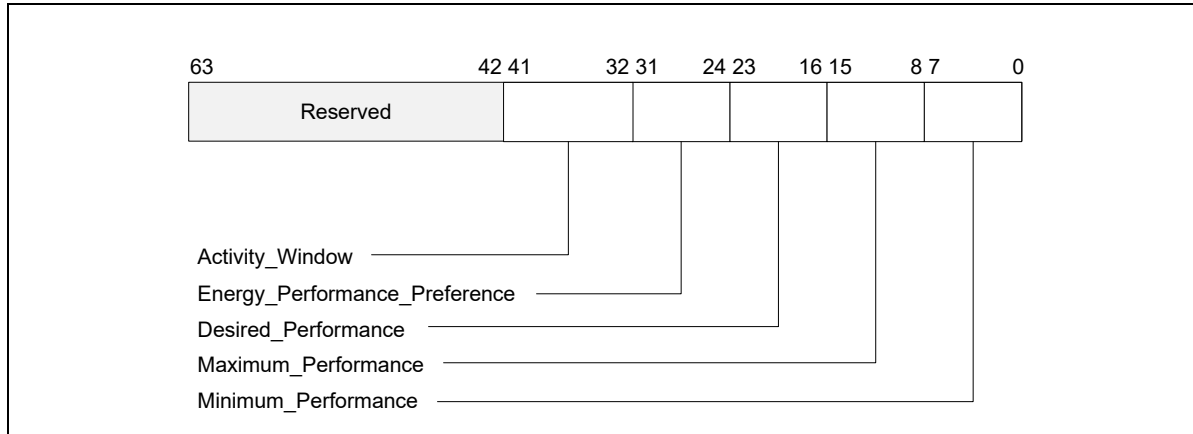


Figure 17-7. IA32_HWP_REQUEST_PKG Register

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.

17.4.4.3 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.06H:EAX[bit 16] is set.

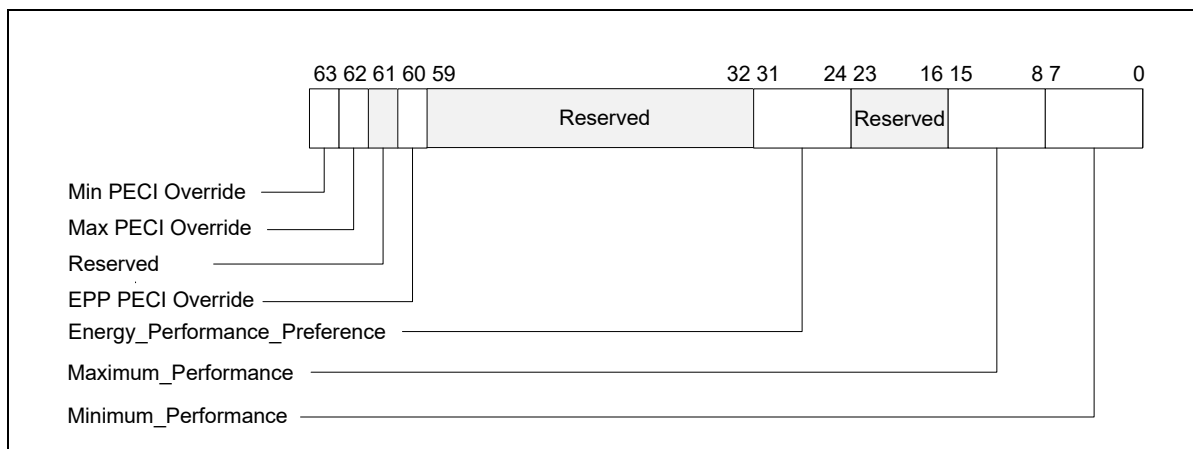


Figure 17-8. IA32_HWP_PECI_REQUEST_INFO MSR

The layout of the IA32_HWP_PECI_REQUEST_INFO MSR is shown in Figure 17-8. 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 is 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 is 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 is 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>

17.4.4.4 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 17.4.4.1 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 17.4.4.1 for additional details.

Section 17-2 summarizes the IA32_HWP_CTL MSR bit 0 control behavior.

Table 17-2. 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.06H:EAX[bit 22] is set.

If the IA32_PM_ENABLE[HWP_ENABLE] (bit 0) is not set, access to this MSR will generate a #GP fault.

17.4.5 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 17.2) 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 17-9. 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.06H:EAX[bit 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 17.4.4.3 for

- additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override entry are supported if CPUID.06H:EAX[bit 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 17.4.4.3 for additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override exit are supported if CPUID.06H:EAX[bit 16] is set.
 - Bits 63:6 are reserved and must be zero.

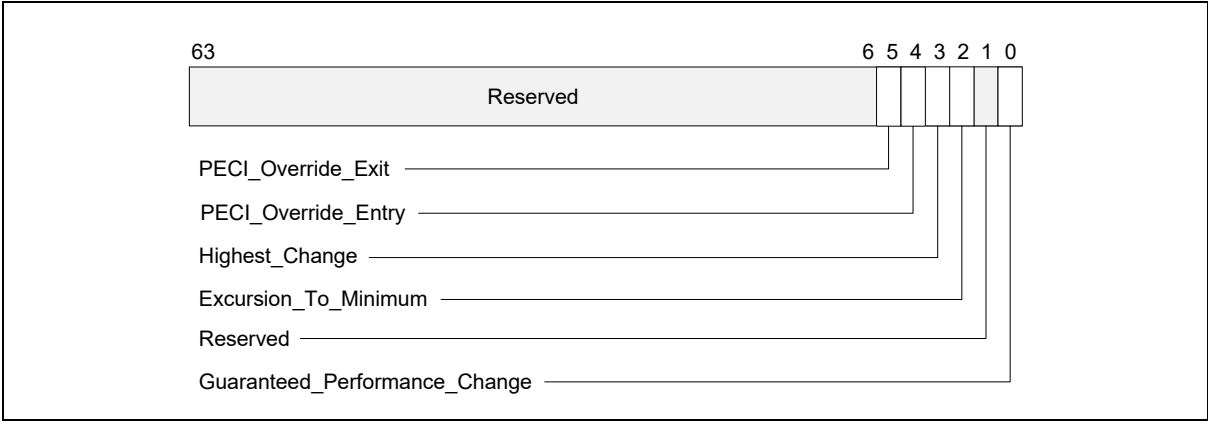


Figure 17-9. IA32_HWP_STATUS MSR

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 17-10. 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 17.8.5.2).

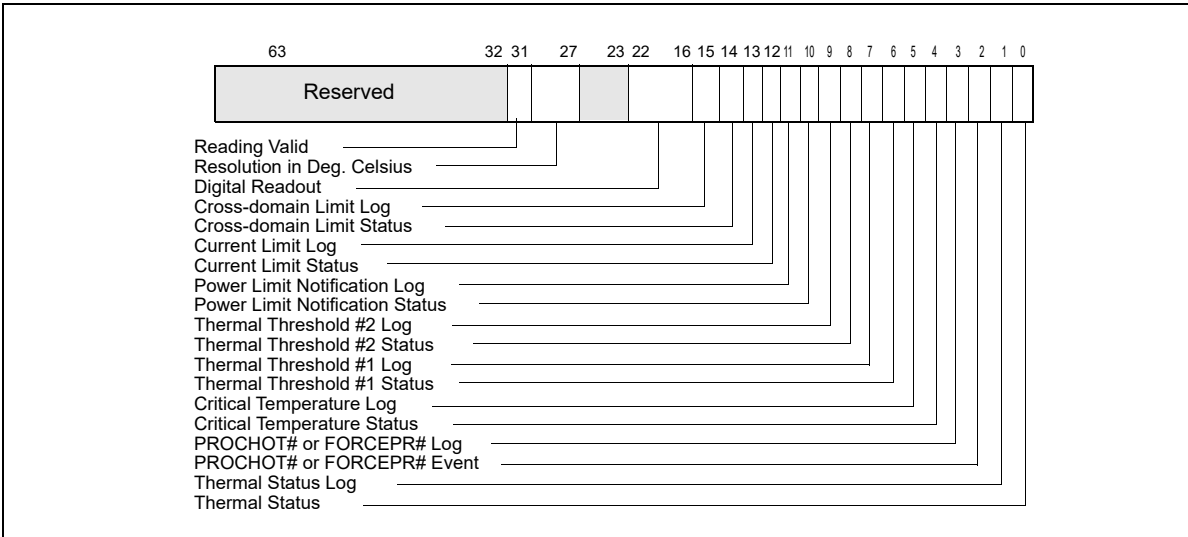


Figure 17-10. IA32_THERM_STATUS Register With HWP Feedback

- Bits 11:0, See Section 17.8.5.2.
- **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 17.8.5.2.

17.4.5.1 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 17-11.

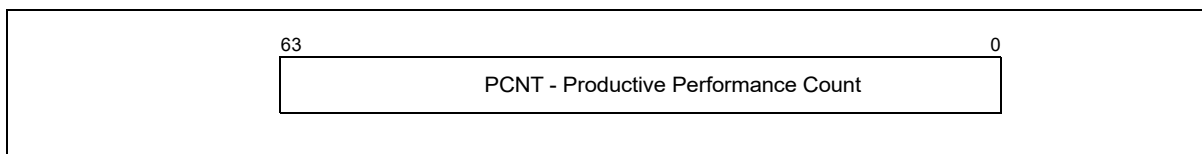


Figure 17-11. MSR_PPERF MSR

- **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 ($\Delta\text{PCNT}/\Delta\text{ACNT}$) is an indicator of workload scalability (0% to 100%). Note that values in this register are valid even when HWP is not enabled.

17.4.6 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 17-12. The bit fields are described below:

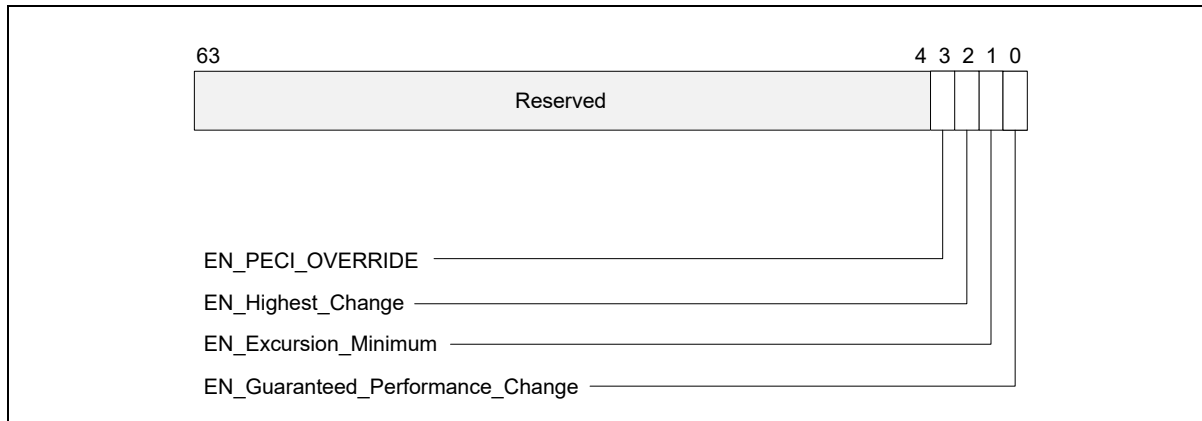


Figure 17-12. IA32_HWP_INTERRUPT MSR

- **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.06H:EAX[bit 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 17.4.4.3. The default value is 0 (interrupt generation is disabled). See Section 17.4.5 and Section 17.4.4.3 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.06H:EAX[bit 16] is set.
- Bits 63:4 are reserved and must be zero.

17.4.7 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.06H:EAX[bit 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.

17.4.8 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 17.4.8.3 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 17.4.8.1). 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.

17.4.8.1 FAST_UNCORE_MSRS_CAPABILITY (Address: 65FH, Logical Processor Scope)

Operating systems or BIOS can read the FAST_UNCORE_MSRS_CAPABILITY MSR to enumerate those MSRs that are Fast Write MSRs.

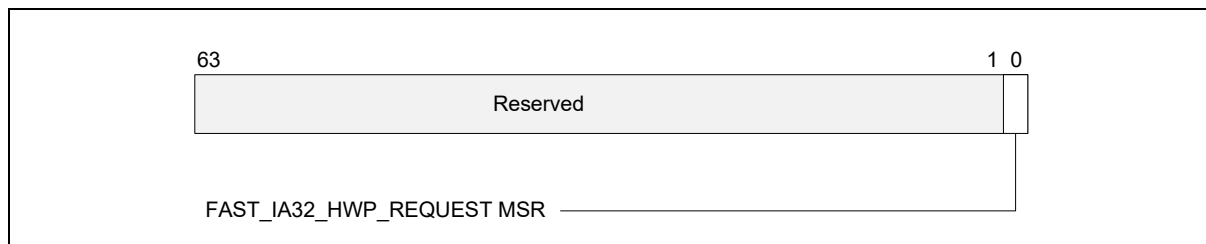


Figure 17-13. FAST_UNCORE_MSRS_CAPABILITY MSR

- **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.

17.4.8.2 FAST_UNCORE_MSRS_CTL (Address: 657H, 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.

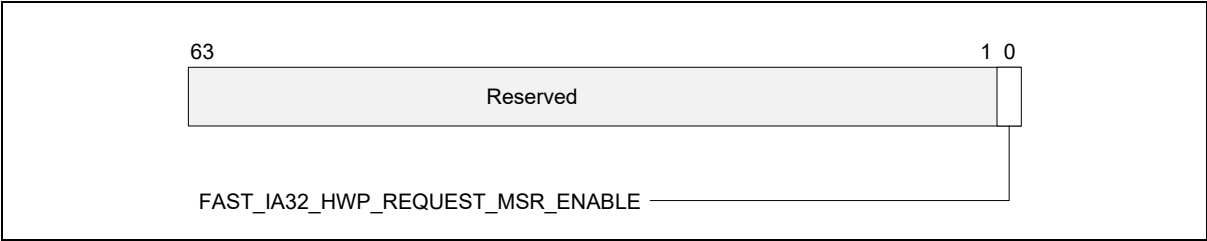


Figure 17-14. FAST_UNCORE_MSRS_CTL MSR

- **FAST_IA32_HWP_REQUEST_MSR_ENABLE (bit 0, R/W)** — When set (1), enables the fast access mode, low latency, posted IA32_HWP_REQUEST MSR feature and sets the related CPUID.06H:EAX[18] bit. 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 CPU RESET will clear this bit.
- Bits 63:1 are reserved and must be zero.

17.4.8.3 FAST_UNCORE_MSRS_STATUS (Address: 65EH, 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”.

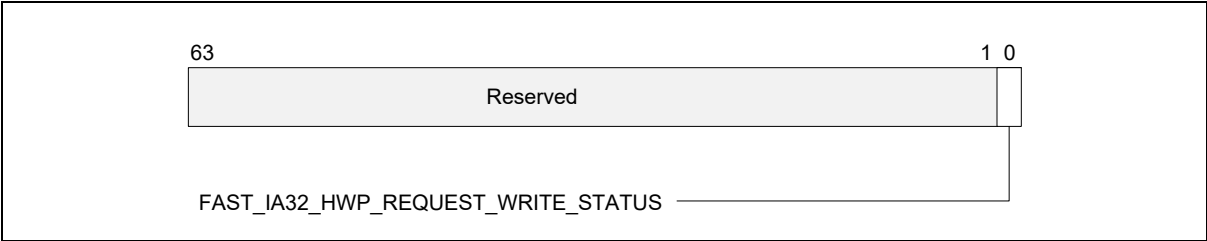


Figure 17-15. FAST_UNCORE_MSRS_STATUS MSR

- **FAST_IA32_HWP_REQUEST_WRITE_STATUS (bit 0, R/O)** — 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.

17.4.9 Fast_IA32_HWP_REQUEST CPUID

IA32_HWP_REQUEST is an architectural MSR that exists in processors whose CPUID.06H: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 bit is set (CPUID.06H:EAX[18]), writes to the IA32_HWP_REQUEST MSR are visible outside the logical processor via the “Fast Write” feature described in Section 17.4.8.

17.4.10 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.

17.5 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 of 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.

17.5.1 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, CPUID.06H:EAX[13] indicates the processor’s support of the following aspects of HDC.
 - Availability of HDC baseline resource, CPUID.06H:EAX[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 17-3. Architectural and non-Architecture MSRs Related to HDC

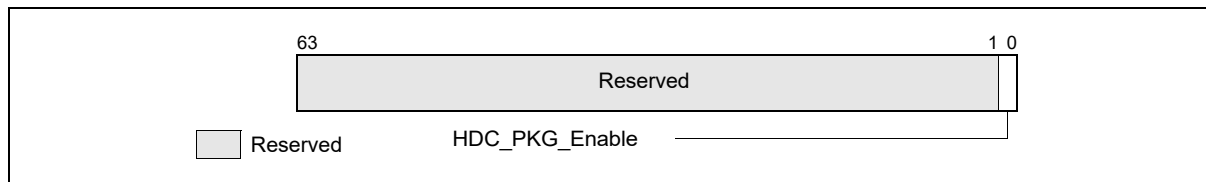
Address	Architectural	Register Name	Description
DB0H	Y	IA32_PKG_HDC_CTL	Package Enable/Disable HDC.
DB1H	Y	IA32_PM_CTL1	Per-logical-processor select control to allow/block HDC forced idling.
DB2H	Y	IA32_THREAD_STALL	Accumulate stalled cycles on this logical processor due to HDC forced idling.
653H	N	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.

17.5.2 Package level Enabling HDC

The layout of the IA32_PKG_HDC_CTL MSR is shown in Figure 17-16. IA32_PKG_HDC_CTL is a writable MSR from any logical processor in a package. The bit fields are described below:

**Figure 17-16. 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 17.5.3) 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[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 17.5.3). 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.

17.5.3 Logical-Processor Level HDC Control

The layout of the IA32_PM_CTL1 MSR is shown in Figure 17-17. Each logical processor in a package has its own IA32_PM_CTL1 MSR. The bit fields are described below:

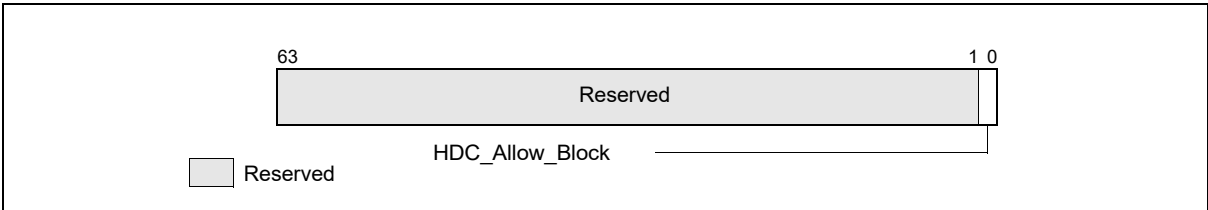


Figure 17-17. IA32_PM_CTL1 MSR

- **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[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_P-M_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.

17.5.4 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.

17.5.4.1 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 17-18. Each logical processor in a package has its own IA32_THREAD_STALL MSR. The bit fields are described below:

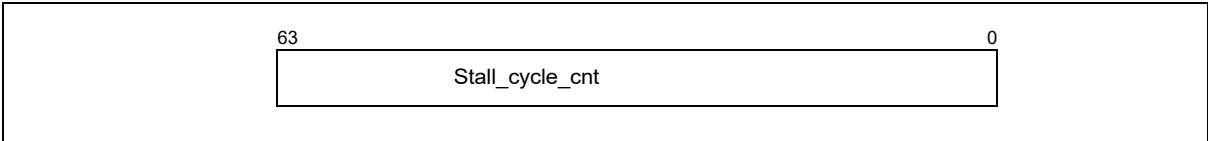


Figure 17-18. IA32_THREAD_STALL MSR

- **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[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[13] = 1.

17.5.4.2 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 17-19. Each processor core in a package has its own MSR_CORE_HDC_RESIDENCY MSR. The bit fields are described below:

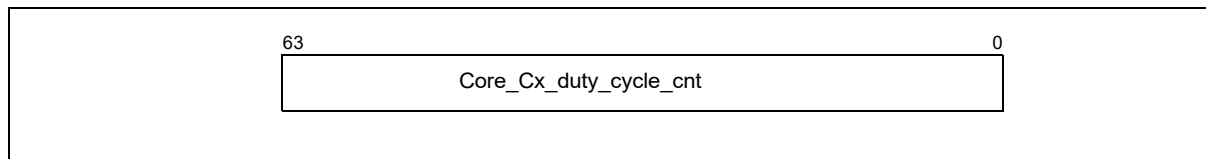


Figure 17-19. MSR_CORE_HDC_RESIDENCY MSR

- **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[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 17-20. There is one MSR_PKG_HDC_SHALLOW_RESIDENCY per package. The bit fields are described below:

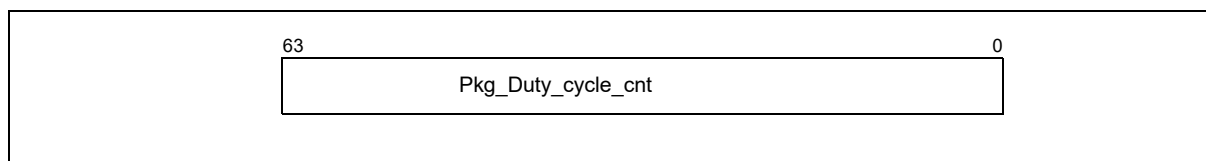


Figure 17-20. MSR PKG HDC SHALLOW RESIDENCY MSR

- **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[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 17-21. There is one MSR_PKG_HDC_DEEP_RESIDENCY per package. The bit fields are described below:

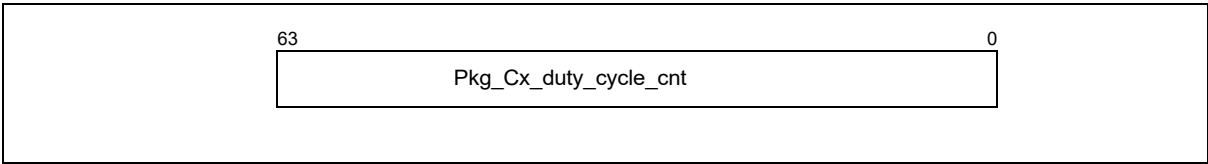


Figure 17-21. MSR_PKG_HDC_DEEP_RESIDENCY MSR

- **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[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 17-22. There is one MSR_PKG_HDC_CONFIG per package. The bit fields are described below:

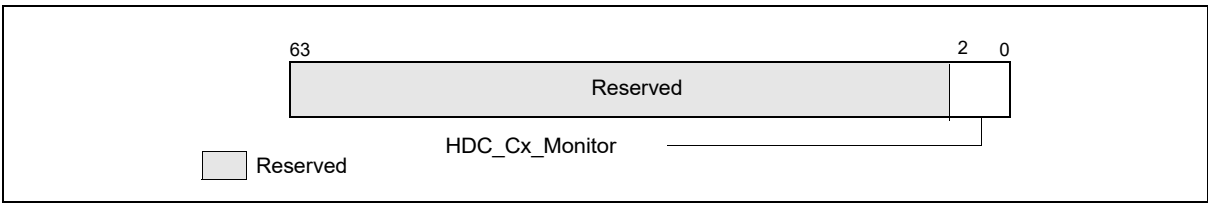


Figure 17-22. MSR_PKG_HDC_CONFIG MSR

- **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[13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).
- Bits 63:3 are reserved and must be zero.

17.5.5 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.

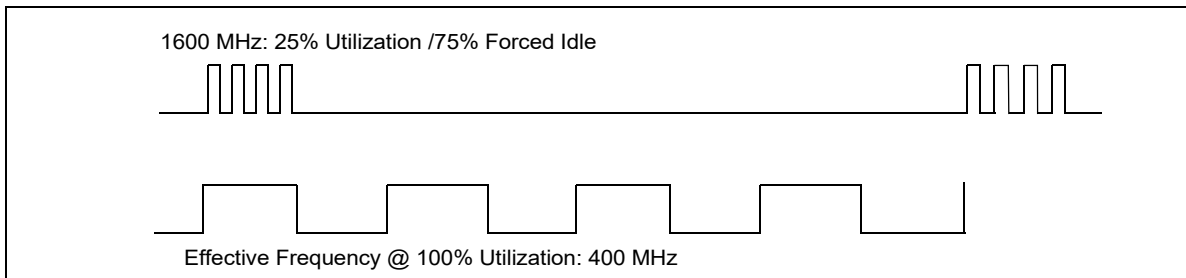


Figure 17-23. Example of Effective Frequency Reduction and Forced Idle Period of HDC

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 $\Delta\text{ACNT}/\Delta\text{MCNT} * \text{TSC Frequency}$.

17.6 HARDWARE FEEDBACK INTERFACE AND INTEL® THREAD DIRECTOR

Intel processors that enumerate CPUID.06H.00H:EAX.HW_FEEDBACK[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 17.6.1.

Intel processors that enumerate CPUID.06H.00H:EAX[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 17.6.2.

17.6.1 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 17-4. 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 17-5.

Table 17-5. 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 the OS to 0.
8	1	Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
9	1	Energy Efficiency Capability Changed	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
10	6	Reserved	Initialized by the 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.00H:EDX[31:16] by executing CPUID on that logical processor.

Table 17-6. 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. The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.00H: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. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.00H:EDX[1] enumerates support for Energy Efficiency capability reporting.
2	6	Reserved	The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback.

17.6.2 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 17-7). Following this global header there are multiple Logical Processor related entries. The structure is designed as follows.

Table 17-7. 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.00H:EDX[7:0] and CL (# of classes): CPUID.06H.00H: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.00H: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 17-8.

Table 17-8. 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 the OS to 0.	
8	1	Class 0 Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + 1	1	Class 0 Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the 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 Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + CP + 1	1	Class 1 Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the 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 Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + (CL-1)*CP + 1	1	Class #(CL-1) Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.

Table 17-8. Intel® Thread Director Global Header Structure (Contd.)

Byte Offset	Size (Bytes)	Description	
...			
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 the 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 17-9. 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 the 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. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the 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 the 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 the 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. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
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).

Table 17-9. Intel® Thread Director Logical Processor Entry Structure (Contd.)

Byte Offset	Size (Bytes)	Field Name	Description
2*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the 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 the 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. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the 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 the OS to 0 prior to enabling Intel Thread Director.

17.6.3 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: $\text{Perf Ratio}_1 = \text{Perf}_{ik1} / \text{Perf}_{jk1}$ and $\text{Perf Ratio}_2 = \text{Perf}_{ik2} / \text{Perf}_{jk2}$, or two energy efficiency ratios: $\text{Energy Eff. Ratio}_1 = \text{Energy Eff}_{ik1} / \text{Energy Eff}_{jk1}$ and $\text{Energy Eff. Ratio}_2 = \text{Energy Eff}_{ik2} / \text{Energy Eff}_{jk2}$ 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 $\text{Perf Ratio}_1 > \text{Perf Ratio}_2$. 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.

17.6.4 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:MAXPHYADDR¹ — 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.00H:EDX[11:8] enumerates the size of memory that must be allocated by the OS for this structure.

17.6.5 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.06H:EAX`[bit 23] is set. When setting this bit while support is not enumerated, the hardware generates #GP.

Table 17-10 summarizes the control options described above.

See Section 17.6.9 for details on scenarios where `IA32_HW_FEEDBACK_CONFIG` bits are implicitly reset by the hardware.

1. MAXPHYADDR is derived from the value enumerated in `CPUID.80000008H:EAX`[7:0]. If `IA32_TME_ACTIVATE`[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of `IA32_TME_ACTIVATE`[39:36] when a logical processor is out-side secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.

Table 17-10. 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.

17.6.6 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.

17.6.7 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 17.6.6 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.

Zeroing a performance and energy efficiency cell hint of a logical processor across all classes along with Capability Flag bit 1 set to 1 across all capabilities and classes, indicates to the OS to force idle logical processor(s), and if affinity activity occurs on those logical processor(s), the OS should inject idle periods such that overall utilization of those idled cores has a minimal-to-no impact to power. Capability Flag bit 1 will be set to 1 while this hint persists.

When EE=255 is set on one or more logical processors, it represents a request that the OS attempt to consolidate work to those logical processors with EE=255. These requests are made when the SOC has knowledge that consolidating the work to a subset of cores will result in significantly better platform energy efficiency. Examples of consolidating work would include, but not limited to, delaying less important work as needed to provide compute bandwidth for more important work, and routing interrupts to the logical processors with EE=255. When the cumulative workload requires performance greater than that which is available on the subset of cores with EE=255, it is expected that the OS will scale the work out to additional logical processors.

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.

17.6.8 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 17.6.9 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

17.6.9 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[SENDER] 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.

17.6.10 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 17-8.

This MSR is valid only if CPUID.06H:EAX[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.

17.6.11 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.

17.6.11.1 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.

17.6.11.2 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[SENDER] 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[SENDER] 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.

17.7 MWAIT EXTENSIONS FOR ADVANCED POWER MANAGEMENT

IA-32 processors may support a number of C-states¹ 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[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[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.

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.

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.

17.8 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 mechanisms** force the processor to reduce its 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 17-24, 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.

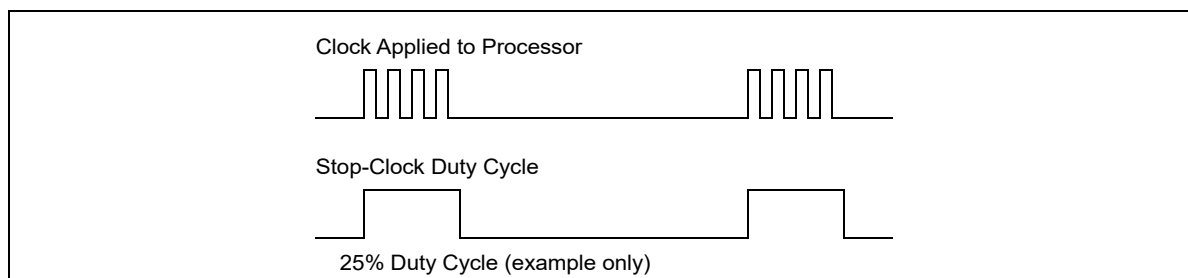


Figure 17-24. Processor Modulation Through Stop-Clock Mechanism

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.

17.8.1 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.

17.8.2 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.

17.8.2.1 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.01H:EDX.TM[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.

17.8.2.2 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.01H:ECX.TM2[8] = 1.

17.8.2.3 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 17-25) 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.

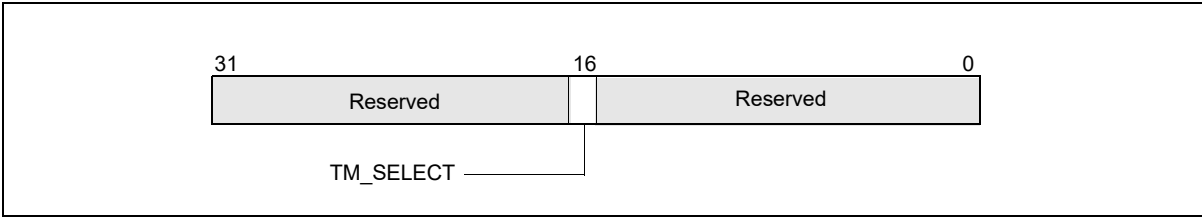


Figure 17-25. MSR_THERM2_CTL Register On Processors with CPUID Family/Model/Stepping Signature Encoded as 0x69n or 0x6Dn

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 17-26). 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 TM1 or TM2; and they must not alter the value in bits 15:0 of the MSR_THERM2_CTL register.

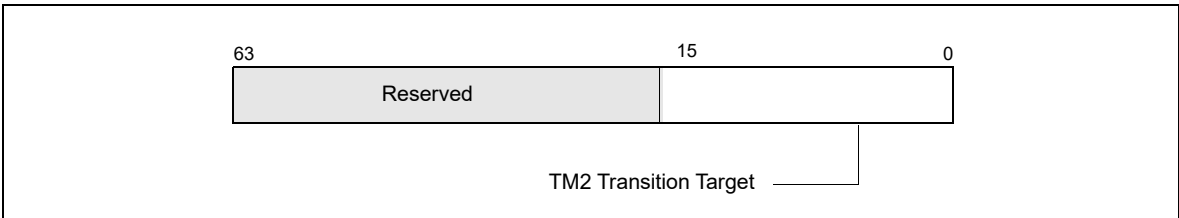


Figure 17-26. MSR_THERM2_CTL Register for Supporting TM2

17.8.2.4 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.

17.8.2.5 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 17-27).

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.

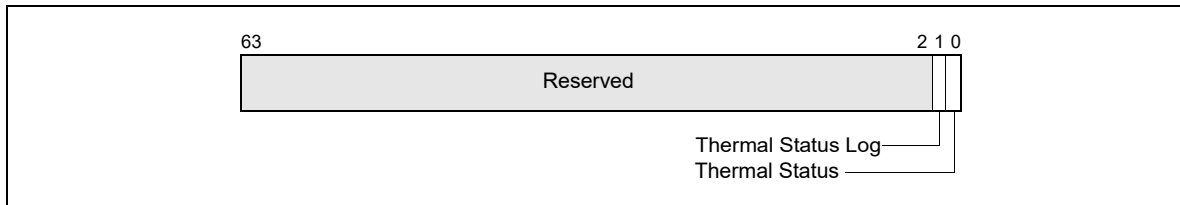


Figure 17-27. IA32_THERM_STATUS MSR

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 13.5.1, "Local Vector Table"). The low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR (see Figure 17-28) control when the interrupt is generated; that is, on a transition from a temperature below the trip point to above and/or vice-versa.

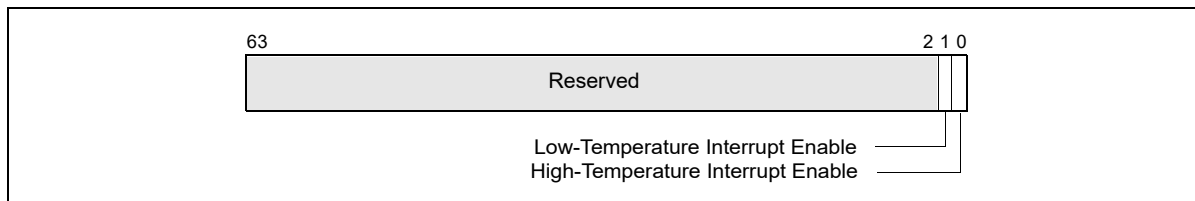


Figure 17-28. IA32_THERM_INTERRUPT MSR

- **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).

17.8.2.6 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 17.8.2.1 and Section 17.8.2.2. 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.

17.8.3 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 17-29).

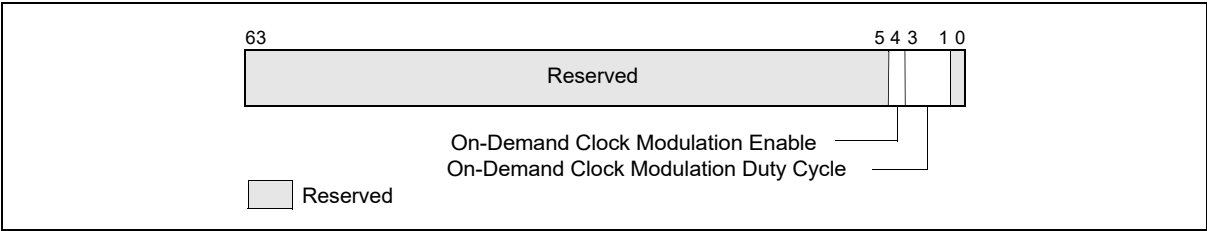


Figure 17-29. IA32_CLOCK_MODULATION MSR

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 17-11). 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 17-11. 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 Chapter 3, “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.

17.8.3.1 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 17-30).

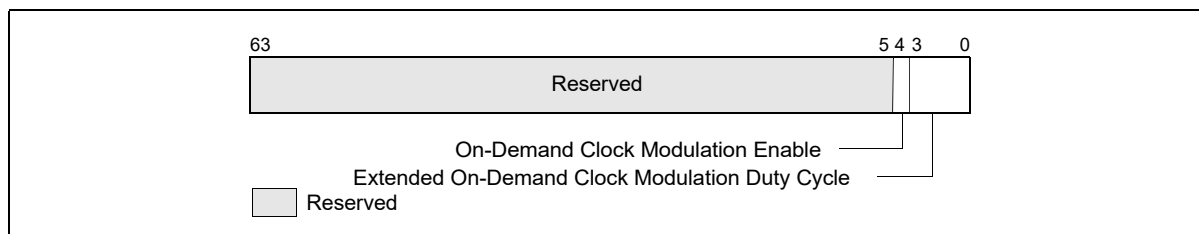


Figure 17-30. IA32_CLOCK_MODULATION MSR with Clock Modulation Extension

Extension to software controlled clock modulation is supported only if CPUID.06H:EAX[5] = 1. If CPUID.06H:EAX[5] = 0, then bit 0 of IA32_CLOCK_MODULATION is reserved.

17.8.4 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.

17.8.4.1 Detection of Software Controlled Clock Modulation Extension

Processor’s support of software controlled clock modulation extension is indicated by CPUID.06H:EAX[5] = 1.

17.8.5 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.

17.8.5.1 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.

17.8.5.2 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 17-31 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.

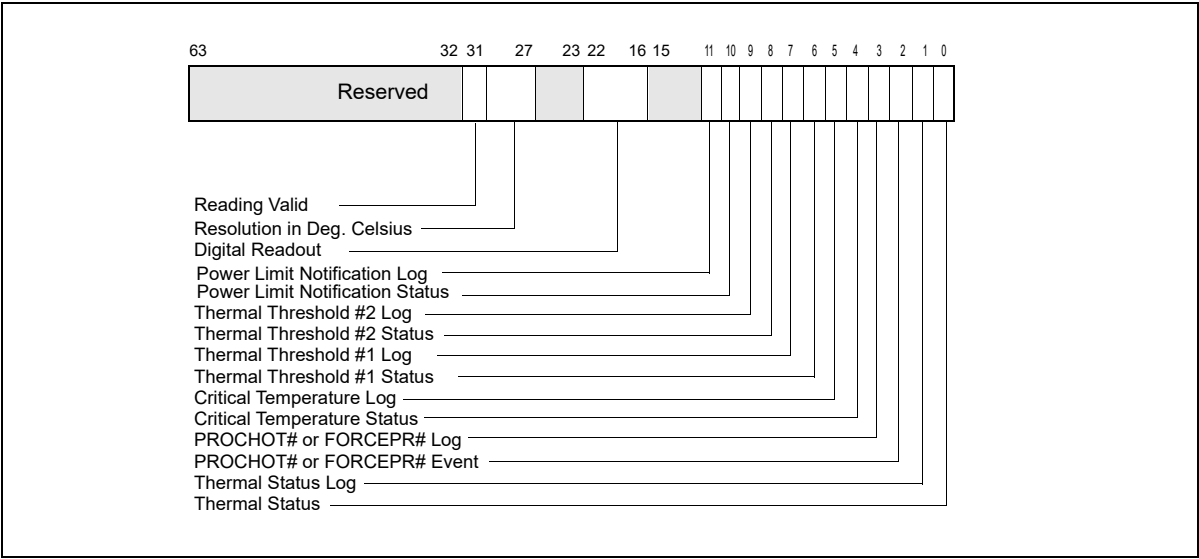


Figure 17-31. IA32_THERM_STATUS Register

- **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[4] = 1. Package level power limit notification can be delivered independently to IA32_PACKAGE_THERM_STATUS MSR.
- **Power Notification Log (bit 11, R/WC0)** — 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[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 17-32); 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.

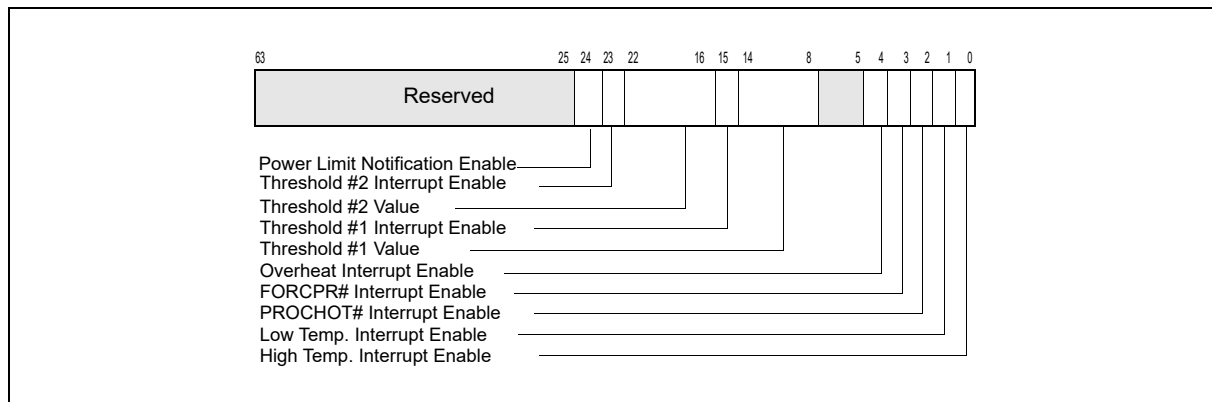


Figure 17-32. IA32_THERM_INTERRUPT Register

See Figure 17-32 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 = 1 enables 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[4] = 1. Package level power limit notification can be enabled independently by IA32_PACKAGE_THERM_INTERRUPT MSR.

17.8.6 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[4] = 1`.

If `CPUID.06H:EAX[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 17-31).
- 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 17-32).

17.9 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[6] = 1`.

The layout of `IA32_PACKAGE_THERM_STATUS` MSR is shown in Figure 17-33.

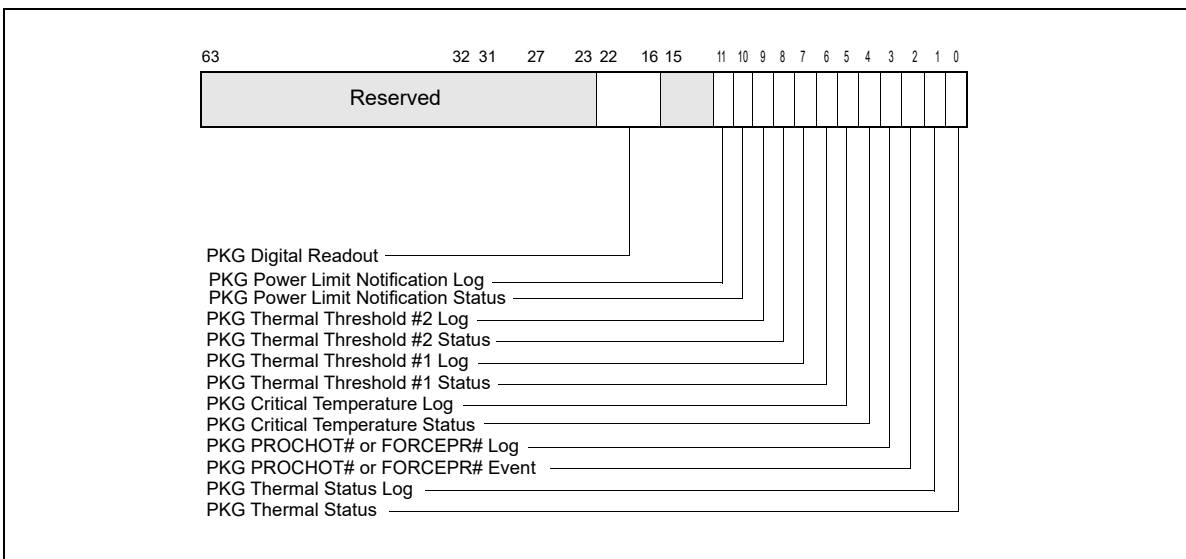


Figure 17-33. `IA32_PACKAGE_THERM_STATUS` Register

- **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 Thermal 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 or 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 17-31).
- **Package Power Notification Log (bit 11, R/WC0)** — 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 17-34.

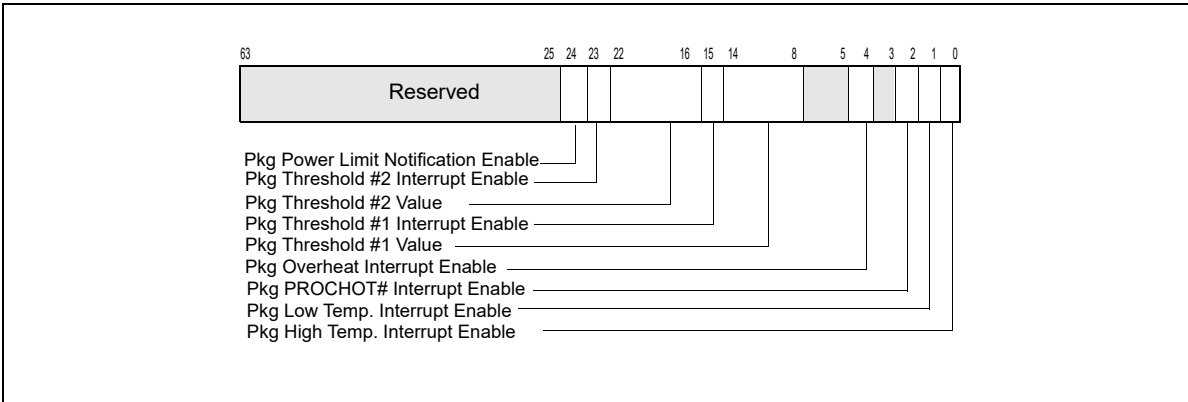


Figure 17-34. IA32_PACKAGE_THERM_INTERRUPT Register

- **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.

17.9.1 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).

17.10 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.

17.10.1 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.

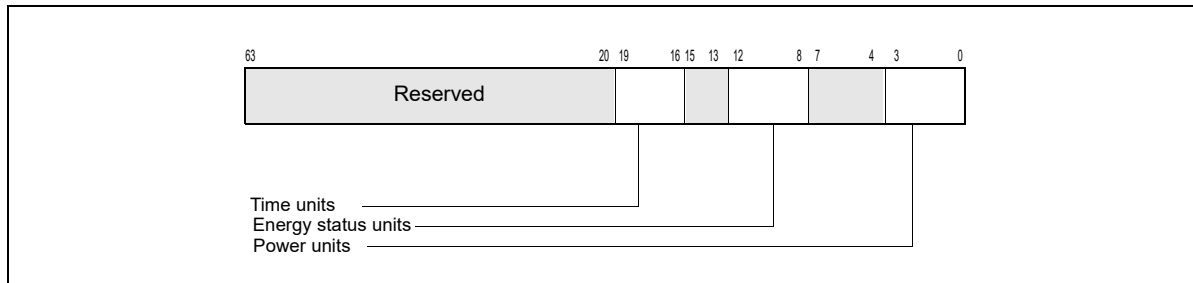


Figure 17-35. MSR_RAPL_POWER_UNIT Register

MSR_RAPL_POWER_UNIT (Figure 17-35) provides the following information across all RAPL domains:

- **Power Units** (bits 3:0): Power related information (in Watts) is based on the multiplier, $1/2^{\text{PU}}$; 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, $1/2^{\text{ESU}}$; 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, $1/2^{\text{TU}}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating time unit is in 976 micro-seconds increment.

17.10.2 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 17-12 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 17-12. 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

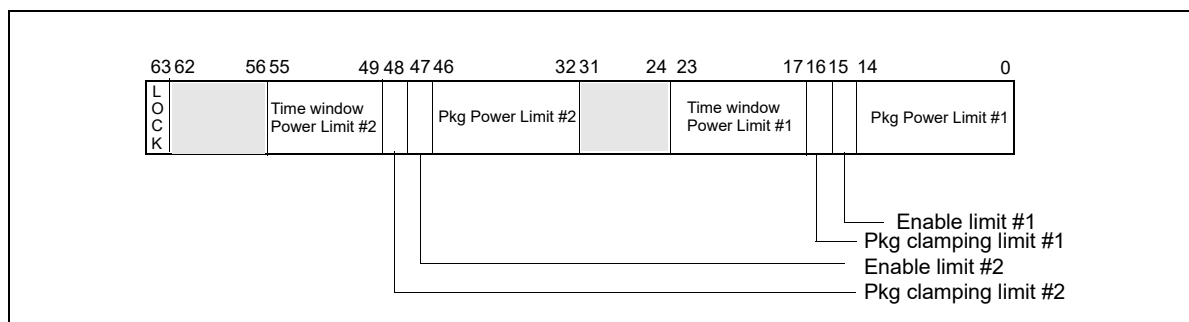
The presence of the optional MSR interfaces (the three right-most columns of Table 17-12) may be model-specific. See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for details.

17.10.3 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 17-36. 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 17-36) 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

$$\text{Time limit} = 2^Y * (1.0 + Z/4.0) * \text{Time\_Unit}$$

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

$$\text{Time limit} = 2^Y * (1.0 + Z/4.0) * \text{Time\_Unit}$$

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.

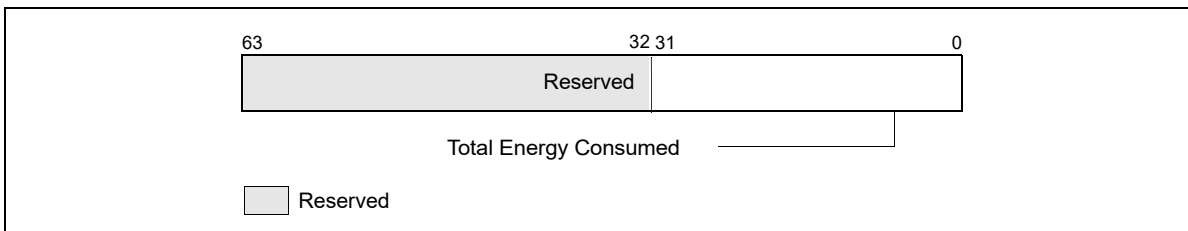


Figure 17-37. MSR_PKG_ENERGY_STATUS MSR

- **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.

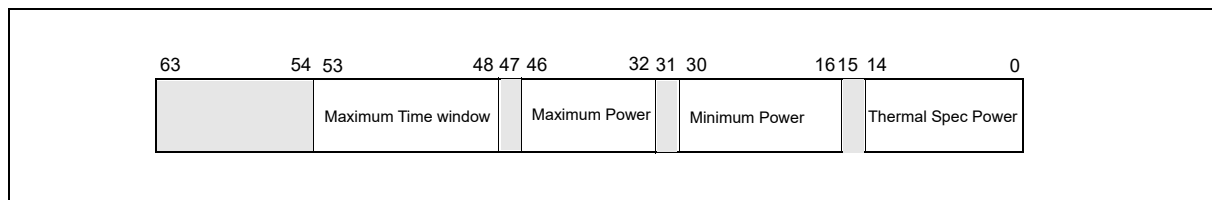


Figure 17-38. MSR_PKG_POWER_INFO Register

- **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.

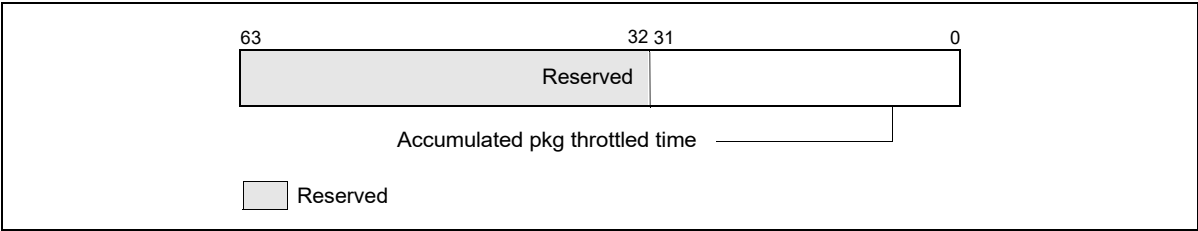


Figure 17-39. MSR_PKG_PERF_STATUS MSR

- **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.

17.10.4 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.

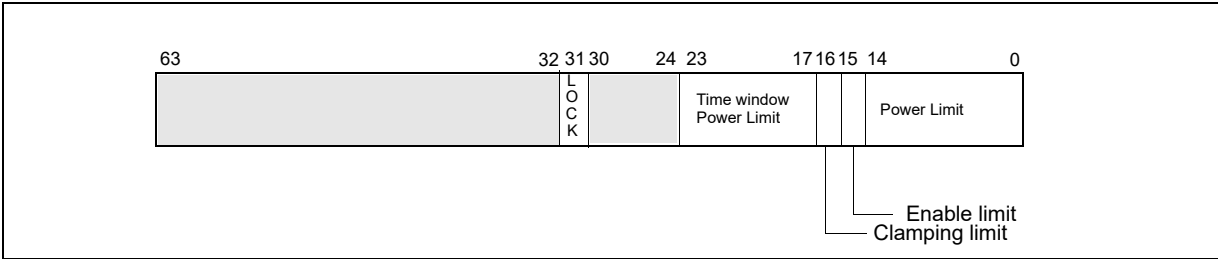


Figure 17-40. MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT Register

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 17-40) 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.

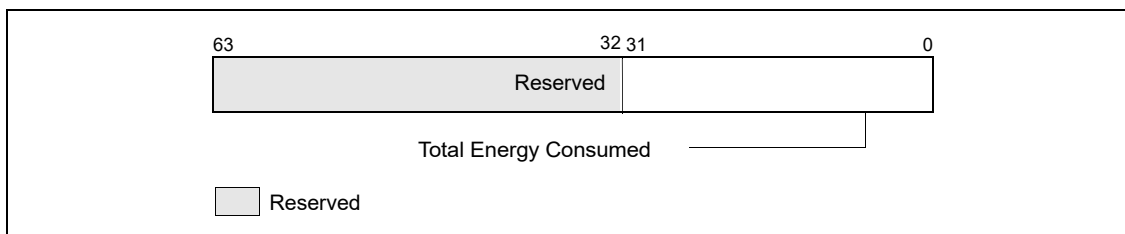


Figure 17-41. MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS MSR

- **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.

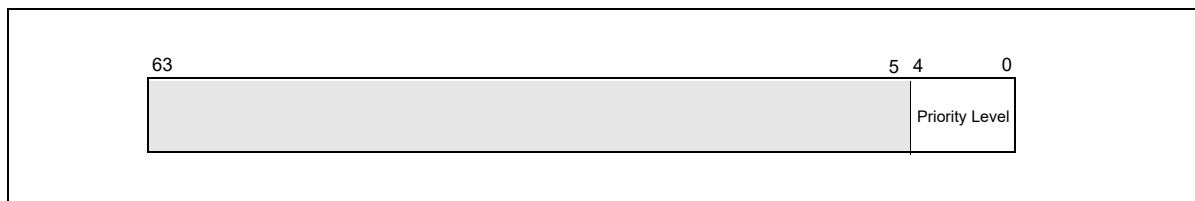


Figure 17-42. MSR_PP0_POLICY/MSR_PP1_POLICY Register

- **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.

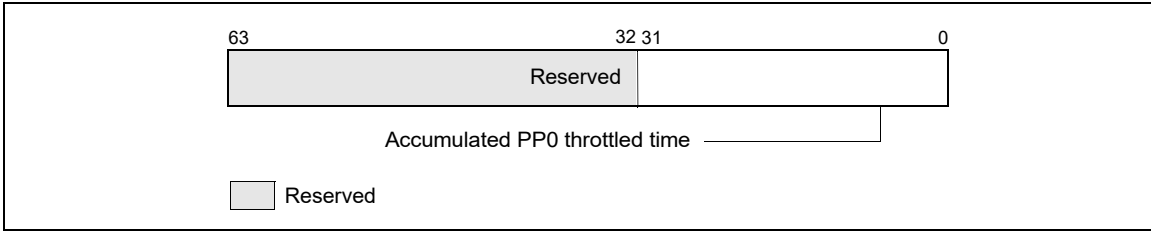


Figure 17-43. MSR_PPO_PERF_STATUS MSR

- **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.

17.10.5 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.

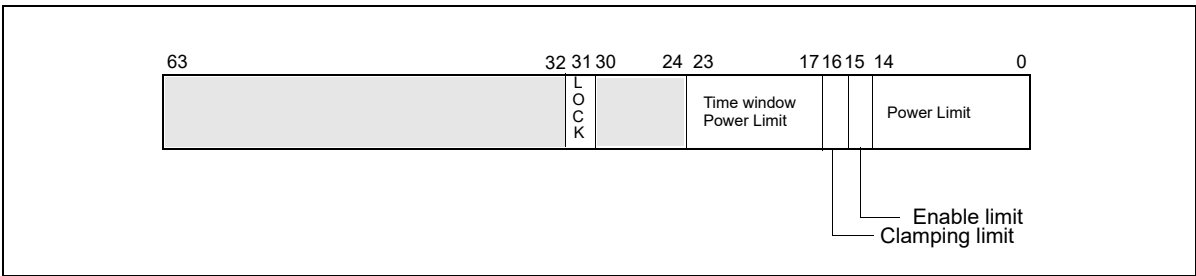


Figure 17-44. MSR_DRAM_POWER_LIMIT Register

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 un-modifiable until next RESET.

The bit fields of MSR_DRAM_POWER_LIMIT (Figure 17-44) 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.

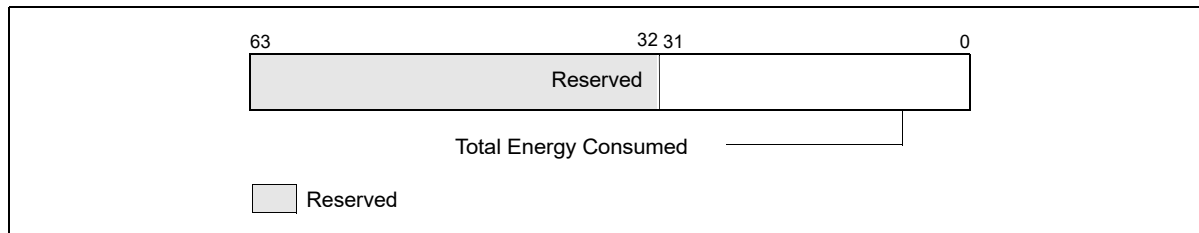


Figure 17-45. MSR_DRAM_ENERGY_STATUS MSR

- **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.

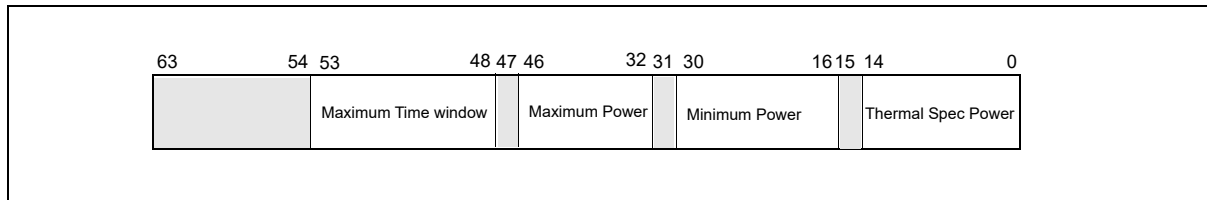


Figure 17-46. MSR_DRAM_POWER_INFO Register

- **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.

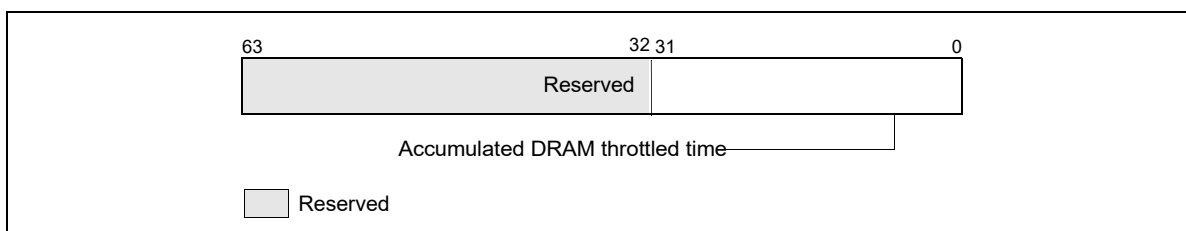


Figure 17-47. MSR_DRAM_PERF_STATUS MSR

- **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.

9. Updates to Chapter 20, Volume 3B

Change bars and violet text show changes to Chapter 20 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- In Figure 20-23, added details for RMID and CLOS bit fields in IA32_PQR_ASSOC MSR.
- In Figure 20-31, added Resource Priority (RP) and Cache Bandwidth Allocation (CBA) bits to CPUID.10H.00H:EBX register.
- In Section 20.19.4.4, "Class of Service to Cache Mask Association: Common Across Allocation Features," updated the bits containing the CLOS field to M:32 for a logical processor, defining formulas for CLOS_MAX, CLOS_WIDTH, and M.
- Added new Section 20.19.8, "Cache Bandwidth Allocation (CBA)."
- In Section 20.20.3, CPUID field names were updated for Asymmetric RDT-M and Asymmetric RDT-A in Table 20-22 and Table 20-23.
- Section 20.21 was updated and renamed to "Intel® RDT Derivative Technologies" moving Core Telemetry for Intel PMT to a subsection and adding Resource Priority as a subsection.

CHAPTER 20

DEBUG, BRANCH PROFILE, TSC, AND INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) FEATURES

NOTE

This chapter makes numerous references to last-branch recording (LBR) facilities. Unless noted otherwise, all such references in this chapter are to an earlier non-architectural form of the feature. Chapter 21 defines an architectural form of last-branch recording that is supported on newer processors.

Intel 64 and IA-32 architectures provide debug facilities for use in debugging code and monitoring performance. These facilities are valuable for debugging application software, system software, and multitasking operating systems. Debug support is accessed using debug registers (DR0 through DR7) and model-specific registers (MSRs):

- Debug registers hold the addresses of memory and I/O locations called breakpoints. Breakpoints are user-selected locations in a program, a data-storage area in memory, or specific I/O ports. They are set where a programmer or system designer wishes to halt execution of a program and examine the state of the processor by invoking debugger software. A debug exception (#DB) is generated when a memory or I/O access is made to a breakpoint address.
- MSRs monitor branches, interrupts, and exceptions; they record addresses of the last branch, interrupt or exception taken and the last branch taken before an interrupt or exception.
- Time stamp counter is described in Section 20.17, "Time-Stamp Counter."
- Features that allow monitoring of shared platform resources such as the L3 cache are described in Section 20.18, "Intel® Resource Director Technology (Intel® RDT) Monitoring Features."
- Features that enable control over shared platform resources are described in Section 20.19, "Intel® Resource Director Technology (Intel® RDT) Allocation Features."
- Features that enable control over shared platform resources for non-CPU agents are described in Section 20.20, "Intel® Resource Director Technology (Intel® RDT) for Non-CPU Agents."¹

20.1 OVERVIEW OF DEBUG SUPPORT FACILITIES

The following processor facilities support debugging and performance monitoring:

- **Debug exception (#DB)** — Transfers program control to a debug procedure or task when a debug event occurs.
- **Breakpoint exception (#BP)** — See breakpoint instruction (INT3) below.
- **Breakpoint-address registers (DR0 through DR3)** — Specifies the addresses of up to 4 breakpoints.
- **Debug status register (DR6)** — Reports the conditions that were in effect when a debug or breakpoint exception was generated.
- **Debug control register (DR7)** — Specifies the forms of memory or I/O access that cause breakpoints to be generated.
- **T (trap) flag, TSS** — Generates a debug exception (#DB) when an attempt is made to switch to a task with the T flag set in its TSS.
- **RF (resume) flag, EFLAGS register** — Suppresses multiple exceptions to the same instruction.
- **TF (trap) flag, EFLAGS register** — Generates a debug exception (#DB) after every execution of an instruction.

1. Additional information about Intel® RDT can be found in the document titled "Intel® Resource Director Technology Architecture Specification," available here: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>.

- **Breakpoint instruction (INT3)** — Generates a breakpoint exception (#BP) that transfers program control to the debugger procedure or task. This instruction is an alternative way to set instruction breakpoints. It is especially useful when more than four breakpoints are desired, or when breakpoints are being placed in the source code.
- **Last branch recording facilities** — Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consist of a branch-from and a branch-to instruction address. Send branch records out on the system bus as branch trace messages (BTMs).

These facilities allow a debugger to be called as a separate task or as a procedure in the context of the current program or task. The following conditions can be used to invoke the debugger:

- Task switch to a specific task.
- Execution of the breakpoint instruction.
- Execution of any instruction.
- Execution of an instruction at a specified address.
- Read or write to a specified memory address/range.
- Write to a specified memory address/range.
- Input from a specified I/O address/range.
- Output to a specified I/O address/range.
- Attempt to change the contents of a debug register.

20.2 DEBUG REGISTERS

Eight debug registers (see Figure 20-1 for 32-bit operation and Figure 20-2 for 64-bit operation) control the debug operation of the processor. These registers can be written to and read using the move to/from debug register form of the MOV instruction. A debug register may be the source or destination operand for one of these instructions.

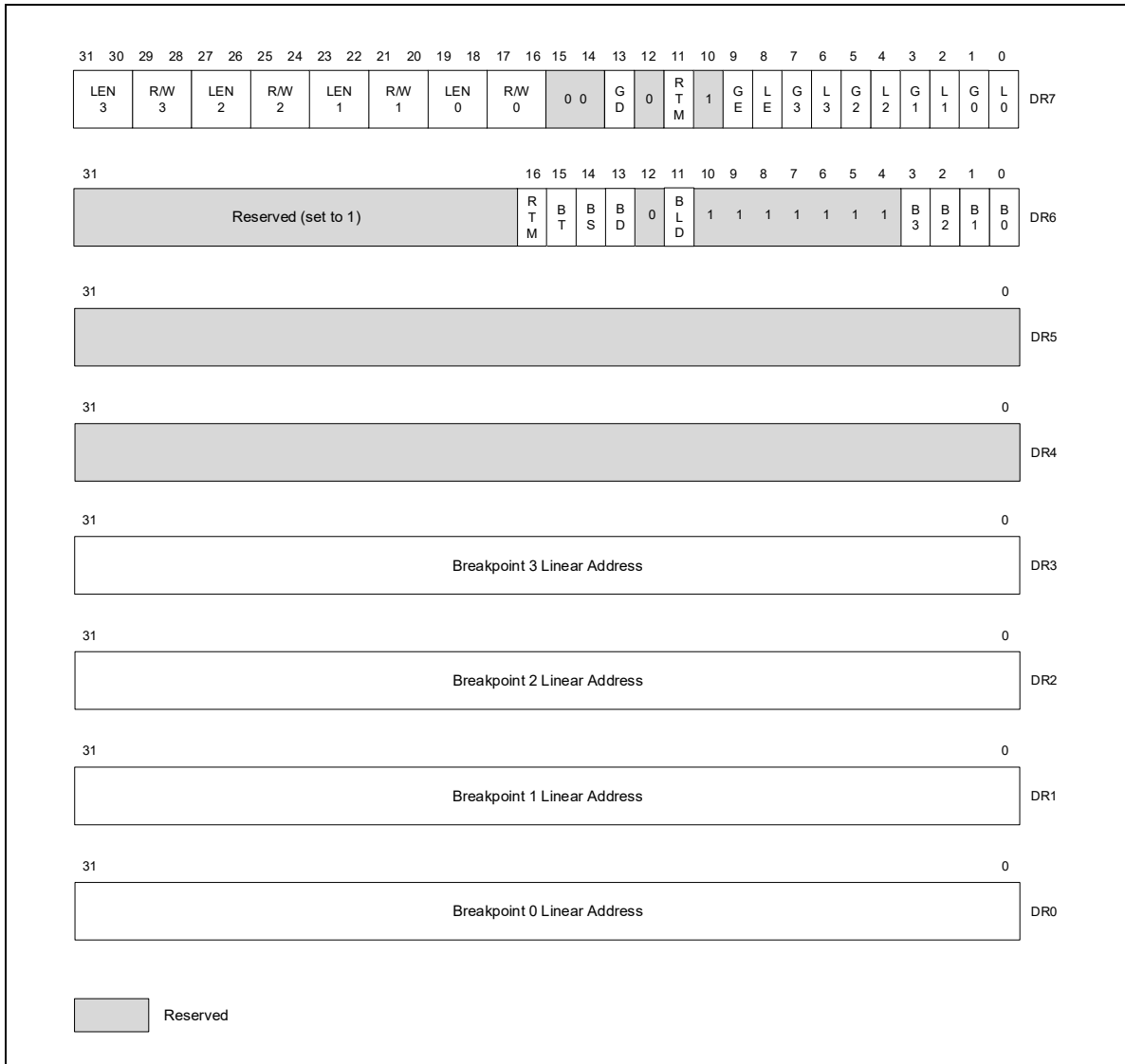


Figure 20-1. Debug Registers

Debug registers are privileged resources; a MOV instruction that accesses these registers can only be executed in real-address mode, in SMM or in protected mode at a CPL of 0. An attempt to read or write the debug registers from any other privilege level generates a general-protection exception (#GP).

The primary function of the debug registers is to set up and monitor from 1 to 4 breakpoints, numbered 0 through 3. For each breakpoint, the following information can be specified:

- The linear address where the breakpoint is to occur.
- The length of the breakpoint location: 1, 2, 4, or 8 bytes (refer to the notes in Section 20.2.4).
- The operation that must be performed at the address for a debug exception to be generated.
- Whether the breakpoint is enabled.
- Whether the breakpoint condition was present when the debug exception was generated.

The following paragraphs describe the functions of flags and fields in the debug registers.

20.2.1 Debug Address Registers (DR0-DR3)

Each of the debug-address registers (DR0 through DR3) holds the 32-bit linear address of a breakpoint (see Figure 20-1). Breakpoint comparisons are made before physical address translation occurs. The contents of debug register DR7 further specifies breakpoint conditions.

20.2.2 Debug Registers DR4 and DR5

Debug registers DR4 and DR5 are reserved when debug extensions are enabled (when the DE flag in control register CR4 is set) and attempts to reference the DR4 and DR5 registers cause invalid-opcode exceptions (#UD). When debug extensions are not enabled (when the DE flag is clear), these registers are aliased to debug registers DR6 and DR7.

20.2.3 Debug Status Register (DR6)

The debug status register (DR6) reports debug conditions that were sampled at the time the last debug exception was generated (see Figure 20-1). Updates to this register only occur when an exception is generated. The flags in this register show the following information:

- **B0 through B3 (breakpoint condition detected) flags (bits 0 through 3)** — Indicates (when set) that its associated breakpoint condition was met when a debug exception was generated. These flags are set if the condition described for each breakpoint by the LEN_n and R/W_n flags in debug control register DR7 is true. They may or may not be set if the breakpoint is not enabled by the Ln or the Gn flags in register DR7. Therefore on a #DB, a debug handler should check only those B0-B3 bits which correspond to an enabled breakpoint.
- **BLD (bus-lock detected) flag (bit 11)** — Indicates (when **clear**) that the debug exception was triggered by the assertion of a bus lock when $CPL > 0$ and OS bus-lock detection was enabled (see Section 20.3.1.6). Other debug exceptions do not modify this bit. To avoid confusion in identifying debug exceptions, software debug-exception handlers should set bit 11 to 1 before returning. (Software that never enables OS bus-lock detection need not do this as $DR6[11] = 1$ following reset.) This bit is always 1 if the processor does not support OS bus-lock detection.
- **BD (debug register access detected) flag (bit 13)** — Indicates that the next instruction in the instruction stream accesses one of the debug registers (DR0 through DR7). This flag is enabled when the GD (general detect) flag in debug control register DR7 is set. See Section 20.2.4, “Debug Control Register (DR7),” for further explanation of the purpose of this flag.
- **BS (single step) flag (bit 14)** — Indicates (when set) that the debug exception was triggered by the single-step execution mode (enabled with the TF flag in the EFLAGS register). The single-step mode is the highest-priority debug exception. When the BS flag is set, any of the other debug status bits also may be set.
- **BT (task switch) flag (bit 15)** — Indicates (when set) that the debug exception resulted from a task switch where the T flag (debug trap flag) in the TSS of the target task was set. See Section 10.2.1, “Task-State Segment (TSS),” for the format of a TSS. There is no flag in debug control register DR7 to enable or disable this exception; the T flag of the TSS is the only enabling flag.
- **RTM (restricted transactional memory) flag (bit 16)** — Indicates (when **clear**) that a debug exception (#DB) or breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 20.3.3). This bit is set for any other debug exception (including all those that occur when advanced debugging of RTM transactional regions is not enabled). This bit is always 1 if the processor does not support RTM.

Certain debug exceptions may clear bits 0-3. The remaining contents of the DR6 register are never cleared by the processor. To avoid confusion in identifying debug exceptions, debug handlers should clear the register (except bit 16, which they should set) before returning to the interrupted task.

20.2.4 Debug Control Register (DR7)

The debug control register (DR7) enables or disables breakpoints and sets breakpoint conditions (see Figure 20-1). The flags and fields in this register control the following things:

- **L0 through L3 (local breakpoint enable) flags (bits 0, 2, 4, and 6)** — Enables (when set) the breakpoint condition for the associated breakpoint for the current task. When a breakpoint condition is detected and its associated *L_n* flag is set, a debug exception is generated. The processor automatically clears these flags on every task switch to avoid unwanted breakpoint conditions in the new task.
- **G0 through G3 (global breakpoint enable) flags (bits 1, 3, 5, and 7)** — Enables (when set) the breakpoint condition for the associated breakpoint for all tasks. When a breakpoint condition is detected and its associated *G_n* flag is set, a debug exception is generated. The processor does not clear these flags on a task switch, allowing a breakpoint to be enabled for all tasks.
- **LE and GE (local and global exact breakpoint enable) flags (bits 8, 9)** — This feature is not supported in the P6 family processors, later IA-32 processors, and Intel 64 processors. When set, these flags cause the processor to detect the exact instruction that caused a data breakpoint condition. For backward and forward compatibility with other Intel processors, we recommend that the LE and GE flags be set to 1 if exact breakpoints are required.
- **RTM (restricted transactional memory) flag (bit 11)** — Enables (when set) advanced debugging of RTM transactional regions (see Section 20.3.3). This advanced debugging is enabled only if IA32_DEBUGCTL.RTM is also set.
- **GD (general detect enable) flag (bit 13)** — Enables (when set) debug-register protection, which causes a debug exception to be generated prior to any MOV instruction that accesses a debug register. When such a condition is detected, the BD flag in debug status register DR6 is set prior to generating the exception. This condition is provided to support in-circuit emulators.

When the emulator needs to access the debug registers, emulator software can set the GD flag to prevent interference from the program currently executing on the processor.

The processor clears the GD flag upon entering to the debug exception handler, to allow the handler access to the debug registers.

- **R/W0 through R/W3 (read/write) fields (bits 16, 17, 20, 21, 24, 25, 28, and 29)** — Specifies the breakpoint condition for the corresponding breakpoint. The DE (debug extensions) flag in control register CR4 determines how the bits in the R/W_{*n*} fields are interpreted. When the DE flag is set, the processor interprets bits as follows:

- 00 — Break on instruction execution only.
- 01 — Break on data writes only.
- 10 — Break on I/O reads or writes.
- 11 — Break on data reads or writes but not instruction fetches.

When the DE flag is clear, the processor interprets the R/W_{*n*} bits the same as for the Intel386™ and Intel486™ processors, which is as follows:

- 00 — Break on instruction execution only.
- 01 — Break on data writes only.
- 10 — Undefined.
- 11 — Break on data reads or writes but not instruction fetches.

- **LEN0 through LEN3 (Length) fields (bits 18, 19, 22, 23, 26, 27, 30, and 31)** — Specify the size of the memory location at the address specified in the corresponding breakpoint address register (DR0 through DR3). These fields are interpreted as follows:

- 00 — 1-byte length.
- 01 — 2-byte length.
- 10 — Undefined (or 8 byte length, see note below).
- 11 — 4-byte length.

If the corresponding R/W_{*n*} field in register DR7 is 00 (instruction execution), then the LEN_{*n*} field should also be 00. The effect of using other lengths is undefined. See Section 20.2.5, “Breakpoint Field Recognition,” below.

NOTES

For Pentium® 4 and Intel® Xeon® processors with a CPUID signature corresponding to family 15 (model 3, 4, and 6), break point conditions permit specifying 8-byte length on data read/write with an of encoding 10B in the LEN_n field.

Encoding 10B is also supported in processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture, the respective CPUID signatures corresponding to family 6, model 15, and family 6, DisplayModel value 23 (see the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). The Encoding 10B is supported in processors based on Intel Atom® microarchitecture, with CPUID signature of family 6, DisplayModel value 1CH. The encoding 10B is undefined for other processors.

20.2.5 Breakpoint Field Recognition

Breakpoint address registers (debug registers DR0 through DR3) and the LEN_n fields for each breakpoint define a range of sequential byte addresses for a data or I/O breakpoint. The LEN_n fields permit specification of a 1-, 2-, 4- or 8-byte range, beginning at the linear address specified in the corresponding debug register (DR_n). Two-byte ranges must be aligned on word boundaries; 4-byte ranges must be aligned on doubleword boundaries, 8-byte ranges must be aligned on quadword boundaries. I/O addresses are zero-extended (from 16 to 32 bits, for comparison with the breakpoint address in the selected debug register). These requirements are enforced by the processor; it uses LEN_n field bits to mask the lower address bits in the debug registers. Unaligned data or I/O breakpoint addresses do not yield valid results.

A data breakpoint for reading or writing data is triggered if any of the bytes participating in an access is within the range defined by a breakpoint address register and its LEN_n field. Table 20-1 provides an example setup of debug registers and data accesses that would subsequently trap or not trap on the breakpoints.

A data breakpoint for an unaligned operand can be constructed using two breakpoints, where each breakpoint is byte-aligned and the two breakpoints together cover the operand. The breakpoints generate exceptions only for the operand, not for neighboring bytes.

Instruction breakpoint addresses must have a length specification of 1 byte (the LEN_n field is set to 00). Instruction breakpoints for other operand sizes are undefined. The processor recognizes an instruction breakpoint address only when it points to the first byte of an instruction. If the instruction has prefixes, the breakpoint address must point to the first prefix.

Table 20-1. Breakpoint Examples

Debug Register Setup			
Debug Register	R/W _n	Breakpoint Address	LEN _n
DR0	R/W0 = 11 (Read/Write)	A0001H	LEN0 = 00 (1 byte)
DR1	R/W1 = 01 (Write)	A0002H	LEN1 = 00 (1 byte)
DR2	R/W2 = 11 (Read/Write)	B0002H	LEN2 = 01) (2 bytes)
DR3	R/W3 = 01 (Write)	C0000H	LEN3 = 11 (4 bytes)
Data Accesses			
Operation		Address	Access Length (In Bytes)
Data operations that trap			
- Read or write		A0001H	1
- Read or write		A0001H	2
- Write		A0002H	1
- Write		A0002H	2
- Read or write		B0001H	4
- Read or write		B0002H	1
- Read or write		B0002H	2
- Write		C0000H	4
- Write		C0001H	2
- Write		C0003H	1

Table 20-1. Breakpoint Examples (Contd.)

Debug Register Setup			
Debug Register	R/Wn	Breakpoint Address	LENn
Data operations that do not trap - Read or write - Read - Read or write - Read or write - Read - Read or write		A0000H	1
		A0002H	1
		A0003H	4
		B0000H	2
		C0000H	2
		C0004H	4

20.2.6 Debug Registers and Intel® 64 Processors

For Intel 64 architecture processors, debug registers DR0–DR7 are 64 bits. In 16-bit or 32-bit modes (protected mode and compatibility mode), writes to a debug register fill the upper 32 bits with zeros. Reads from a debug register return the lower 32 bits. In 64-bit mode, MOV DRn instructions read or write all 64 bits. Operand-size prefixes are ignored.

In 64-bit mode, the upper 32 bits of DR6 and DR7 are reserved and must be written with zeros. Writing 1 to any of the upper 32 bits results in a #GP(0) exception (see Figure 20-2). All 64 bits of DR0–DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0–DR3 are in the linear-address limits of the processor implementation (address matching is supported only on valid addresses generated by the processor implementation). Breakpoint conditions for 8-byte memory read/writes are supported in all modes.

20.3 DEBUG EXCEPTIONS

The Intel 64 and IA-32 architectures dedicate two interrupt vectors to handling debug exceptions: vector 1 (debug exception, #DB) and vector 3 (breakpoint exception, #BP). The following sections describe how these exceptions are generated and typical exception handler operations.

20.3.1 Debug Exception (#DB)—Interrupt Vector 1

The debug-exception handler is usually a debugger program or part of a larger software system. The processor generates a debug exception for any of several conditions. The debugger checks flags in the DR6 and DR7 registers to determine which condition caused the exception and which other conditions might apply. Table 20-2 shows the states of these flags following the generation of each kind of breakpoint condition.

Instruction-breakpoint and general-detect condition (see Section 20.3.1.3, “General-Detect Exception Condition”) result in faults; other debug-exception conditions result in traps. The debug exception may report one or both at one time. The following sections describe each class of debug exception.

The INT1 instruction generates a debug exception as a trap. Hardware vendors may use the INT1 instruction for hardware debug. For that reason, Intel recommends software vendors instead use the INT3 instruction for software breakpoints.

See also: Chapter 7, “Interrupt 1—Debug Exception (#DB),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

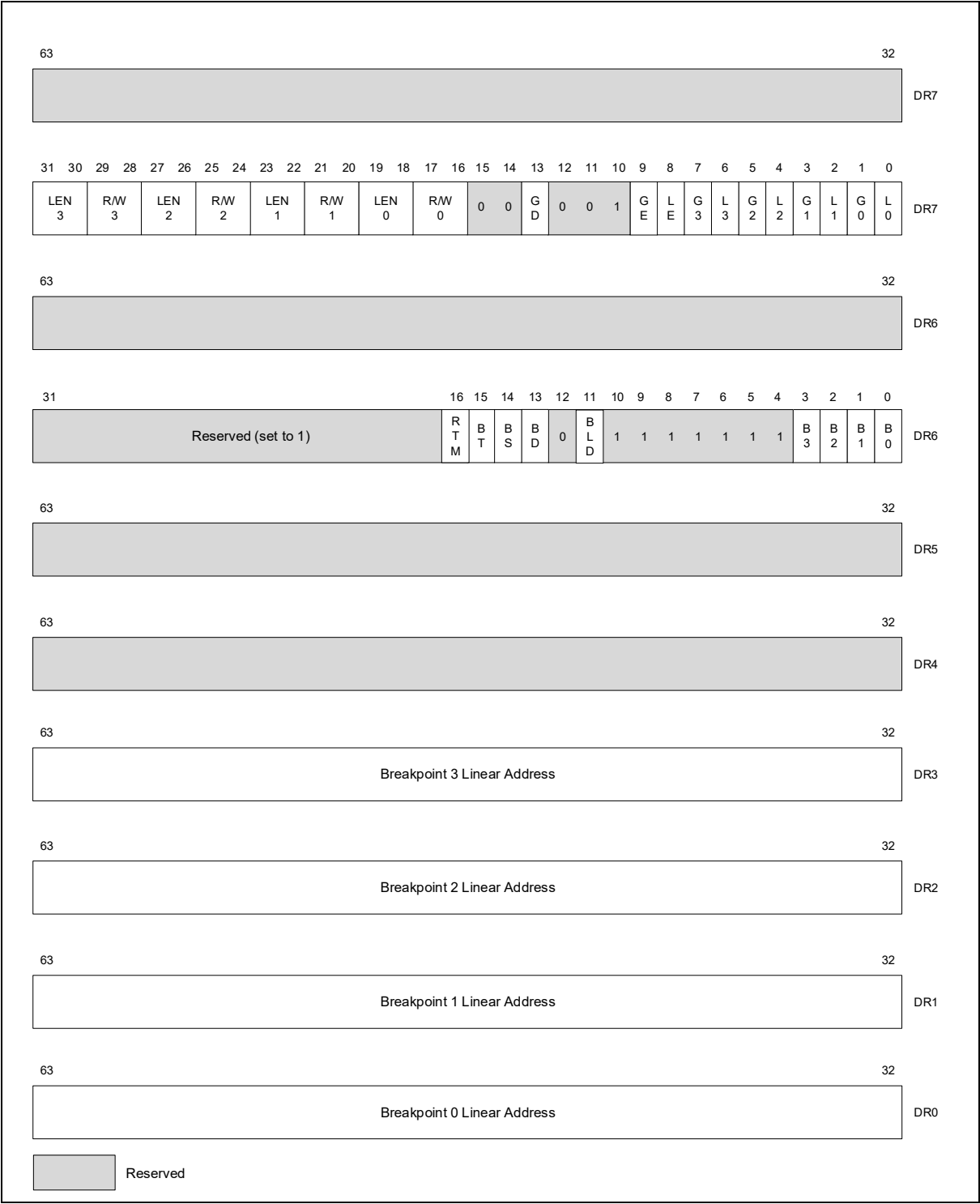


Figure 20-2. DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture

Table 20-2. Debug Exception Conditions

Debug or Breakpoint Condition	DR6 Flags Tested	DR7 Flags Tested	Exception Class
Single-step trap	BS = 1		Trap
Instruction breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 0	Fault
Data write breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 1	Trap
I/O read or write breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 2	Trap
Data read or write (but not instruction fetches), at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 3	Trap
General detect fault, resulting from an attempt to modify debug registers (usually in conjunction with in-circuit emulation)	BD = 1	None	Fault
Task switch	BT = 1	None	Trap
INT1 instruction	None	None	Trap

20.3.1.1 Instruction-Breakpoint Exception Condition

The processor reports an instruction breakpoint when it attempts to execute an instruction at an address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect instruction execution (R/W flag is set to 0). Upon reporting the instruction breakpoint, the processor generates a fault-class, debug exception (#DB) before it executes the target instruction for the breakpoint.

Instruction breakpoints are the highest priority debug exceptions. They are serviced before any other exceptions detected during the decoding or execution of an instruction. However, if an instruction breakpoint is placed on an instruction located immediately after a POP SS/MOV SS instruction, the breakpoint will be suppressed as if EFLAGS.RF were 1 (see the next paragraph and Section 7.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).

Because the debug exception for an instruction breakpoint is generated before the instruction is executed, if the instruction breakpoint is not removed by the exception handler; the processor will detect the instruction breakpoint again when the instruction is restarted and generate another debug exception. To prevent looping on an instruction breakpoint, the Intel 64 and IA-32 architectures provide the RF flag (resume flag) in the EFLAGS register (see Section 2.3, “System Flags and Fields in the EFLAGS Register,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). When the RF flag is set, the processor ignores instruction breakpoints.

All Intel 64 and IA-32 processors manage the RF flag as follows. The RF Flag is cleared at the start of the instruction after the check for instruction breakpoints, CS limit violations, and FP exceptions. Task Switches and IRETD/IRETQ instructions transfer the RF image from the TSS/stack to the EFLAGS register.

When calling an event handler, Intel 64 and IA-32 processors establish the value of the RF flag in the EFLAGS image pushed on the stack:

- For any fault-class exception except a debug exception generated in response to an instruction breakpoint, the value pushed for RF is 1.
- For any interrupt arriving after any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For any trap-class exception generated by any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For other cases, the value pushed for RF is the value that was in EFLAG.RF at the time the event handler was called. This includes:
 - Debug exceptions generated in response to instruction breakpoints
 - Hardware-generated interrupts arriving between instructions (including those arriving after the last iteration of a repeated string instruction)

- Trap-class exceptions generated after an instruction completes (including those generated after the last iteration of a repeated string instruction)
- Software-generated interrupts (RF is pushed as 0, since it was cleared at the start of the software interrupt)

As noted above, the processor does not set the RF flag prior to calling the debug exception handler for debug exceptions resulting from instruction breakpoints. The debug exception handler can prevent recurrence of the instruction breakpoint by setting the RF flag in the EFLAGS image on the stack. If the RF flag in the EFLAGS image is set when the processor returns from the exception handler, it is copied into the RF flag in the EFLAGS register by IRETD/IRETQ or a task switch that causes the return. The processor then ignores instruction breakpoints for the duration of the next instruction. (Note that the POPF, POPFD, and IRET instructions do not transfer the RF image into the EFLAGS register.) Setting the RF flag does not prevent other types of debug-exception conditions (such as, I/O or data breakpoints) from being detected, nor does it prevent non-debug exceptions from being generated.

For the Pentium processor, when an instruction breakpoint coincides with another fault-type exception (such as a page fault), the processor may generate one spurious debug exception after the second exception has been handled, even though the debug exception handler set the RF flag in the EFLAGS image. To prevent a spurious exception with Pentium processors, all fault-class exception handlers should set the RF flag in the EFLAGS image.

20.3.1.2 Data Memory and I/O Breakpoint Exception Conditions

Data memory and I/O breakpoints are reported when the processor attempts to access a memory or I/O address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect data or I/O accesses (R/W flag is set to 1, 2, or 3). The processor generates the exception after it executes the instruction that made the access, so these breakpoint condition causes a trap-class exception to be generated.

Because data breakpoints are traps, an instruction that writes memory overwrites the original data before the debug exception generated by a data breakpoint is generated. If a debugger needs to save the contents of a write breakpoint location, it should save the original contents before setting the breakpoint. The handler can report the saved value after the breakpoint is triggered. The address in the debug registers can be used to locate the new value stored by the instruction that triggered the breakpoint.

If a data breakpoint is detected during an iteration of a string instruction executed with fast-string operation (see Section 7.3.9.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1), delivery of the resulting debug exception may be delayed until completion of the corresponding group of iterations.

Intel486 and later processors ignore the GE and LE flags in DR7. In Intel386 processors, exact data breakpoint matching does not occur unless it is enabled by setting the LE and/or the GE flags.

For repeated INS and OUTS instructions that generate an I/O-breakpoint debug exception, the processor generates the exception after the completion of the first iteration. Repeated INS and OUTS instructions generate a data-breakpoint debug exception after the iteration in which the memory address breakpoint location is accessed.

If an execution of the MOV or POP instruction loads the SS register and encounters a data breakpoint, the resulting debug exception is delivered after completion of the next instruction (the one after the MOV or POP).

Any pending data or I/O breakpoints are lost upon delivery of an exception. For example, if a machine-check exception (#MC) occurs following an instruction that encounters a data breakpoint (but before the resulting debug exception is delivered), the data breakpoint is lost. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, the data breakpoint is lost if the next instruction causes a fault.

Delivery of events due to INT *n*, INT3, or INTO does not cause a loss of data breakpoints. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, and the next instruction is software interrupt (INT *n*, INT3, or INTO), a debug exception (#DB) resulting from a data breakpoint will be delivered after the transition to the software-interrupt handler. The #DB handler should account for the fact that the #DB may have been delivered after a invocation of a software-interrupt handler, and in particular that the CPL may have changed between recognition of the data breakpoint and delivery of the #DB.

20.3.1.3 General-Detect Exception Condition

When the GD flag in DR7 is set, the general-detect debug exception occurs when a program attempts to access any of the debug registers (DR0 through DR7) at the same time they are being used by another application, such as an emulator or debugger. This protection feature guarantees full control over the debug registers when required. The

debug exception handler can detect this condition by checking the state of the BD flag in the DR6 register. The processor generates the exception before it executes the MOV instruction that accesses a debug register, which causes a fault-class exception to be generated.

20.3.1.4 Single-Step Exception Condition

The processor generates a single-step debug exception if (while an instruction is being executed) it detects that the TF flag in the EFLAGS register is set. The exception is a trap-class exception, because the exception is generated after the instruction is executed. The processor will not generate this exception after the instruction that sets the TF flag. For example, if the POPF instruction is used to set the TF flag, a single-step trap does not occur until after the instruction that follows the POPF instruction.

The processor clears the TF flag before calling the exception handler. If the TF flag was set in a TSS at the time of a task switch, the exception occurs after the first instruction is executed in the new task.

The TF flag normally is not cleared by privilege changes inside a task. The INT *n*, INT3, and INTO instructions, however, do clear this flag. Therefore, software debuggers that single-step code must recognize and emulate INT *n* or INTO instructions rather than executing them directly. To maintain protection, the operating system should check the CPL after any single-step trap to see if single stepping should continue at the current privilege level.

The interrupt priorities guarantee that, if an external interrupt occurs, single stepping stops. When both an external interrupt and a single-step interrupt occur together, the single-step interrupt is processed first. This operation clears the TF flag. After saving the return address or switching tasks, the external interrupt input is examined before the first instruction of the single-step handler executes. If the external interrupt is still pending, then it is serviced. The external interrupt handler does not run in single-step mode. To single step an interrupt handler, single step an INT *n* instruction that calls the interrupt handler.

If an occurrence of the MOV or POP instruction loads the SS register executes with EFLAGS.TF = 1, no single-step debug exception occurs following the MOV or POP instruction.

20.3.1.5 Task-Switch Exception Condition

The processor generates a debug exception after a task switch if the T flag of the new task's TSS is set. This exception is generated after program control has passed to the new task, and prior to the execution of the first instruction of that task. The exception handler can detect this condition by examining the BT flag of the DR6 register.

If entry 1 (#DB) in the IDT is a task gate, the T bit of the corresponding TSS should not be set. Failure to observe this rule will put the processor in a loop.

20.3.1.6 OS Bus-Lock Detection

OS bus-lock detection is a feature that causes the processor to generate a debug exception (called a **bus-lock detection debug exception**) if it detects that a bus lock has been asserted (see Section 11.1.2). Such an exception is a trap-class exception, because it is generated after execution of an instruction that asserts a bus lock. The exception thus does not prevent assertion of the bus lock. Delivery of a bus-lock detection debug exception clears DR6.BLD.

Software can enable OS bus-lock detection by setting IA32_DEBUGCTL.BLD[bit 2]. Bus-lock detection debug exceptions occur only if CPL > 0.

20.3.2 Breakpoint Exception (#BP)—Interrupt Vector 3

The breakpoint exception (interrupt 3) is caused by execution of an INT3 instruction. See Chapter 7, "Interrupt 3—Breakpoint Exception (#BP)." Debuggers use breakpoint exceptions in the same way that they use the breakpoint registers; that is, as a mechanism for suspending program execution to examine registers and memory locations. With earlier IA-32 processors, breakpoint exceptions are used extensively for setting instruction breakpoints.

With the Intel386 and later IA-32 processors, it is more convenient to set breakpoints with the breakpoint-address registers (DR0 through DR3). However, the breakpoint exception still is useful for breakpointing debuggers,

because a breakpoint exception can call a separate exception handler. The breakpoint exception is also useful when it is necessary to set more breakpoints than there are debug registers or when breakpoints are being placed in the source code of a program under development.

20.3.3 Debug Exceptions, Breakpoint Exceptions, and Restricted Transactional Memory (RTM)

Chapter 16, “Programming with Intel® AVX10,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, describes Restricted Transactional Memory (RTM). This is an instruction-set interface that allows software to identify **transactional regions** (or critical sections) using the XBEGIN and XEND instructions.

Execution of an RTM transactional region begins with an XBEGIN instruction. If execution of the region successfully reaches an XEND instruction, the processor ensures that all memory operations performed within the region appear to have occurred instantaneously when viewed from other logical processors. Execution of an RTM transaction region does not succeed if the processor cannot commit the updates atomically. When this happens, the processor rolls back the execution, a process referred to as a **transactional abort**. In this case, the processor discards all updates performed in the region, restores architectural state to appear as if the execution had not occurred, and resumes execution at a fallback instruction address that was specified with the XBEGIN instruction.

If debug exception (#DB) or breakpoint exception (#BP) occurs within an RTM transaction region, a transactional abort occurs, the processor sets EAX[4], and no exception is delivered.

Software can enable **advanced debugging of RTM transactional regions** by setting DR7.RTM[bit 11] and IA32_DEBUGCTL.RTM[bit 15]. If these bits are both set, the transactional abort caused by a #DB or #BP within an RTM transaction region does **not** resume execution at the fallback instruction address specified with the XBEGIN instruction that begin the region. Instead, execution is resumed at that XBEGIN instruction, and a #DB is delivered. (A #DB is delivered even if the transactional abort was caused by a #BP.) Such a #DB will clear DR6.RTM[bit 16] (all other debug exceptions set DR6[16]).

20.4 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING OVERVIEW

P6 family processors introduced the ability to set breakpoints on taken branches, interrupts, and exceptions, and to single-step from one branch to the next. This capability has been modified and extended in the Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel Atom® processors to allow logging of branch trace messages in a branch trace store (BTS) buffer in memory.

See the following sections for processor specific implementation of last branch, interrupt, and exception recording:

- Section 20.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors).”
- Section 20.6, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Microarchitecture.”
- Section 20.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture.”
- Section 20.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture.”
- Section 20.11, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Haswell Microarchitecture.”
- Section 20.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture.”
- Section 20.14, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors).”
- Section 20.15, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors).”
- Section 20.16, “Last Branch, Interrupt, and Exception Recording (P6 Family Processors).”

The following subsections of Section 20.4 describe common features of profiling branches. These features are generally enabled using the IA32_DEBUGCTL MSR (older processor may have implemented a subset or model-specific features, see definitions of MSR_DEBUGCTLA, MSR_DEBUGCTLB, MSR_DEBUGCTL).

20.4.1 IA32_DEBUGCTL MSR

The **IA32_DEBUGCTL** MSR provides bit field controls to enable debug trace interrupts, debug trace stores, trace messages enable, single stepping on branches, last branch record recording, and to control freezing of LBR stack or performance counters on a PMI request. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 20-3 for the MSR layout and the bullets below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the Section 20.5.1, “LBR Stack” (Intel® Core™2 Duo and Intel Atom® processor family) and Section 20.9.1, “LBR Stack” (processors based on Nehalem microarchitecture).
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 20.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
- **BLD (bus-lock detection) flag (bit 2)** — If this bit is set, OS bus-lock detection is enabled when CPL > 0. See Section 20.3.1.6.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 20.4.4, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 20.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bit 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 20.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

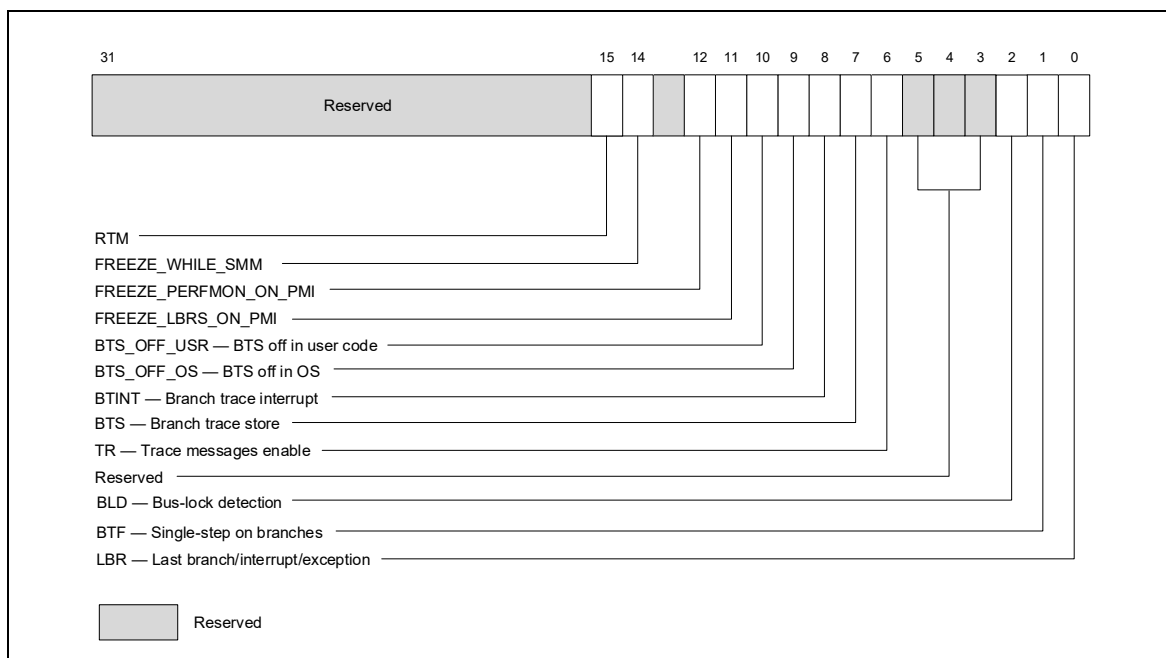


Figure 20-3. IA32_DEBUGCTL MSR for Processors Based on Intel® Core™ Microarchitecture

- **BTS_OFF_OS (branch trace off in privileged code) flag (bit 9)** — When set, BTS or BTM is skipped if CPL is 0. See Section 20.13.2.
- **BTS_OFF_USR (branch trace off in user code) flag (bit 10)** — When set, BTS or BTM is skipped if CPL is greater than 0. See Section 20.13.2.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — When set, the LBR stack is frozen on a hardware PMI request (e.g., when a counter overflows and is configured to trigger PMI). See Section 20.4.7 for details.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — When set, the performance counters (IA32_PMCx and IA32_FIXED_CTRx) are frozen on a PMI request. See Section 20.4.7 for details.
- **FREEZE_WHILE_SMM (bit 14)** — If this bit is set, upon the delivery of an SMI, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler. If Intel Thread Director support was enabled before transferring control to the SMI handler, then the processor will also reset the Intel Thread Director history (see Section 17.6.11 for more details about Intel Thread Director enable, reset, and history reset operations).
Subsequently, the enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its service. If Intel Thread Director support is enabled when RSM is executed, then the processor resets the Intel Thread Director history.
Note that system software must check if the processor supports the IA32_DEBUGCTL.FREEZE_WHILE_SMM control bit. IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 22.8 for details of detecting the presence of IA32_PERF_CAPABILITIES MSR.
- **RTM (bit 15)** — If this bit is set, advanced debugging of RTM transactional regions is enabled if DR7.RTM is also set. See Section 20.3.3.

20.4.2 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag (bit 0) in the IA32_DEBUGCTL MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions) in the LBR stack MSRs.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler. This action does not clear previously stored LBR stack MSRs.

A debugger can use the linear addresses in the LBR stack to re-set breakpoints in the breakpoint address registers (DR0 through DR3). This allows a backward trace from the manifestation of a particular bug toward its source.

On some processors, if the LBR flag is cleared and TR flag in the IA32_DEBUGCTL MSR remains set, the processor will continue to update LBR stack MSRs. This is because those processors use the entries in the LBR stack in the process of generating BTM/BTS records. A #DB does not automatically clear the TR flag.

20.4.3 Single-Stepping on Branches

When software sets both the BTF flag (bit 1) in the IA32_DEBUGCTL MSR and the TF flag in the EFLAGS register, the processor generates a single-step debug exception only after instructions that cause a branch.¹ This mechanism allows a debugger to single-step on control transfers caused by branches. This “branch single stepping” helps isolate a bug to a particular block of code before instruction single-stepping further narrows the search. The processor clears the BTF flag when it generates a debug exception. The debugger must set the BTF flag before resuming program execution to continue single-stepping on branches.

1. Executions of CALL, IRET, and JMP that cause task switches never cause single-step debug exceptions (regardless of the value of the BTF flag). A debugger desiring debug exceptions on switches to a task should set the T flag (debug trap flag) in the TSS of that task. See Section 10.2.1, “Task-State Segment (TSS).”

20.4.4 Branch Trace Messages

Setting the TR flag (bit 6) in the IA32_DEBUGCTL MSR enables branch trace messages (BTMs). Thereafter, when the processor detects a branch, exception, or interrupt, it sends a branch record out on the system bus as a BTM. A debugging device that is monitoring the system bus can read these messages and synchronize operations with taken branch, interrupt, and exception events.

When interrupts or exceptions occur in conjunction with a taken branch, additional BTMs are sent out on the bus, as described in Section 20.4.2, “Monitoring Branches, Exceptions, and Interrupts.”

For the P6 processor family, Pentium M processor family, and processors based on Intel Core microarchitecture, TR and LBR bits can not be set at the same time due to hardware limitation. The content of LBR stack is undefined when TR is set.

For processors with Intel NetBurst microarchitecture, Intel Atom processors, and Intel Core and related Intel Xeon processors both starting with the Nehalem microarchitecture, the processor can collect branch records in the LBR stack and at the same time send/store BTMs when both the TR and LBR flags are set in the IA32_DEBUGCTL MSR (or the equivalent MSR_DEBUGCTLA, MSR_DEBUGCTLB).

The following exception applies:

- BTM may not be observable on Intel Atom processor families that do not provide an externally visible system bus (i.e., processors based on the Silvermont microarchitecture or later).

20.4.4.1 Branch Trace Message Visibility

Branch trace message (BTM) visibility is implementation specific and limited to systems with a front side bus (FSB). BTMs may not be visible to newer system link interfaces or a system bus that deviates from a traditional FSB.

20.4.5 Branch Trace Store (BTS)

A trace of taken branches, interrupts, and exceptions is useful for debugging code by providing a method of determining the decision path taken to reach a particular code location. The LBR flag (bit 0) of IA32_DEBUGCTL provides a mechanism for capturing records of taken branches, interrupts, and exceptions and saving them in the last branch record (LBR) stack MSRs, setting the TR flag for sending them out onto the system bus as BTMs. The branch trace store (BTS) mechanism provides the additional capability of saving the branch records in a memory-resident BTS buffer, which is part of the DS save area. The BTS buffer can be configured to be circular so that the most recent branch records are always available or it can be configured to generate an interrupt when the buffer is nearly full so that all the branch records can be saved. The BTINT flag (bit 8) can be used to enable the generation of interrupt when the BTS buffer is full. See Section 20.4.9.2, “Setting Up the DS Save Area,” for additional details.

Setting this flag (BTS) alone can greatly reduce the performance of the processor. CPL-qualified branch trace storing mechanism can help mitigate the performance impact of sending/logging branch trace messages.

20.4.6 CPL-Qualified Branch Trace Mechanism

CPL-qualified branch trace mechanism is available to a subset of Intel 64 and IA-32 processors that support the branch trace storing mechanism. The processor supports the CPL-qualified branch trace mechanism if CPUID.01H:ECX[4] = 1.

The CPL-qualified branch trace mechanism is described in Section 20.4.9.4. System software can selectively specify CPL qualification to not send/store Branch Trace Messages associated with a specified privilege level. Two bit fields, BTS_OFF_USR (bit 10) and BTS_OFF_OS (bit 9), are provided in the debug control register to specify the CPL of BTMs that will not be logged in the BTS buffer or sent on the bus.

20.4.7 Freezing LBR and Performance Counters on PMI

Many issues may generate a performance monitoring interrupt (PMI); a PMI service handler will need to determine cause to handle the situation. Two capabilities that allow a PMI service routine to improve branch tracing and performance monitoring are available for processors supporting architectural performance monitoring version 2 or

greater (i.e., CPUID.0AH:EAX[7:0] > 1). These capabilities provides the following interface in IA32_DEBUGCTL to reduce runtime overhead of PMI servicing, profiler-contributed skew effects on analysis or counter metrics:

- **Freezing LBRs on PMI (bit 11)** — Allows the PMI service routine to ensure the content in the LBR stack are associated with the target workload and not polluted by the branch flows of handling the PMI. Depending on the version ID enumerated by CPUID.0AH:EAX.VERSION[7:0], two flavors are supported:
 - Legacy Freeze_LBR_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID > 1. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the LBR is frozen on the overflowed condition of the buffer area, the processor clears the LBR bit (bit 0) in IA32_DEBUGCTL. Software must then re-enable IA32_DEBUGCTL.LBR to resume recording branches. When using this feature, software should be careful about writes to IA32_DEBUGCTL to avoid re-enabling LBRs by accident if they were just disabled.
 - Streamlined Freeze_LBR_on_PMI is supported for ArchPerfMonVerID >= 4. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the processor behaves as follows:
 - sets IA32_PERF_GLOBAL_STATUS.LBR_Frz = 1 to disable recording, but does not change the LBR bit (bit 0) in IA32_DEBUGCTL. The LBRs are frozen on the overflowed condition of the buffer area.
- **Freezing PMCs on PMI (bit 12)** — Allows the PMI service routine to ensure the content in the performance counters are associated with the target workload and not polluted by the PMI and activities within the PMI service routine. Depending on the version ID enumerated by CPUID.0AH:EAX.VERSION[7:0], two flavors are supported:
 - Legacy Freeze_PerfMon_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID > 1. If IA32_DEBUGCTL.Freeze_PerfMon_On_PMI = 1, the performance counters are frozen on the counter overflowed condition when the processor clears the IA32_PERF_GLOBAL_CTRL MSR (see Figure 22-3). The PMCs affected include both general-purpose counters and fixed-function counters (see Section 22.6.2.1, “Fixed-function Performance Counters”). Software must re-enable counts by writing 1s to the corresponding enable bits in IA32_PERF_GLOBAL_CTRL before leaving a PMI service routine to continue counter operation.
 - Streamlined Freeze_PerfMon_on_PMI is supported for ArchPerfMonVerID >= 4. The processor behaves as follows:
 - sets IA32_PERF_GLOBAL_STATUS.CTR_Frz = 1 to disable counting on a counter overflow condition, but does not change the IA32_PERF_GLOBAL_CTRL MSR.

Freezing LBRs and PMCs on PMIs (both legacy and streamlined operation) occur when one of the following applies:

- A performance counter had an overflow and was programmed to signal a PMI in case of an overflow.
 - For the general-purpose counters; enabling PMI is done by setting bit 20 of the IA32_PERFEVTSELx register.
 - For the fixed-function counters; enabling PMI is done by setting the 3rd bit in the corresponding 4-bit control field of the MSR_PERF_FIXED_CTR_CTRL register (see Figure 22-1) or IA32_FIXED_CTR_CTRL MSR (see Figure 22-2).
- The PEBS buffer is almost full and reaches the interrupt threshold.
- The BTS buffer is almost full and reaches the interrupt threshold.

Table 20-3 compares the interaction of the processor with the PMI handler using the legacy versus streamlined Freeze_PerfMon_On_PMI interface.

Table 20-3. Legacy and Streamlined Operation with Freeze_PerfMon_On_PMI = 1, Counter Overflowed

Legacy Freeze_PerfMon_On_PMI	Streamlined Freeze_PerfMon_On_PMI	Comment
Processor freezes the counters on overflow	Processor freezes the counters on overflow	Unchanged
Processor clears IA32_PERF_GLOBAL_CTRL	Processor set IA32_PERF_GLOBAL_STATUS.CTR_FTZ	
Handler reads IA32_PERF_GLOBAL_STATUS(0x38E) to examine which counter(s) overflowed	mask = RDMSR(0x38E)	Similar
Handler services the PMI	Handler services the PMI	Unchanged
Handler writes 1s to IA32_PERF_GLOBAL_OVF_CTL (0x390)	Handler writes mask into IA32_PERF_GLOBAL_OVF_RESET (0x390)	
Processor clears IA32_PERF_GLOBAL_STATUS	Processor clears IA32_PERF_GLOBAL_STATUS	Unchanged
Handler re-enables IA32_PERF_GLOBAL_CTRL	None	Reduced software overhead

20.4.8 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel 64 and IA-32 processor families. However, the number of MSRs in the LBR stack and the valid range of TOS pointer value can vary between different processor families. Table 20-4 lists the LBR stack size and TOS pointer range for several processor families according to the CPUID signatures of DisplayFamily_DisplayModel encoding (see the CPUID instruction in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A).

Table 20-4. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Component of an LBR Entry	Range of TOS Pointer
06_5CH, 06_5FH	32	FROM_IP, TO_IP	0 to 31
06_4EH, 06_5EH, 06_8EH, 06_9EH, 06_55H, 06_66H, 06_7AH, 06_67H, 06_6AH, 06_6CH, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_A5H, 06_A6H, 06_A7H, 06_A8H, 06_86H, 06_8AH, 06_96H, 06_9CH	32	FROM_IP, TO_IP, LBR_INFO ¹	0 to 31
06_3DH, 06_47H, 06_4FH, 06_56H, 06_3CH, 06_45H, 06_46H, 06_3FH, 06_2AH, 06_2DH, 06_3AH, 06_3EH, 06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	16	FROM_IP, TO_IP	0 to 15
06_17H, 06_1DH, 06_0FH	4	FROM_IP, TO_IP	0 to 3
06_37H, 06_4AH, 06_4CH, 06_4DH, 06_5AH, 06_5DH, 06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	8	FROM_IP, TO_IP	0 to 7

NOTES:

1. See Section 20.12.

The last branch recording mechanism tracks not only branch instructions (e.g., JMP, Jcc, LOOP, and CALL instructions), but also other operations that cause a change in the instruction pointer (e.g., external interrupts, traps, and faults). The branch recording mechanisms generally employs a set of MSRs, referred to as last branch record (LBR) stack. The size and exact locations of the LBR stack are generally model-specific (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for model-specific MSR addresses).

- **Last Branch Record (LBR) Stack** — The LBR consists of N pairs of MSRs (N is listed in the LBR stack size column of Table 20-4) that store source and destination address of recent branches (see Figure 20-3):
 - MSR_LASTBRANCH_0_FROM_IP (address is model specific) through the next consecutive (N-1) MSR address store source addresses.
 - MSR_LASTBRANCH_0_TO_IP (address is model specific) through the next consecutive (N-1) MSR address store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant M bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address is model specific) contains an M-bit pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. The valid range of the M-bit POS pointer is given in Table 20-4.

20.4.8.1 LBR Stack and Intel® 64 Processors

LBR MSRs are 64-bits. In 64-bit mode, last branch records store the full address. Outside of 64-bit mode, the upper 32-bits of branch addresses will be stored as 0.

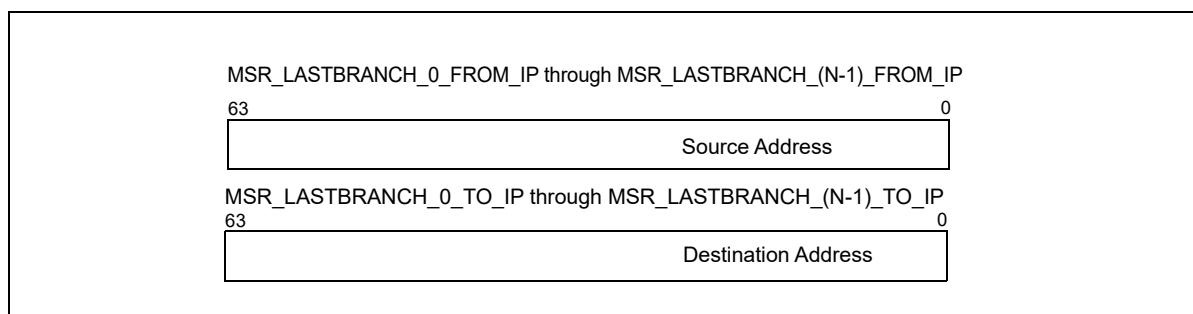


Figure 20-4. 64-bit Address Layout of LBR MSR

Software should query an architectural MSR IA32_PERF_CAPABILITIES[5:0] about the format of the address that is stored in the LBR stack. Four formats are defined by the following encoding:

- **000000B (32-bit record format)** — Stores 32-bit offset in current CS of respective source/destination,
- **000001B (64-bit LIP record format)** — Stores 64-bit linear address of respective source/destination,
- **000010B (64-bit EIP record format)** — Stores 64-bit offset (effective address) of respective source/destination.
- **000011B (64-bit EIP record format) and Flags** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction info is reported in the upper bit of 'FROM' registers in the LBR stack. See LBR stack details below for flag support and definition.
- **000100B (64-bit EIP record format), Flags, and TSX** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction and TSX info are reported in the upper bits of 'FROM' registers in the LBR stack.
- **000101B (64-bit EIP record format), Flags, TSX, and LBR_INFO** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction, TSX, and elapsed cycles since the last LBR update are reported in the LBR_INFO MSR stack.
- **000110B (64-bit LIP record format), Flags, and Cycles** — Stores 64-bit linear address (CS.Base + effective address) of respective source/destination. Misprediction info is reported in the upper bits of

'FROM' registers in the LBR stack. Elapsed cycles since the last LBR update are reported in the upper 16 bits of the 'TO' registers in the LBR stack (see Section 20.6).

- **000111B (64-bit LIP record format), Flags, and LBR_INFO** — Stores 64-bit linear address (CS.Base + effective address) of respective source/destination. Misprediction, and elapsed cycles since the last LBR update are reported in the LBR_INFO MSR stack.

Processor's support for the architectural MSR IA32_PERF_CAPABILITIES is provided by CPUID.01H:ECX.PERF_CAPABILITIES (bit 15).

20.4.8.2 LBR Stack and IA-32 Processors

The LBR MSRs in IA-32 processors introduced prior to Intel 64 architecture store the 32-bit "To Linear Address" and "From Linear Address" using the high and low half of each 64-bit MSR.

20.4.8.3 Last Exception Records and Intel 64 Architecture

Intel 64 and IA-32 processors also provide MSRs that store the branch record for the last branch taken prior to an exception or an interrupt. The location of the last exception record (LER) MSRs are model specific. The MSRs that store last exception records are 64-bits. If IA-32e mode is disabled, only the lower 32-bits of the address is recorded. If IA-32e mode is enabled, the processor writes 64-bit values into the MSR. In 64-bit mode, last exception records store 64-bit addresses; in compatibility mode, the upper 32-bits of last exception records are cleared.

20.4.9 BTS and DS Save Area

The **Debug store (DS)** feature flag (bit 21), returned by CPUID.01H:EDX[21] indicates that the processor provides the debug store (DS) mechanism. The DS mechanism allows:

- BTMs to be stored in a memory-resident BTS buffer. See Section 20.4.5, "Branch Trace Store (BTS)."
- Processor event-based sampling (PEBS) also uses the DS save area provided by debug store mechanism. The capability of PEBS varies across different microarchitectures. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)," and the relevant PEBS sub-sections across the core PMU sections in Chapter 22, "Performance Monitoring."

When CPUID.01H:EDX[21] is set:

- The BTS_UNAVAILABLE and PEBS_UNAVAILABLE flags in the IA32_MISC_ENABLE MSR indicate (when clear) the availability of the BTS and PEBS facilities, including the ability to set the BTS and BTINT bits in the appropriate DEBUGCTL MSR.
- The IA32_DS_AREA MSR exists and points to the DS save area.

The debug store (DS) save area is a software-designated area of memory that is used to collect the following two types of information:

- **Branch records** — When the BTS flag in the IA32_DEBUGCTL MSR is set, a branch record is stored in the BTS buffer in the DS save area whenever a taken branch, interrupt, or exception is detected.
- **PEBS records** — When a performance counter is configured for PEBS, a PEBS record is stored in the PEBS buffer in the DS save area after the counter overflow occurs. This record contains the architectural state of the processor (state of the 8 general purpose registers, EIP register, and EFLAGS register) at the next occurrence of the PEBS event that caused the counter to overflow. When the state information has been logged, the counter is automatically reset to a specified value, and event counting begins again. The content layout of a PEBS record varies across different implementations that support PEBS. See Section 22.6.2.4.2 for details of enumerating PEBS record format.

NOTES

Prior to processors based on the Goldmont microarchitecture, PEBS facility only supports a subset of implementation-specific precise events. See Section 22.5.3.1 for a PEBS enhancement that can generate records for both precise and non-precise events.

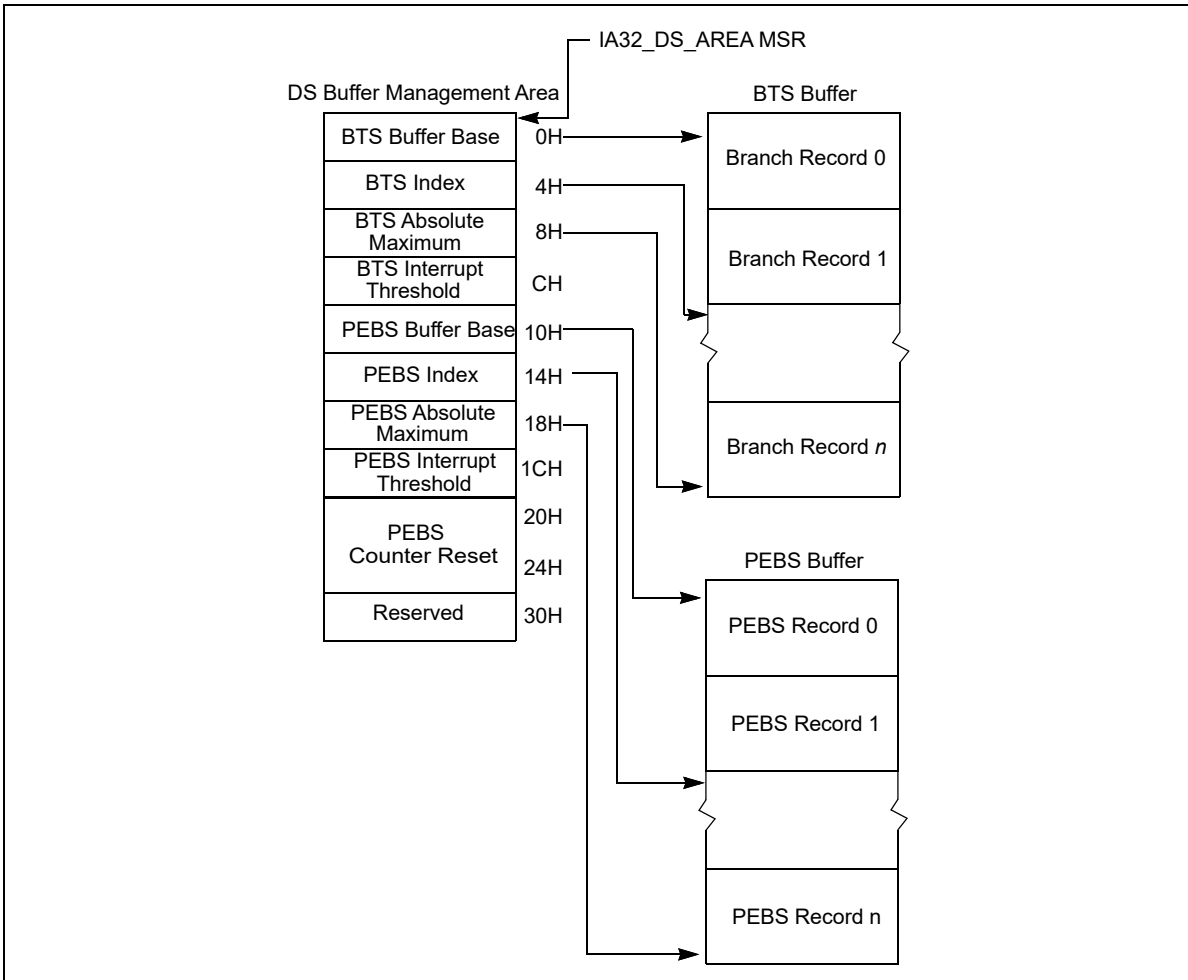
The DS save area and recording mechanism are disabled on INIT, processor Reset or transition to system-management mode (SMM) or IA-32e mode. It is similarly disabled on the generation of a machine-check exception on 45nm and 32nm Intel Atom processors and on processors with Netburst or Intel Core microarchitecture.

The BTS and PEBS facilities may not be available on all processors. The availability of these facilities is indicated by the `BTS_UNAVAILABLE` and `PEBS_UNAVAILABLE` flags, respectively, in the `IA32_MISC_ENABLE` MSR (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4).

The DS save area is divided into three parts: buffer management area, branch trace store (BTS) buffer, and PEBS buffer (see Figure 20-5). The buffer management area is used to define the location and size of the BTS and PEBS buffers. The processor then uses the buffer management area to keep track of the branch and/or PEBS records in their respective buffers and to record the performance counter reset value. The linear address of the first byte of the DS buffer management area is specified with the `IA32_DS_AREA` MSR.

The fields in the buffer management area are as follows:

- **BTS buffer base** — Linear address of the first byte of the BTS buffer. This address should point to a natural doubleword boundary.
- **BTS index** — Linear address of the first byte of the next BTS record to be written to. Initially, this address should be the same as the address in the BTS buffer base field.
- **BTS absolute maximum** — Linear address of the next byte past the end of the BTS buffer. This address should be a multiple of the BTS record size (12 bytes) plus 1.
- **BTS interrupt threshold** — Linear address of the BTS record on which an interrupt is to be generated. This address must point to an offset from the BTS buffer base that is a multiple of the BTS record size. Also, it must be several records short of the BTS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the BTS absolute maximum record.
- **PEBS buffer base** — Linear address of the first byte of the PEBS buffer. This address should point to a natural doubleword boundary.
- **PEBS index** — Linear address of the first byte of the next PEBS record to be written to. Initially, this address should be the same as the address in the PEBS buffer base field.
- **PEBS absolute maximum** — Linear address of the next byte past the end of the PEBS buffer. This address should be a multiple of the PEBS record size (40 bytes) plus 1.
- **PEBS interrupt threshold** — Linear address of the PEBS record on which an interrupt is to be generated. This address must point to an offset from the PEBS buffer base that is a multiple of the PEBS record size. Also, it must be several records short of the PEBS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the PEBS absolute maximum record.
- **PEBS counter reset value** — A 64-bit value that the counter is to be set to when a PEBS record is written. Bits beyond the size of the counter are ignored. This value allows state information to be collected regularly every time the specified number of events occur.

Figure 20-5. DS Save Area Example¹**NOTES:**

1. This example represents the format for a system that supports PEBS on only one counter.

Figure 20-6 shows the structure of a 12-byte branch record in the BTS buffer. The fields in each record are as follows:

- **Last branch from** — Linear address of the instruction from which the branch, interrupt, or exception was taken.
- **Last branch to** — Linear address of the branch target or the first instruction in the interrupt or exception service routine.
- **Branch predicted** — Bit 4 of field indicates whether the branch that was taken was predicted (set) or not predicted (clear).

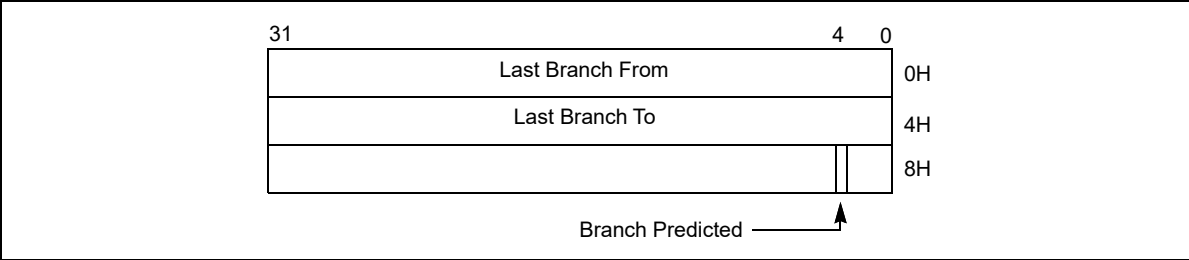


Figure 20-6. 32-bit Branch Trace Record Format

Figure 20-7 shows the structure of the 40-byte PEBS records. Nominally the register values are those at the beginning of the instruction that caused the event. However, there are cases where the registers may be logged in a partially modified state. The linear IP field shows the value in the EIP register translated from an offset into the current code segment to a linear address.

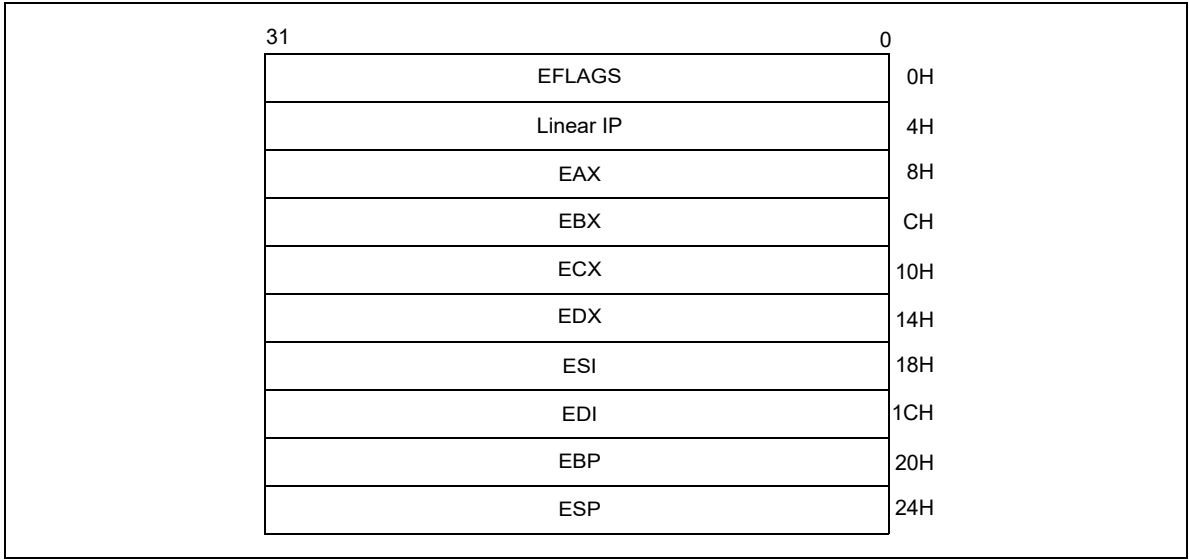


Figure 20-7. PEBS Record Format

20.4.9.1 64 Bit Format of the DS Save Area

When DTES64 = 1 (CPUID.01H:ECX[2] = 1), the structure of the DS save area is shown in Figure 20-8. When DTES64 = 0 (CPUID.01H:ECX[2] = 0) and IA-32e mode is active, the structure of the DS save area is shown in Figure 20-8. If IA-32e mode is not active the structure of the DS save area is as shown in Figure 20-5.

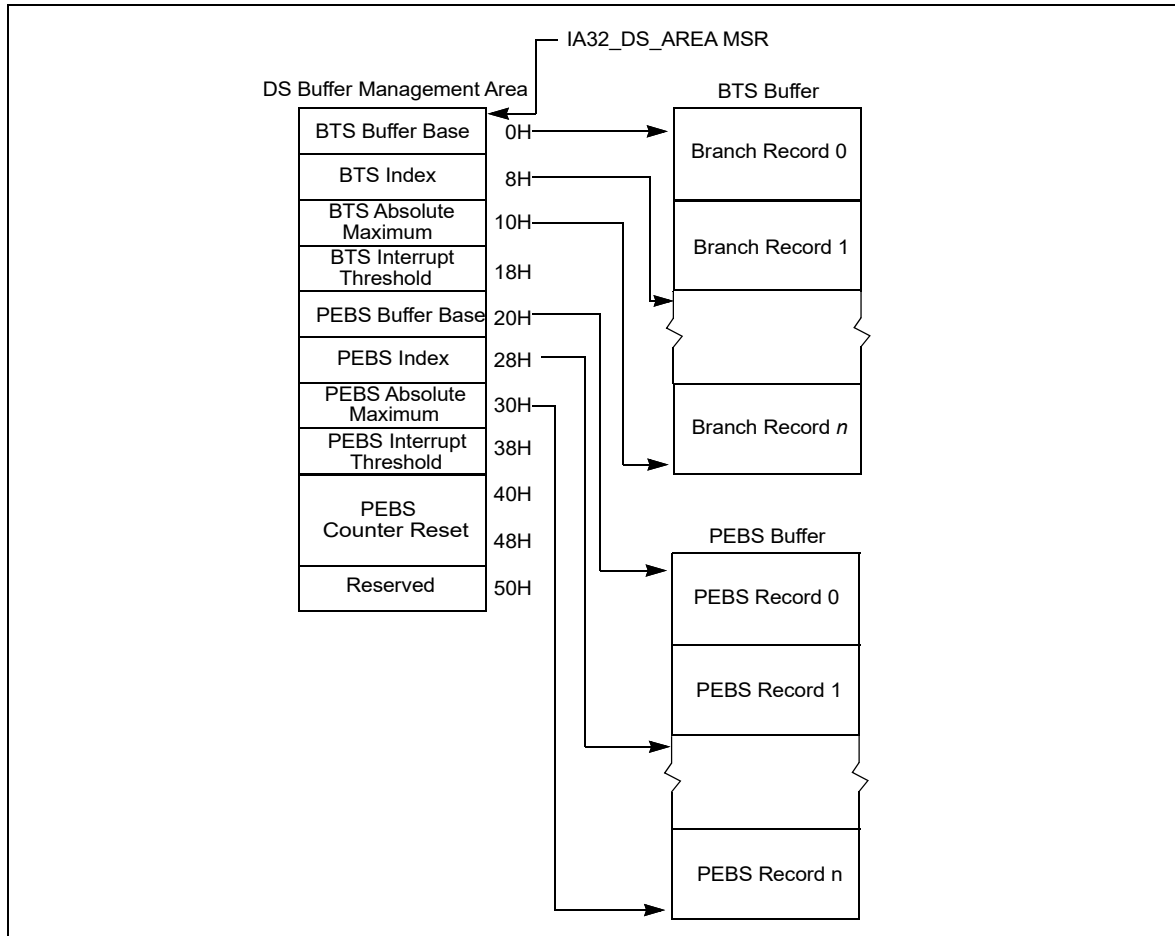


Figure 20-8. IA-32e Mode DS Save Area Example¹

NOTES:

1. This example represents the format for a system that supports PEBS on only one counter.

The IA32_DS_AREA MSR holds the 64-bit linear address of the first byte of the DS buffer management area. The structure of a branch trace record is similar to that shown in Figure 20-6, but each field is 8 bytes in length. This makes each BTS record 24 bytes (see Figure 20-9). The structure of a PEBS record is similar to that shown in Figure 20-7, but each field is 8 bytes in length and architectural states include register R8 through R15. This makes the size of a PEBS record in 64-bit mode 144 bytes (see Figure 20-10).

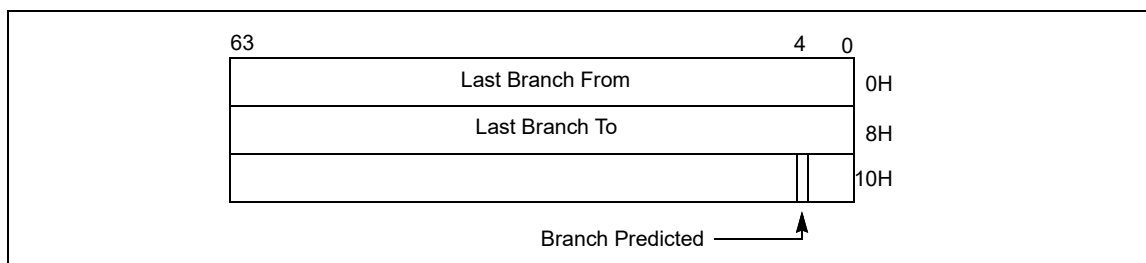


Figure 20-9. 64-bit Branch Trace Record Format

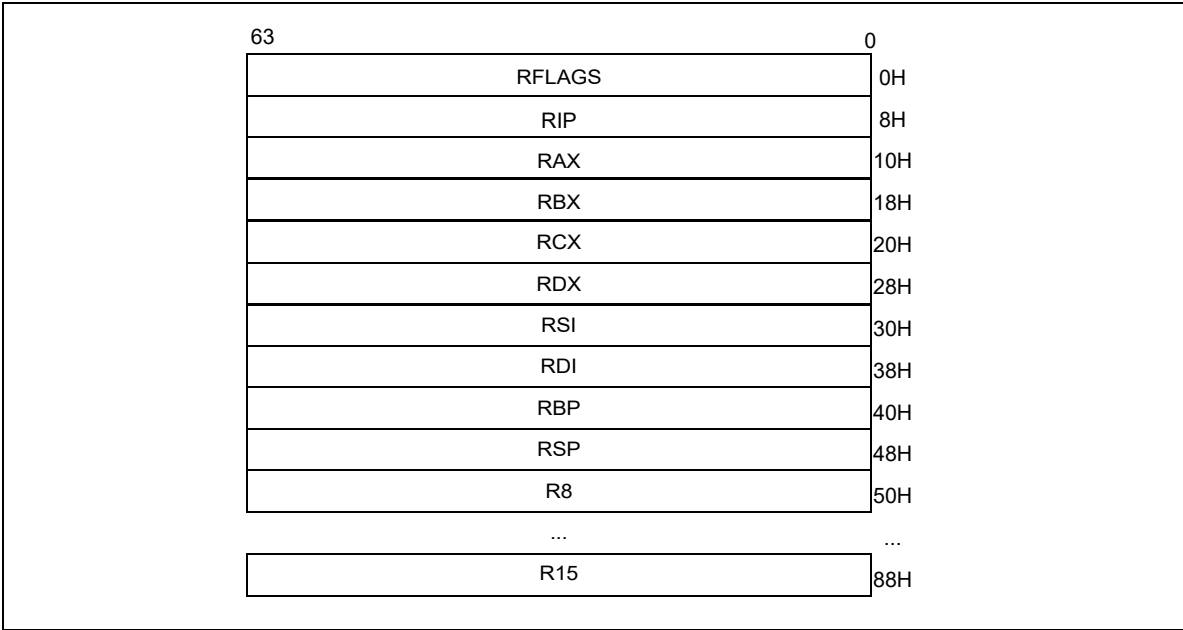


Figure 20-10. 64-bit PEBS Record Format

Fields in the buffer management area of a DS save area are described in Section 20.4.9.

The format of a branch trace record and a PEBS record are the same as the 64-bit record formats shown in Figure 20-9. "64-bit Branch Trace Record Format," and Figure 20-10. "64-bit PEBS Record Format," with the exception that the branch predicted bit is not supported by Intel Core microarchitecture or Intel Atom microarchitecture. The 64-bit record formats for BTS and PEBS apply to DS save area for all operating modes.

The procedures used to program IA32_DEBUGCTL MSR to set up a BTS buffer or a CPL-qualified BTS are described in Section 20.4.9.3 and Section 20.4.9.4.

Required elements for writing a DS interrupt service routine are largely the same on processors that support using DS Save area for BTS or PEBS records. However, on processors based on Intel NetBurst® microarchitecture, re-enabling counting requires writing to CCCRs. But a DS interrupt service routine on processors supporting architectural performance monitoring should:

- Re-enable the enable bits in IA32_PERF_GLOBAL_CTRL MSR if it is servicing an overflow PMI due to PEBS.
- Clear overflow indications by writing to IA32_PERF_GLOBAL_OVF_CTRL when a counting configuration is changed. This includes bit 62 (ClrOvfBuffer) and the overflow indication of counters used in either PEBS or general-purpose counting (specifically: bits 0 or 1; see Figure 22-3. "Layout of IA32_PERF_GLOBAL_CTRL MSR,").

20.4.9.2 Setting Up the DS Save Area

To save branch records with the BTS buffer, the DS save area must first be set up in memory as described in the following procedure (See Section 22.6.2.4.1, "Setting up the PEBS Buffer," for instructions for setting up a PEBS buffer, respectively, in the DS save area):

1. Create the DS buffer management information area in memory (see Section 20.4.9, "BTS and DS Save Area," and Section 20.4.9.1, "64 Bit Format of the DS Save Area"). Also see the additional notes in this section.
2. Write the base linear address of the DS buffer management area into the IA32_DS_AREA MSR.
3. Set up the performance counter entry in the xAPIC LVT for fixed delivery and edge sensitive. See Section 13.5.1, "Local Vector Table."

4. Establish an interrupt handler in the IDT for the vector associated with the performance counter entry in the xAPIC LVT.
5. Write an interrupt service routine to handle the interrupt. See Section 20.4.9.5, “Writing the DS Interrupt Service Routine.”

The following restrictions should be applied to the DS save area.

- The recording of branch records in the BTS buffer (or PEBS records in the PEBS buffer) may not operate properly if accesses to the linear addresses in any of the three DS save area sections cause page faults, VM exits, or the setting of accessed or dirty flags in the paging structures (ordinary or EPT). For that reason, system software should establish paging structures (both ordinary and EPT) to prevent such occurrences. Implications of this may be that an operating system should allocate this memory from a non-paged pool and that system software cannot do “lazy” page-table entry propagation for these pages. Some newer processor generations support “lazy” EPT page-table entry propagation for PEBS; see Section 22.3.9.1 and Section 22.9.5 for more information. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.
- The DS save area can be larger than a page, but the pages must be mapped to contiguous linear addresses. The buffer may share a page, so it need not be aligned on a 4-KByte boundary. For performance reasons, the base of the buffer must be aligned on a doubleword boundary and should be aligned on a cache line boundary.
- It is recommended that the buffer size for the BTS buffer and the PEBS buffer be an integer multiple of the corresponding record sizes.
- The precise event records buffer should be large enough to hold the number of precise event records that can occur while waiting for the interrupt to be serviced.
- The DS save area should be in kernel space. It must not be on the same page as code, to avoid triggering self-modifying code actions.
- There are no memory type restrictions on the buffers, although it is recommended that the buffers be designated as WB memory type for performance considerations.
- Either the system must be prevented from entering A20M mode while DS save area is active, or bit 20 of all addresses within buffer bounds must be 0.
- Pages that contain buffers must be mapped to the same physical addresses for all processes, such that any change to control register CR3 will not change the DS addresses.
- The DS save area is expected to be used only on systems with an enabled APIC. The LVT Performance Counter entry in the APCI must be initialized to use an interrupt gate instead of the trap gate.

20.4.9.3 Setting Up the BTS Buffer

Three flags in the MSR_DEBUGCTLA MSR (see Table 20-5), IA32_DEBUGCTL (see Figure 20-3), or MSR_DEBUGCTLB (see Figure 20-16) control the generation of branch records and storing of them in the BTS buffer; these are TR, BTS, and BTINT. The TR flag enables the generation of BTMs. The BTS flag determines whether the BTMs are sent out on the system bus (clear) or stored in the BTS buffer (set). BTMs cannot be simultaneously sent to the system bus and logged in the BTS buffer. The BTINT flag enables the generation of an interrupt when the BTS buffer is full. When this flag is clear, the BTS buffer is a circular buffer.

Table 20-5. IA32_DEBUGCTL Flag Encodings

TR	BTS	BTINT	Description
0	X	X	Branch trace messages (BTMs) off
1	0	X	Generate BTMs
1	1	0	Store BTMs in the BTS buffer, used here as a circular buffer
1	1	1	Store BTMs in the BTS buffer, and generate an interrupt when the buffer is nearly full

The following procedure describes how to set up a DS Save area to collect branch records in the BTS buffer:

1. Place values in the BTS buffer base, BTS index, BTS absolute maximum, and BTS interrupt threshold fields of the DS buffer management area to set up the BTS buffer in memory.

- 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).
- Clear the BTINT flag in the corresponding IA32_DEBUGCTL (or MSR_DEBUGCTLA MSR; or MSR_DEBUGCTLB) if a circular BTS buffer is desired.

NOTES

If the buffer size is set to less than the minimum allowable value (i.e., BTS absolute maximum < 1 + size of BTS record), the results of BTS is undefined.

In order to prevent generating an interrupt, when working with circular BTS buffer, SW need to set BTS interrupt threshold to a value greater than BTS absolute maximum (fields of the DS buffer management area). It's not enough to clear the BTINT flag itself only.

20.4.9.4 Setting Up CPL-Qualified BTS

If the processor supports CPL-qualified last branch recording mechanism, the generation of branch records and storing of them in the BTS buffer are determined by: TR, BTS, BTS_OFF_OS, BTS_OFF_USR, and BTINT. The encoding of these five bits are shown in Table 20-6.

Table 20-6. CPL-Qualified Branch Trace Store Encodings

TR	BTS	BTS_OFF_OS	BTS_OFF_USR	BTINT	Description
0	X	X	X	X	Branch trace messages (BTMs) off
1	0	X	X	X	Generates BTMs but do not store BTMs
1	1	0	0	0	Store all BTMs in the BTS buffer, used here as a circular buffer
1	1	1	0	0	Store BTMs with CPL > 0 in the BTS buffer
1	1	0	1	0	Store BTMs with CPL = 0 in the BTS buffer
1	1	1	1	X	Generate BTMs but do not store BTMs
1	1	0	0	1	Store all BTMs in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	1	0	1	Store BTMs with CPL > 0 in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	0	1	1	Store BTMs with CPL = 0 in the BTS buffer; generate an interrupt when the buffer is nearly full

20.4.9.5 Writing the DS Interrupt Service Routine

The BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector and interrupt service routine (called the debug store interrupt service routine or DS ISR). To handle BTS, non-precise event-based sampling, and PEBS interrupts: separate handler routines must be included in the DS ISR. Use the following guidelines when writing a DS ISR to handle BTS, non-precise event-based sampling, and/or PEBS interrupts.

- The DS interrupt service routine (ISR) must be part of a kernel driver and operate at a current privilege level of 0 to secure the buffer storage area.
- Because the BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector, the DS ISR must check for all the possible causes of interrupts from these facilities and pass control on to the appropriate handler.

BTS and PEBS buffer overflow would be the sources of the interrupt if the buffer index matches/exceeds the interrupt threshold specified. Detection of non-precise event-based sampling as the source of the interrupt is accomplished by checking for counter overflow.

- There must be separate save areas, buffers, and state for each processor in an MP system.
- Upon entering the ISR, branch trace messages and PEBS should be disabled to prevent race conditions during access to the DS save area. This is done by clearing TR flag in the IA32_DEBUGCTL (or MSR_DEBUGCTLA MSR)

and by clearing the precise event enable flag in the MSR_PEBS_ENABLE MSR. These settings should be restored to their original values when exiting the ISR.

- The processor will not disable the DS save area when the buffer is full and the circular mode has not been selected. The current DS setting must be retained and restored by the ISR on exit.
- After reading the data in the appropriate buffer, up to but not including the current index into the buffer, the ISR must reset the buffer index to the beginning of the buffer. Otherwise, everything up to the index will look like new entries upon the next invocation of the ISR.
- The ISR must clear the mask bit in the performance counter LVT entry.
- The ISR must re-enable the counters to count via IA32_PERF_GLOBAL_CTRL/IA32_PERF_GLOBAL_OVF_CTRL if it is servicing an overflow PMI due to PEBS (or via CCCR's ENABLE bit on processor based on Intel NetBurst microarchitecture).
- The Pentium 4 Processor and Intel Xeon Processor mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

20.5 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ 2 DUO AND INTEL ATOM® PROCESSORS)

The Intel Core 2 Duo processor family and Intel Xeon processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture provide last branch interrupt and exception recording. The facilities described in this section also apply to 45 nm and 32 nm Intel Atom processors. These capabilities are similar to those found in Pentium 4 processors, including support for the following facilities:

- **Debug Trace and Branch Recording Control** — The IA32_DEBUGCTL MSR provide bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 20.4.1 for a description of the flags. See Figure 20-3 for the MSR layout.
- **Last branch record (LBR) stack** — There are a collection of MSR pairs that store the source and destination addresses related to recently executed branches. See Section 20.5.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts**
 - See Section 20.4.2 and Section 20.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.
 - 45 nm and 32 nm Intel Atom processors clear the TR flag when the FREEZE_LBRS_ON_PMI flag is set.
- **Branch trace messages** — See Section 20.4.4.
- **Last exception records** — See Section 20.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 20.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 20.4.7 for legacy Freeze_LBRS_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 20.4.7 for legacy Freeze_PerfMon_On_PMI operation.
- **FREEZE_WHILE_SMM (bit 14)** — FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 20.4.1.

20.5.1 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel Core 2, Intel Atom processor families, and Intel processors based on Intel NetBurst microarchitecture.

Four pairs of MSRs are supported in the LBR stack for Intel Core 2 processors families and Intel processors based on Intel NetBurst microarchitecture:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_3_FROM_IP (address 43H) store source addresses

- MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_3_TO_IP (address 63H) store destination addresses
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 2 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

Eight pairs of MSRs are supported in the LBR stack for 45 nm and 32 nm Intel Atom processors:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_7_FROM_IP (address 47H) store source addresses
 - MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_7_TO_IP (address 67H) store destination addresses
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 3 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

The address format written in the FROM_IP/TO_IP MSRs may differ between processors. Software should query IA32_PERF_CAPABILITIES[5:0] and consult Section 20.4.8.1. The behavior of the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs corresponds to that of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

20.5.2 LBR Stack in Intel Atom® Processors based on the Silvermont Microarchitecture

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in Intel Atom processors based on the Silvermont and Airmont microarchitectures. Eight pairs of MSRs are supported in the LBR stack.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 20-11.

20.6 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Processors based on the Goldmont microarchitecture extend the capabilities described in Section 20.5.2 with the following enhancements:

- Supports new LBR format encoding 00110b in IA32_PERF_CAPABILITIES[5:0].
- Size of LBR stack increased to 32. Each entry includes MSR_LASTBRANCH_x_FROM_IP (address 0x680..0x69f) and MSR_LASTBRANCH_x_TO_IP (address 0x6c0..0x6df).
- LBR call stack filtering supported. The layout of MSR_LBR_SELECT is described in Table 20-13.
- Elapsed cycle information is added to MSR_LASTBRANCH_x_TO_IP. Format is shown in Table 20-7.
- Misprediction info is reported in the upper bits of MSR_LASTBRANCH_x_FROM_IP. MISRPRED bit format is shown in Table 20-8.
- Streamlined Freeze_LBRs_On_PMI operation; see Section 20.12.2.
- LBR MSRs may be cleared when MWAIT is used to request a C-state that is numerically higher than C1; see Section 20.12.3.

Table 20-7. MSR_LASTBRANCH_x_TO_IP for the Goldmont Microarchitecture

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 20.4.8.1 for address format.
Cycle Count (Saturating)	63:48	R/W	Elapsed core clocks since last update to the LBR stack.

20.7 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Next generation Intel Atom processors are based on the Goldmont Plus microarchitecture. Processors based on the Goldmont Plus microarchitecture extend the capabilities described in Section 20.6 with the following changes:

- Enumeration of new LBR format: encoding 00111b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 20.4.8.1.
- Each LBR stack entry consists of three MSRs:
 - MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 20-9.
 - MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 20-9.
 - MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data. Layout is the same as Table 20-16.

20.8 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR INTEL® XEON PHI™ PROCESSOR 7200/5200/3200

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in the Intel® Xeon Phi™ processor 7200/5200/3200 series based on the Knights Landing microarchitecture. Eight pairs of MSRs are supported in the LBR stack, per thread:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 680H) through MSR_LASTBRANCH_7_FROM_IP (address 687H) store source addresses.
 - MSR_LASTBRANCH_0_TO_IP (address 6C0H) through MSR_LASTBRANCH_7_TO_IP (address 6C7H) store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 3 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 20-11.

The address format written in the FROM_IP/TO_IP MSRs may differ between processors. Software should query IA32_PERF_CAPABILITIES[5:0] and consult Section 20.4.8.1. The behavior of the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs corresponds to that of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors.

20.9 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON NEHALEM MICROARCHITECTURE

The processors based on Nehalem microarchitecture and Westmere microarchitecture support last branch interrupt and exception recording. These capabilities are similar to those found in Intel Core 2 processors and add additional capabilities:

- **Debug Trace and Branch Recording Control** — The IA32_DEBUGCTL MSR provides bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 20.4.1 for a description of the flags. See Figure 20-11 for the MSR layout.
- **Last branch record (LBR) stack** — There are 16 MSR pairs that store the source and destination addresses related to recently executed branches. See Section 20.9.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts** — See Section 20.4.2 and Section 20.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.

- **Branch trace messages** — The IA32_DEBUGCTL MSR provides bit fields for software to enable each logical processor to generate branch trace messages. See Section 20.4.4. However, not all BTM messages are observable using the Intel® QPI link.
- **Last exception records** — See Section 20.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 20.4.6 and Section 20.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 20.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 20.4.7 for legacy Freeze_PerfMon_On_PMI operation.
- **UNCORE_PMI_EN (bit 13)** — When set, this logical processor is enabled to receive an counter overflow interrupt form the uncore.
- **FREEZE_WHILE_SMM (bit 14)** — FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 20.4.1.

Processors based on Nehalem microarchitecture provide additional capabilities:

- **Independent control of uncore PMI** — The IA32_DEBUGCTL MSR provides a bit field (see Figure 20-11) for software to enable each logical processor to receive an uncore counter overflow interrupt.
- **LBR filtering** — Processors based on Nehalem microarchitecture support filtering of LBR based on combination of CPL and branch type conditions. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT.

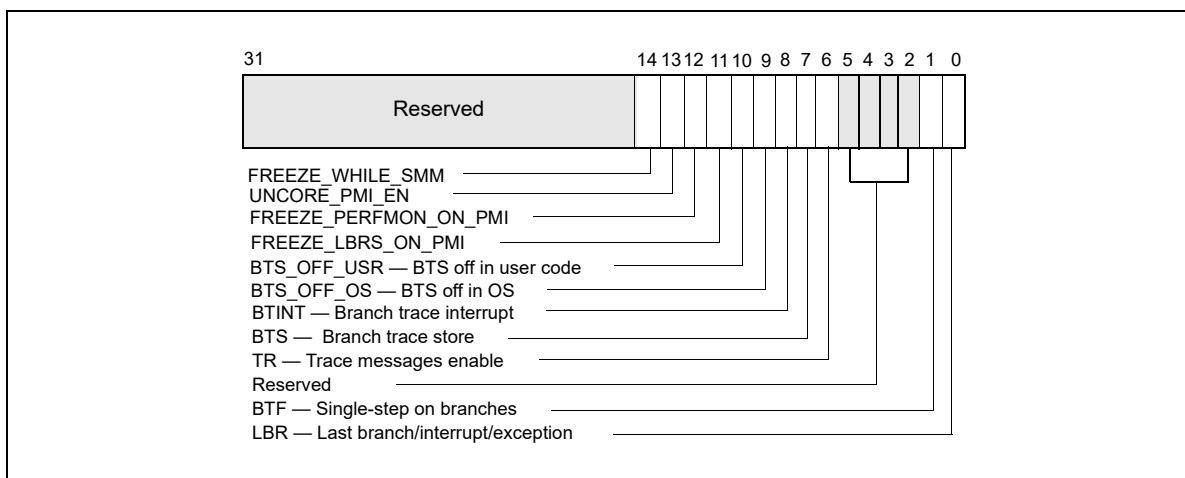


Figure 20-11. IA32_DEBUGCTL MSR for Processors Based on Nehalem Microarchitecture

20.9.1 LBR Stack

Processors based on Nehalem microarchitecture provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is shown in Table 20-8 and Table 20-9.

Table 20-8. MSR_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the "branch from" address. See Section 20.4.8.1 for address format.
SIGN_EXT	62:48	R/W	Signed extension of bit 47 of this register.
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

Table 20-9. MSR_LASTBRANCH_x_TO_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 20.4.8.1 for address format
SIGN_EXT	63:48	R/W	Signed extension of bit 47 of this register.

Processors based on Nehalem microarchitecture have an LBR MSR Stack as shown in Table 20-10.

Table 20-10. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_1AH	16	0 to 15

20.9.2 Filtering of Last Branch Records

MSR_LBR_SELECT is cleared to zero at RESET, and LBR filtering is disabled, i.e., all branches will be captured. MSR_LBR_SELECT provides bit fields to specify the conditions of subsets of branches that will not be captured in the LBR. The layout of MSR_LBR_SELECT is shown in Table 20-11.

Table 20-11. MSR_LBR_SELECT for Nehalem Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

20.10 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SANDY BRIDGE MICROARCHITECTURE

Generally, all of the last branch record, interrupt, and exception recording facility described in Section 20.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture,” apply to processors based on Sandy Bridge microarchitecture. For processors based on Ivy Bridge microarchitecture, the same holds true.

One difference of note is that MSR_LBR_SELECT is shared between two logical processors in the same core. In Sandy Bridge microarchitecture, each logical processor has its own MSR_LBR_SELECT. The filtering semantics for “Near_ind_jmp” and “Near_rel_jmp” has been enhanced, see Table 20-12.

Table 20-12. MSR_LBR_SELECT for Sandy Bridge Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

20.11 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON HASWELL MICROARCHITECTURE

Generally, all of the last branch record, interrupt, and exception recording facility described in Section 20.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture,” apply to next generation processors based on Haswell microarchitecture.

The LBR facility also supports an alternate capability to profile call stack profiles. Configuring the LBR facility to conduct call stack profiling is by writing 1 to the MSR_LBR_SELECT.EN_CALLSTACK[bit 9]; see Table 20-13. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 20.10.

Table 20-13. MSR_LBR_SELECT for Haswell Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
EN_CALLSTACK ¹	9		Enable LBR stack to use LIFO filtering to capture Call stack profile
Reserved	63:10		Must be zero

NOTES:

1. Must set valid combination of bits 0-8 in conjunction with bit 9 (as described below), otherwise the contents of the LBR MSRs are undefined.

The call stack profiling capability is an enhancement of the LBR facility. The LBR stack is a ring buffer typically used to profile control flow transitions resulting from branches. However, the finite depth of the LBR stack often become less effective when profiling certain high-level languages (e.g., C++), where a transition of the execution flow is accompanied by a large number of leaf function calls, each of which returns an individual parameter to form the list

of parameters for the main execution function call. A long list of such parameters returned by the leaf functions would serve to flush the data captured in the LBR stack, often losing the main execution context.

When the call stack feature is enabled, the LBR stack will capture unfiltered call data normally, but as return instructions are executed the last captured branch record is flushed from the on-chip registers in a last-in first-out (LIFO) manner. Thus, branch information relative to leaf functions will not be captured, while preserving the call stack information of the main line execution path.

The configuration of the call stack facility is summarized below:

- Set IA32_DEBUGCTL.LBR (bit 0) to enable the LBR stack to capture branch records. The source and target addresses of the call branches will be captured in the 16 pairs of From/To LBR MSRs that form the LBR stack.
- Program the Top of Stack (TOS) MSR that points to the last valid from/to pair. This register is incremented by 1, modulo 16, before recording the next pair of addresses.
- Program the branch filtering bits of MSR_LBR_SELECT (bits 0:8) as desired.
- Program the MSR_LBR_SELECT to enable LIFO filtering of return instructions with:
 - The following bits in MSR_LBR_SELECT must be set to '1': JCC, NEAR_IND_JMP, NEAR_REL_JMP, FAR_BRANCH, EN_CALLSTACK;
 - The following bits in MSR_LBR_SELECT must be cleared: NEAR_REL_CALL, NEAR-IND_CALL, NEAR_RET;
 - At most one of CPL_EQ_0, CPL_NEQ_0 is set.

Note that when call stack profiling is enabled, “zero length calls” are excluded from writing into the LBRs. (A “zero length call” uses the attribute of the call instruction to push the immediate instruction pointer on to the stack and then pops off that address into a register. This is accomplished without any matching return on the call.)

20.11.1 LBR Stack Enhancement

Processors based on Haswell microarchitecture provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is enumerated by IA32_PERF_CAPABILITIES[5:0] = 04H, and is shown in Table 20-14 and Table 20-9.

Table 20-14. MSR_LASTBRANCH_x_FROM_IP with TSX Information

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch from” address. See Section 20.4.8.1 for address format.
SIGN_EXT	60:48	R/W	Signed extension of bit 47 of this register.
TSX_ABORT	61	R/W	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region, or EIP of the RTM Abort Handler
IN_TSX	62	R/W	When set, indicates the entry occurred in a TSX region
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

20.12 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SKYLAKE MICROARCHITECTURE

Processors based on the Skylake microarchitecture provide a number of enhancement with storing last branch records:

- enumeration of new LBR format: encoding 00101b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 20.4.8.1.
- Each LBR stack entry consists of a triplets of MSRs:

- MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 20-9.
- MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 20-9.
- MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data.
- Size of LBR stack increased to 32.

Processors based on the Skylake microarchitecture support the same LBR filtering capabilities as described in Table 20-13.

Table 20-15. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_4EH, 06_5EH	32	0 to 31

20.12.1 MSR_LBR_INFO_x MSR

The layout of each MSR_LBR_INFO_x MSR is shown in Table 20-16.

Table 20-16. MSR_LBR_INFO_x

Bit Field	Bit Offset	Access	Description
Cycle Count (saturating)	15:0	R/W	Elapsed core clocks since last update to the LBR stack.
Reserved	60:16	R/W	Reserved
TSX_ABORT	61	R/W	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region OR EIP of the RTM Abort Handler
IN_TSX	62	R/W	When set, indicates the entry occurred in a TSX region.
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

20.12.2 Streamlined Freeze_LBRs_On_PMI Operation

The FREEZE_LBRs_ON_PMI feature causes the LBRs to be frozen on a hardware request for a PMI. This prevents the LBRs from being overwritten by new branches, allowing the PMI handler to examine the control flow that preceded the PMI generation. Architectural performance monitoring version 4 and above supports a streamlined FREEZE_LBRs_ON_PMI operation for PMI service routine that replaces the legacy FREEZE_LBRs_ON_PMI operation (see Section 20.4.7).

While the legacy FREEZE_LBRs_ON_PMI clear the LBR bit in the IA32_DEBUGCTL MSR on a PMI request, the streamlined FREEZE_LBRs_ON_PMI will set the LBR_FRZ bit in IA32_PERF_GLOBAL_STATUS. Branches will not cause the LBRs to be updated when LBR_FRZ is set. Software can clear LBR_FRZ at the same time as it clears overflow bits by setting the LBR_FRZ bit as well as the needed overflow bit when writing to IA32_PERF_GLOBAL_STATUS_RESET MSR.

This streamlined behavior avoids race conditions between software and processor writes to IA32_DEBUGCTL that are possible with FREEZE_LBRs_ON_PMI clearing of the LBR enable.

20.12.3 LBR Behavior and Deep C-State

When MWAIT is used to request a C-state that is numerically higher than C1, then LBR state may be initialized to zero depending on optimized “waiting” state that is selected by the processor. The affected LBR states include the FROM, TO, INFO, LAST_BRANCH, LER, and LBR_TOS registers. The LBR enable bit and LBR_FROZEN bit are not affected. The LBR-time of the first LBR record inserted after an exit from such a C-state request will be zero.

20.13 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PROCESSORS BASED ON INTEL NETBURST® MICROARCHITECTURE)

Pentium 4 and Intel Xeon processors based on Intel NetBurst microarchitecture provide the following methods for recording taken branches, interrupts, and exceptions:

- Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consists of a branch-from and a branch-to instruction address.
- Send the branch records out on the system bus as branch trace messages (BTMs).
- Log BTMs in a memory-resident branch trace store (BTS) buffer.

To support these functions, the processor provides the following MSRs and related facilities:

- **MSR_DEBUGCTLA MSR** — Enables last branch, interrupt, and exception recording; single-stepping on taken branches; branch trace messages (BTMs); and branch trace store (BTS). This register is named DebugCtlMSR in the P6 family processors.
- **Debug store (DS) feature flag (CPUID.01H:EDX.DS[21])** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer.
- **CPL-qualified debug store (DS) feature flag (CPUID.01H:ECX.DS_CPL[4])** — Indicates that the processor provides a CPL-qualified debug store (DS) mechanism, which allows software to selectively skip sending and storing BTMs, according to specified current privilege level settings, into a memory-resident BTS buffer.
- **IA32_MISC_ENABLE MSR** — Indicates that the processor provides the BTS facilities.
- **Last branch record (LBR) stack** — The LBR stack is a circular stack that consists of four MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. The LBR stack consists of 16 MSR pairs (MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_15_FROM_IP and MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_15_TO_IP) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H].
- **Last branch record top-of-stack (TOS) pointer** — The TOS Pointer MSR contains a 2-bit pointer (0-3) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. This pointer becomes a 4-bit pointer (0-15) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H]. See also: Table 20-17, Figure 20-12, and Section 20.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **Last exception record** — See Section 20.13.3, “Last Exception Records.”

20.13.1 MSR_DEBUGCTLA MSR

The MSR_DEBUGCTLA MSR enables and disables the various last branch recording mechanisms described in the previous section. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode. A protected-mode operating system procedure is required to provide user access to this register. Figure 20-12 shows the flags in the MSR_DEBUGCTLA MSR. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. Each branch, interrupt, or exception is recorded as a 64-bit branch record. The processor clears this flag whenever a debug exception is generated (for example,

- when an instruction or data breakpoint or a single-step trap occurs). See Section 20.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 20.4.3, “Single-Stepping on Branches.”
 - **TR (trace message enable) flag (bit 2)** — When set, branch trace messages are enabled. Thereafter, when the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 20.4.4, “Branch Trace Messages.”

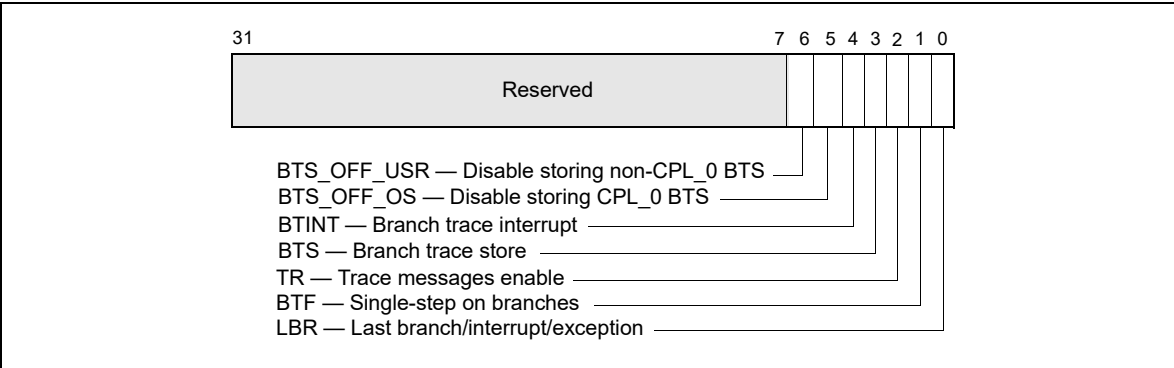


Figure 20-12. MSR_DEBUGCTLA MSR for Pentium 4 and Intel Xeon Processors

- **BTS (branch trace store) flag (bit 3)** — When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 20.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 4)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 20.4.5, “Branch Trace Store (BTS).”
- **BTS_OFF_OS (disable ring 0 branch trace store) flag (bit 5)** — When set, enables the BTS facilities to skip sending/logging CPL_0 BTMs to the memory-resident BTS buffer. See Section 20.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **BTS_OFF_USR (disable ring 0 branch trace store) flag (bit 6)** — When set, enables the BTS facilities to skip sending/logging non-CPL_0 BTMs to the memory-resident BTS buffer. See Section 20.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”

NOTE

The initial implementation of BTS_OFF_USR and BTS_OFF_OS in MSR_DEBUGCTLA is shown in Figure 20-12. The BTS_OFF_USR and BTS_OFF_OS fields may be implemented on other model-specific debug control register at different locations.

See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for a detailed description of each of the last branch recording MSRs.

20.13.2 LBR Stack for Processors Based on Intel NetBurst® Microarchitecture

The LBR stack is made up of LBR MSRs that are treated by the processor as a circular stack. The TOS pointer (MSR_LASTBRANCH_TOS MSR) points to the LBR MSR (or LBR MSR pair) that contains the most recent (last) branch record placed on the stack. Prior to placing a new branch record on the stack, the TOS is incremented by 1. When the TOS pointer reaches its maximum value, it wraps around to 0. See Table 20-17 and Figure 20-12.

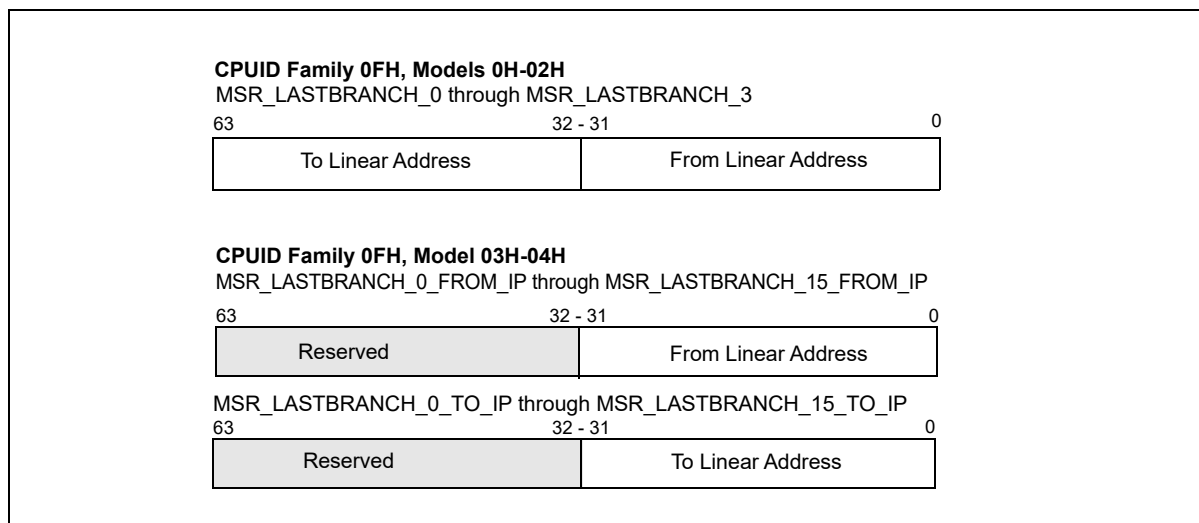
Table 20-17. LBR MSR Stack Size and TOS Pointer Range for the Pentium® 4 and the Intel® Xeon® Processor Family

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
Family 0FH, Models 0H-02H; MSRs at locations 1DBH-1DEH.	4	0 to 3
Family 0FH, Models; MSRs at locations 680H-68FH.	16	0 to 15
Family 0FH, Model 03H; MSRs at locations 6C0H-6CFH.	16	0 to 15

The registers in the LBR MSR stack and the MSR_LASTBRANCH_TOS MSR are read-only and can be read using the RDMSR instruction.

Figure 20-13 shows the layout of a branch record in an LBR MSR (or MSR pair). Each branch record consists of two linear addresses, which represent the “from” and “to” instruction pointers for a branch, interrupt, or exception. The contents of the from and to addresses differ, depending on the source of the branch:

- **Taken branch** — If the record is for a taken branch, the “from” address is the address of the branch instruction and the “to” address is the target instruction of the branch.
- **Interrupt** — If the record is for an interrupt, the “from” address the return instruction pointer (RIP) saved for the interrupt and the “to” address is the address of the first instruction in the interrupt handler routine. The RIP is the linear address of the next instruction to be executed upon returning from the interrupt handler.
- **Exception** — If the record is for an exception, the “from” address is the linear address of the instruction that caused the exception to be generated and the “to” address is the address of the first instruction in the exception handler routine.

**Figure 20-13. LBR MSR Branch Record Layout for the Pentium 4 and Intel® Xeon® Processor Family**

Additional information is saved if an exception or interrupt occurs in conjunction with a branch instruction. If a branch instruction generates a trap type exception, two branch records are stored in the LBR stack: a branch record for the branch instruction followed by a branch record for the exception.

If a branch instruction is immediately followed by an interrupt, a branch record is stored in the LBR stack for the branch instruction followed by a record for the interrupt.

20.13.3 Last Exception Records

The Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel Atom® processors provide two MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate the functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors.

The MSR_LER_TO_LIP and MSR_LER_FROM_LIP MSRs contain a branch record for the last branch that the processor took prior to an exception or interrupt being generated.

20.14 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS)

Intel Core Solo and Intel Core Duo processors provide last branch interrupt and exception recording. This capability is almost identical to that found in Pentium 4 and Intel Xeon processors. There are differences in the stack and in some MSR names and locations.

Note the following:

- **IA32_DEBUGCTL MSR** — Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 20-14 for the layout and the entries below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” below.
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 20.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 20.4.4, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 20.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 20.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

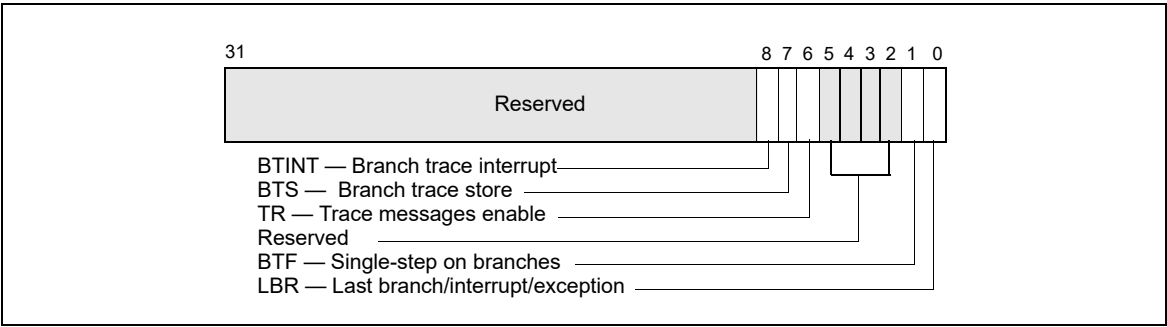


Figure 20-14. IA32_DEBUGCTL MSR for Intel® Core™ Solo and Intel® Core™ Duo Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 20.4.5, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’ address, bits 63-32 hold the ‘to’ address (MSR addresses start at 40H). See Figure 20-15.

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Intel Core Solo and Intel Core Duo processors, this MSR is located at register address 01C9H.

For compatibility, the Intel Core Solo and Intel Core Duo processors provide two 32-bit MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

For details, see Section 20.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture,” and Section 2.20, “MSRs In Intel® Core™ Solo and Intel® Core™ Duo Processors,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

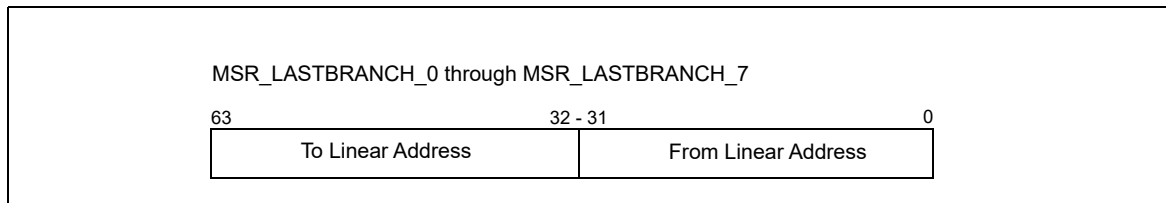


Figure 20-15. LBR Branch Record Layout for the Intel® Core™ Solo and Intel® Core™ Duo Processor

20.15 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PENTIUM M PROCESSORS)

Like the Pentium 4 and Intel Xeon processor family, Pentium M processors provide last branch interrupt and exception recording. The capability operates almost identically to that found in Pentium 4 and Intel Xeon processors. There are differences in the shape of the stack and in some MSR names and locations. Note the following:

- **MSR_DEBUGCTLB MSR** — Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. For Pentium M processors, this MSR is located at register address 01D9H. See Figure 20-16 and the entries below for a description of the flags.
 - **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” bullet below.
 - **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 20.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
 - **PBi (performance monitoring/breakpoint pins) flags (bits 5-2)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding BPi# pin when a breakpoint match occurs. When a PBi flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
 - **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 20.4.4, “Branch Trace Messages,” for more information about the TR flag.
 - **BTS (branch trace store) flag (bit 7)** — When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 20.4.9, “BTS and DS Save Area.”
 - **BTINT (branch trace interrupt) flag (bits 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 20.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

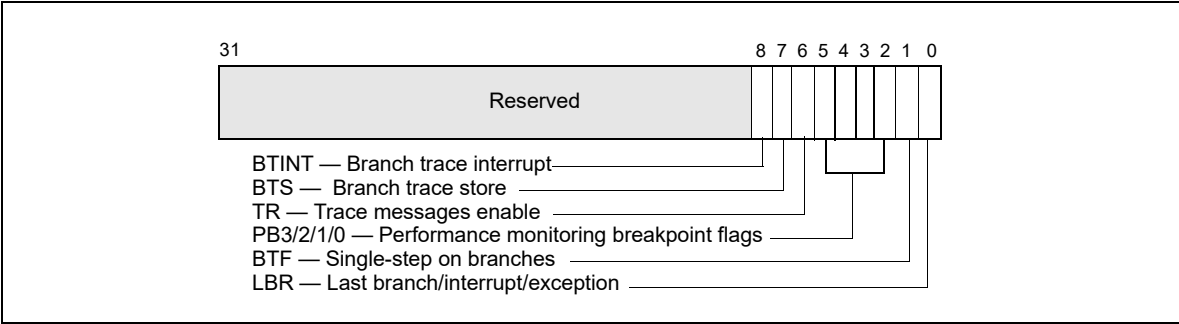


Figure 20-16. MSR_DEBUGCTLB MSR for Pentium M Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 20.4.5, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’ address, bits 63-32 hold the ‘to’ address. For Pentium M Processors, these pairs are located at register addresses 040H-047H. See Figure 20-17.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Pentium M Processors, this MSR is located at register address 01C9H.

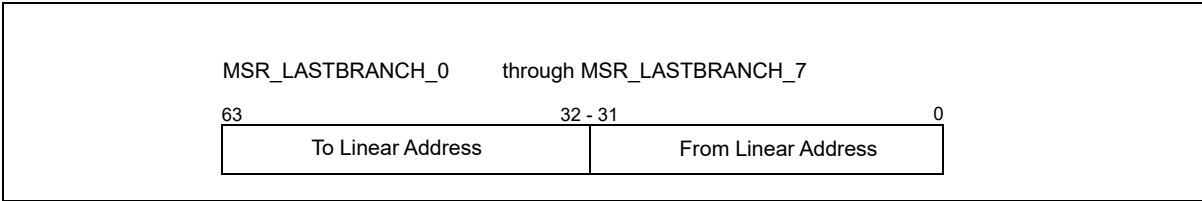


Figure 20-17. LBR Branch Record Layout for the Pentium M Processor

For more detail on these capabilities, see Section 20.13.3, “Last Exception Records,” and Section 2.21, “MSRs In the Pentium M Processor,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

20.16 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (P6 FAMILY PROCESSORS)

The P6 family processors provide five MSRs for recording the last branch, interrupt, or exception taken by the processor: DEBUGCTLMR, LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP. These registers can be used to collect last branch records, to set breakpoints on branches, interrupts, and exceptions, and to single-step from one branch to the next.

See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for a detailed description of each of the last branch recording MSRs.

20.16.1 DEBUGCTLMR Register

The version of the DEBUGCTLMR register found in the P6 family processors enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.

A protected-mode operating system procedure is required to provide user access to this register. Figure 20-18 shows the flags in the DEBUGCTLMSR register for the P6 family processors. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records the source and target addresses (in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs) for the last branch and the last exception or interrupt taken by the processor prior to a debug exception being generated. The processor clears this flag whenever a debug exception, such as an instruction or data breakpoint or single-step trap occurs.

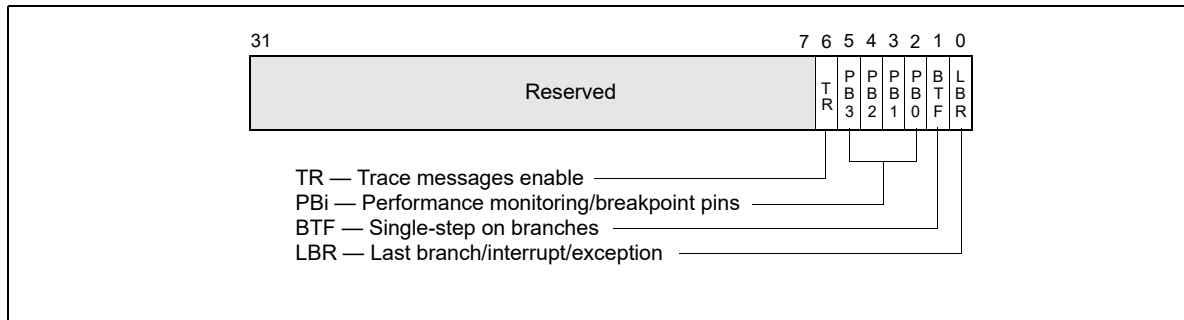


Figure 20-18. DEBUGCTLMSR Register (P6 Family Processors)

- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag. See Section 20.4.3, “Single-Stepping on Branches.”
- **PBi (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 BPi# pin when a breakpoint match occurs. When a PBi flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
- **TR (trace message enable) flag (bit 6)** — When set, trace messages are enabled as described in Section 20.4.4, “Branch Trace Messages.” Setting this flag greatly reduces the performance of the processor. When trace messages are enabled, the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are undefined.

20.16.2 Last Branch and Last Exception MSRs

The LastBranchToIP and LastBranchFromIP MSRs are 32-bit registers for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated. When a branch occurs, the processor loads the address of the branch instruction into the LastBranchFromIP MSR and loads the target address for the branch into the LastBranchToIP MSR.

When an interrupt or exception occurs (other than a debug exception), the address of the instruction that was interrupted by the exception or interrupt is loaded into the LastBranchFromIP MSR and the address of the exception or interrupt handler that is called is loaded into the LastBranchToIP MSR.

The LastExceptionToIP and LastExceptionFromIP MSRs (also 32-bit registers) record the instruction pointers for the last branch that the processor took prior to an exception or interrupt being generated. When an exception or interrupt occurs, the contents of the LastBranchToIP and LastBranchFromIP MSRs are copied into these registers before the to and from addresses of the exception or interrupt are recorded in the LastBranchToIP and LastBranchFromIP MSRs.

These registers can be read using the RDMSR instruction.

Note that the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into the current code segment, as opposed to linear addresses, which are saved in last branch records for the Pentium 4 and Intel Xeon processors.

20.16.3 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag in the DEBUGCTLMSR register is set, the processor automatically begins recording branches that it takes, exceptions that are generated (except for debug exceptions), and interrupts that are serviced. Each time a branch, exception, or interrupt occurs, the processor records the to and from instruction pointers in the LastBranchToIP and LastBranchFromIP MSRs. In addition, for interrupts and exceptions, the processor copies the contents of the LastBranchToIP and LastBranchFromIP MSRs into the LastExceptionToIP and LastExceptionFromIP MSRs prior to recording the to and from addresses of the interrupt or exception.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler, but does not touch the last branch and last exception MSRs. The addresses for the last branch, interrupt, or exception taken are thus retained in the LastBranchToIP and LastBranchFromIP MSRs and the addresses of the last branch prior to an interrupt or exception are retained in the LastExceptionToIP, and LastExceptionFromIP MSRs.

The debugger can use the last branch, interrupt, and/or exception addresses in combination with code-segment selectors retrieved from the stack to reset breakpoints in the breakpoint-address registers (DR0 through DR3), allowing a backward trace from the manifestation of a particular bug toward its source. Because the instruction pointers recorded in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into a code segment, software must determine the segment base address of the code segment associated with the control transfer to calculate the linear address to be placed in the breakpoint-address registers. The segment base address can be determined by reading the segment selector for the code segment from the stack and using it to locate the segment descriptor for the segment in the GDT or LDT. The segment base address can then be read from the segment descriptor.

Before resuming program execution from a debug-exception handler, the handler must set the LBR flag again to re-enable last branch and last exception/interrupt recording.

20.17 TIME-STAMP COUNTER

The Intel 64 and IA-32 architectures (beginning with the Pentium processor) define a time-stamp counter mechanism that can be used to monitor and identify the relative time occurrence of processor events. The counter's architecture includes the following components:

- **TSC flag** — A feature bit that indicates the availability of the time-stamp counter. The counter is available in an if the function CPUID.01H:EDX.TSC[4] = 1.
- **IA32_TIME_STAMP_COUNTER MSR** (called TSC MSR in P6 family and Pentium processors) — The MSR used as the counter.
- **RDTSC instruction** — An instruction used to read the time-stamp counter.
- **TSD flag** — A control register flag is used to enable or disable the time-stamp counter (enabled if CR4.TSD[bit 2] = 1).

The time-stamp counter (as implemented in the P6 family, Pentium, Pentium M, Pentium 4, Intel Xeon, Intel Core Solo and Intel Core Duo processors and later processors) is a 64-bit counter that is set to 0 following a RESET of the processor. Following a RESET, the counter increments even when the processor is halted by the HLT instruction or the external STPCLK# pin. Note that the assertion of the external DPSLP# pin may cause the time-stamp counter to stop.

Processor families increment the time-stamp counter differently:

- For Pentium M processors (family [06H], models [09H, 0DH]); for Pentium 4 processors, Intel Xeon processors (family [0FH], models [00H, 01H, or 02H]); and for P6 family processors: the time-stamp counter increments with every internal processor clock cycle.

The internal processor clock cycle is determined by the current core-clock to bus-clock ratio. Intel® SpeedStep® technology transitions may also impact the processor clock.

- For Pentium 4 processors, Intel Xeon processors (family [0FH], models [03H and higher]); for Intel Core Solo and Intel Core Duo processors (family [06H], model [0EH]); for the Intel Xeon processor 5100 series and Intel Core 2 Duo processors (family [06H], model [0FH]); for Intel Core 2 and Intel Xeon processors (family [06H], DisplayModel [17H]); for Intel Atom processors (family [06H], DisplayModel [1CH]): the time-stamp counter increments at a constant rate. That rate may be set by the maximum core-clock to bus-clock ratio of the

processor or may be set by the maximum resolved frequency at which the processor is booted. The maximum resolved frequency may differ from the processor base frequency, see Section 22.7.2 for more detail. On certain processors, the TSC frequency may not be the same as the frequency in the brand string.

The specific processor configuration determines the behavior. Constant TSC behavior ensures that the duration of each clock tick is uniform and supports the use of the TSC as a wall clock timer even if the processor core changes frequency. This is the architectural behavior moving forward.

NOTE

To determine average processor clock frequency, Intel recommends the use of performance monitoring logic to count processor core clocks over the period of time for which the average is required. See Section 22.6.4.5, “Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture,” and <https://perfmon-events.intel.com/> for more information.

The RDTSC instruction reads the time-stamp counter and is guaranteed to return a monotonically increasing unique value whenever executed, except for a 64-bit counter wraparound. Intel guarantees that the time-stamp counter will not wraparound within 10 years after being reset. The period for counter wrap is longer for Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Normally, the RDTSC instruction can be executed by programs and procedures running at any privilege level and in virtual-8086 mode. The TSD flag allows use of this instruction to be restricted to programs and procedures running at privilege level 0. A secure operating system would set the TSD flag during system initialization to disable user access to the time-stamp counter. An operating system that disables user access to the time-stamp counter should emulate the instruction through a user-accessible programming interface.

The RDTSC instruction is not serializing or ordered with other instructions. It does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDTSC instruction operation is performed.

The RDMSR and WRMSR instructions read and write the time-stamp counter, treating the time-stamp counter as an ordinary MSR (address 10H). In the Pentium 4, Intel Xeon, and P6 family processors, all 64-bits of the time-stamp counter are read using RDMSR (just as with RDTSC). When WRMSR is used to write the time-stamp counter on processors before family [0FH], models [03H, 04H]: only the low-order 32-bits of the time-stamp counter can be written (the high-order 32 bits are cleared to 0). For family [0FH], models [03H, 04H, 06H]; for family [06H]], model [0EH, 0FH]; for family [06H]], DisplayModel [17H, 1AH, 1CH, 1DH]: all 64 bits are writable.

20.17.1 Invariant TSC

The time stamp counter in newer processors may support an enhancement, referred to as invariant TSC. Processor's support for invariant TSC is indicated by CPUID.80000007H:EDX[8].

The invariant TSC will run at a constant rate in all ACPI P-, C-, and T-states. This is the architectural behavior moving forward. On processors with invariant TSC support, the OS may use the TSC for wall clock timer services (instead of ACPI or HPET timers). TSC reads are much more efficient and do not incur the overhead associated with a ring transition or access to a platform resource.

20.17.2 IA32_TSC_AUX Register and RDTSCP Support

Processors based on Nehalem microarchitecture provide an auxiliary TSC register, IA32_TSC_AUX that is designed to be used in conjunction with IA32_TSC. IA32_TSC_AUX provides a 32-bit field that is initialized by privileged software with a signature value (for example, a logical processor ID).

The primary usage of IA32_TSC_AUX in conjunction with IA32_TSC is to allow software to read the 64-bit time stamp in IA32_TSC and signature value in IA32_TSC_AUX with the instruction RDTSCP in an atomic operation. RDTSCP returns the 64-bit time stamp in EDX:EAX and the 32-bit TSC_AUX signature value in ECX. The atomicity of RDTSCP ensures that no context switch can occur between the reads of the TSC and TSC_AUX values.

Support for RDTSCP is indicated by CPUID.80000001H:EDX[27]. As with RDTSC instruction, non-ring 0 access is controlled by CR4.TSD (Time Stamp Disable flag).

User mode software can use RDTSCP to detect if CPU migration has occurred between successive reads of the TSC. It can also be used to adjust for per-CPU differences in TSC values in a NUMA system.

20.17.3 Time-Stamp Counter Adjustment

Software can modify the value of the time-stamp counter (TSC) of a logical processor by using the WRMSR instruction to write to the IA32_TIME_STAMP_COUNTER MSR (address 10H). Because such a write applies only to that logical processor, software seeking to synchronize the TSC values of multiple logical processors must perform these writes on each logical processor. It may be difficult for software to do this in a way that ensures that all logical processors will have the same value for the TSC at a given point in time.

The synchronization of TSC adjustment can be simplified by using the 64-bit IA32_TSC_ADJUST MSR (address 3BH). Like the IA32_TIME_STAMP_COUNTER MSR, the IA32_TSC_ADJUST MSR is maintained separately for each logical processor. A logical processor maintains and uses the IA32_TSC_ADJUST MSR as follows:

- On RESET, the value of the IA32_TSC_ADJUST MSR is 0.
- If an execution of WRMSR to the IA32_TIME_STAMP_COUNTER MSR adds (or subtracts) value X from the TSC, the logical processor also adds (or subtracts) value X from the IA32_TSC_ADJUST MSR.
- If an execution of WRMSR to the IA32_TSC_ADJUST MSR adds (or subtracts) value X from that MSR, the logical processor also adds (or subtracts) value X from the TSC.

Unlike the TSC, the value of the IA32_TSC_ADJUST MSR changes only in response to WRMSR (either to the MSR itself, or to the IA32_TIME_STAMP_COUNTER MSR). Its value does not otherwise change as time elapses. Software seeking to adjust the TSC can do so by using WRMSR to write the same value to the IA32_TSC_ADJUST MSR on each logical processor.

Processor support for the IA32_TSC_ADJUST MSR is indicated by CPUID.07H.00H:EBX.TSC_ADJUST (bit 1).

20.17.4 Invariant Time-Keeping

The invariant TSC is based on the invariant timekeeping hardware (called Always Running Timer or ART), that runs at the core crystal clock frequency. The ratio defined by CPUID.15H expresses the frequency relationship between the ART hardware and TSC.

If CPUID.15H:EBX[31:0] != 0 and CPUID.80000007H:EDX.TSC_INVARIANT = 1, the following linearity relationship holds between TSC and the ART hardware:

$$\text{TSC\_Value} = (\text{ART\_Value} * \text{CPUID.15H:EBX}[31:0]) / \text{CPUID.15H:EAX}[31:0] + K$$

Where 'K' is an offset that can be adjusted by a privileged agent¹.

When ART hardware is reset, both invariant TSC and K are also reset.

20.18 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) MONITORING FEATURES

The Intel Resource Director Technology (Intel RDT) feature set provides a set of monitoring capabilities including Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring (MBM). The Intel® Xeon® processor E5 v3 family introduced resource monitoring capability in each logical processor to measure specific platform shared resource metrics, for example, L3 cache occupancy. The programming interface for these monitoring features is described in this section. Two features within the monitoring feature set provided are described - Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.

Cache Monitoring Technology (CMT) allows an Operating System, Hypervisor or similar system management agent to determine the usage of cache by applications running on the platform. The initial implementation is directed at L3 cache monitoring (currently the last level cache in most server platforms).

1. IA32_TSC_ADJUST MSR and the TSC-offset field in the VM execution controls of VMCS are some of the common interfaces that privileged software can use to manage the time stamp counter for keeping time

Memory Bandwidth Monitoring (MBM), introduced in the Intel® Xeon® processor E5 v4 family, builds on the CMT infrastructure to allow monitoring of bandwidth from one level of the cache hierarchy to the next - in this case focusing on the L3 cache, which is typically backed directly by system memory. As a result of this implementation, memory bandwidth can be monitored.

The monitoring mechanisms described provide the following key shared infrastructure features:

- A mechanism to enumerate the presence of the monitoring capabilities within the platform (via a CPUID feature bit).
- A framework to enumerate the details of each sub-feature (including CMT and MBM, as discussed later, via CPUID leaves and sub-leaves).
- A mechanism for the OS or Hypervisor to indicate a software-defined ID for each of the software threads (applications, virtual machines, etc.) that are scheduled to run on a logical processor. These identifiers are known as Resource Monitoring IDs (RMIDs).
- Mechanisms in hardware to monitor cache occupancy and bandwidth statistics as applicable to a given product generation on a per software-id basis.
- Mechanisms for the OS or Hypervisor to read back the collected metrics such as L3 occupancy or Memory Bandwidth for a given software ID at any point during runtime.

20.18.1 Overview of Cache Monitoring Technology and Memory Bandwidth Monitoring

The shared resource monitoring features described in this chapter provide a layer of abstraction between applications and logical processors through the use of **Resource Monitoring IDs** (RMIDs). Each logical processor in the system can be assigned an RMID independently, or multiple logical processors can be assigned to the same RMID value (e.g., to track an application with multiple threads). For each logical processor, only one RMID value is active at a time. This is enforced by the IA32_PQR_ASSOC MSR, which specifies the active RMID of a logical processor. Writing to this MSR by software changes the active RMID of the logical processor from an old value to a new value.

The underlying platform shared resource monitoring hardware tracks cache metrics such as cache utilization and misses as a result of memory accesses according to the RMIDs and reports monitored data via a counter register (IA32_QM_CTR). The specific event types supported vary by generation and can be enumerated via CPUID. To read back monitored data, software configures an event selection MSR (IA32_QM_EVTSEL) to specify which metric is to be reported and the specific RMID for which the data should be returned.

Processor support of the monitoring framework and sub-features such as CMT is reported via the CPUID instruction. The resource type available to the monitoring framework is enumerated via a new leaf function in CPUID. Reading and writing to the monitoring MSRs requires the RDMSR and WRMSR instructions.

The Cache Monitoring Technology feature set provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the CMT feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- CMT-specific event codes to read occupancy for a given level of the cache hierarchy.

The Memory Bandwidth Monitoring feature provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the MBM feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- MBM-specific event codes to read bandwidth out to the next level of the hierarchy and various sub-event codes to read more specific metrics as discussed later (e.g., total bandwidth vs. bandwidth only from local memory controllers on the same package).

20.18.2 Enabling Monitoring: Usage Flow

Figure 20-19 illustrates the key steps for OS/VMM to detect support of shared resource monitoring features such as CMT and enable resource monitoring for available resource types and monitoring events.

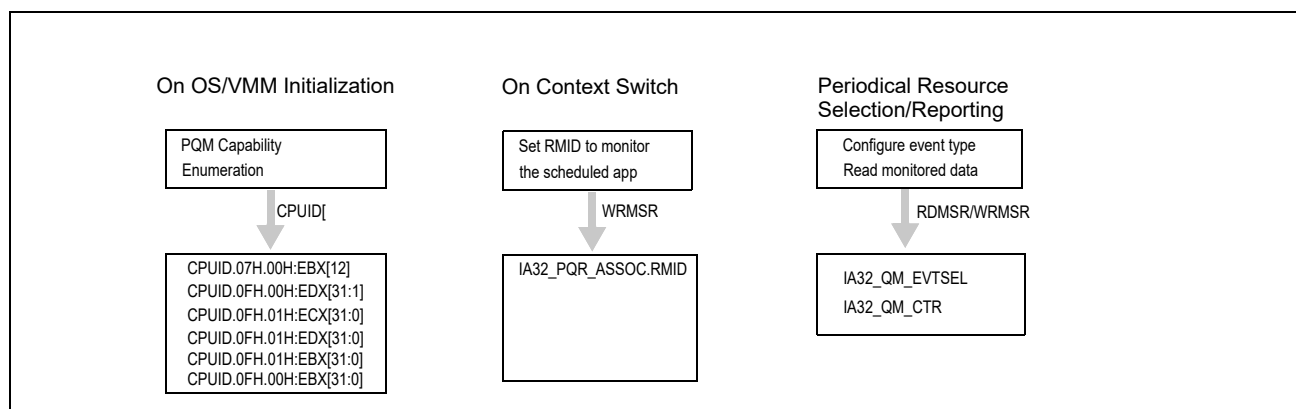


Figure 20-19. Platform Shared Resource Monitoring Usage Flow

20.18.3 Enumeration and Detecting Support of Cache Monitoring Technology and Memory Bandwidth Monitoring

Software can query processor support of shared resource monitoring features capabilities by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_M[12] reports 1, the processor provides the following programming interfaces for shared resource monitoring, including Cache Monitoring Technology:

- CPUID.0FH (Shared Resource Monitoring Enumeration leaf) provides information on available resource types (see Section 20.18.4), and monitoring capabilities for each resource type (see Section 20.18.5). Note CMT and MBM capabilities are enumerated as separate event vectors using shared enumeration infrastructure under a given resource type.
- IA32_PQR_ASSOC.RMID: The per-logical-processor MSR, IA32_PQR_ASSOC, that OS/VMM can use to assign an RMID to each logical processor, see Section 20.18.6.
- IA32_QM_EVTSEL: This MSR specifies an Event ID (EvtID) and an RMID which the platform uses to look up and provide monitoring data in the monitoring counter, IA32_QM_CTR, see Section 20.18.7.
- IA32_QM_CTR: This MSR reports monitored resource data when available along with bits to allow software to check for error conditions and verify data validity.

Software must follow the following sequence of enumeration to discover Cache Monitoring Technology capabilities:

1. Use CPUID.00H to discover the "cpuid_maxLeaf" supported in the processor;
2. If cpuid_maxLeaf >= 7, then verify CPUID.07H.00H:EBX.RDT_M[12] is set;
3. If CPUID.07H.00H:EBX.RDT_M[12] = 1, then use CPUID.0FH.00H to query available resource types that support monitoring;
4. If CPUID.0FH.00H:EDX.CMT_L3[1] = 1, then use CPUID.0FH.01H to query the specific capabilities of L3 Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.
5. If CPUID.0FH.00H:EDX reports additional resource types supporting monitoring, then use CPUID.0FH.ResID for a corresponding resource type ID (ResID) as enumerated by the bit position of CPUID.0FH.00H:EDX.

20.18.4 Monitoring Resource Type and Capability Enumeration

CPUID.0FH (Shared Resource Monitoring Enumeration leaf) provides one sub-leaf (sub-function 0) that reports shared enumeration infrastructure, and one or more sub-functions that report feature-specific enumeration data:

- Monitoring leaf sub-function 0 enumerates available resources that support monitoring, i.e., use CPUID.0FH.00H. In the initial implementation, L3 cache is the only resource type available. Each supported resource type is represented by a bit in CPUID.0FH.00H:EDX[31:1]. The bit position corresponds to the sub-leaf index (ResID) that software must use to query details of the monitoring capability of that resource type (see Figure 20-21 and Figure 20-22). Reserved bits of CPUID.0FH.00H:EDX[31:2] correspond to unsupported

sub-leaves of the CPUID.0FH leaf. Additionally, CPUID.0FH.00H:EBX reports the highest RMID value of any resource type that supports monitoring in the processor.

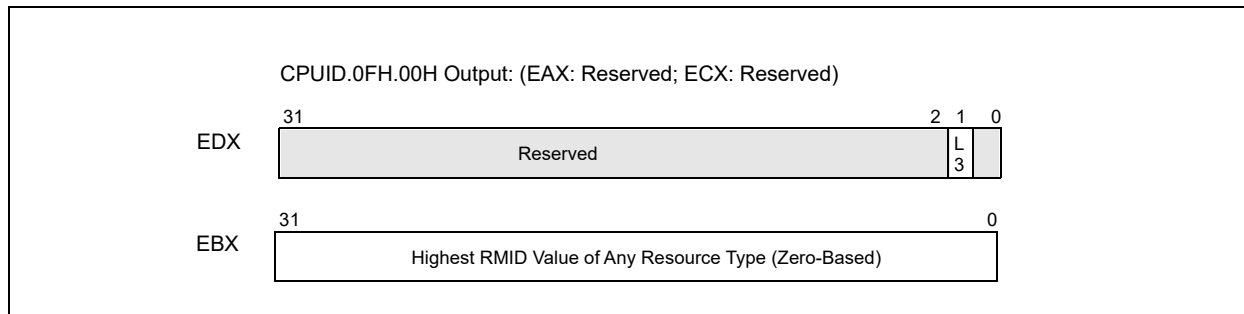


Figure 20-20. CPUID.0FH.00H Monitoring Resource Type Enumeration

20.18.5 Feature-Specific Enumeration

Each additional sub-leaf of CPUID.0FH.ResID enumerates the specific details for software to program monitoring MSRs using the resource type associated with the given ResID.

Note that in future Monitoring implementations the meanings of the returned registers may vary in other sub-leaves that are not yet defined. The registers will be specified and defined on a per-ResID basis.

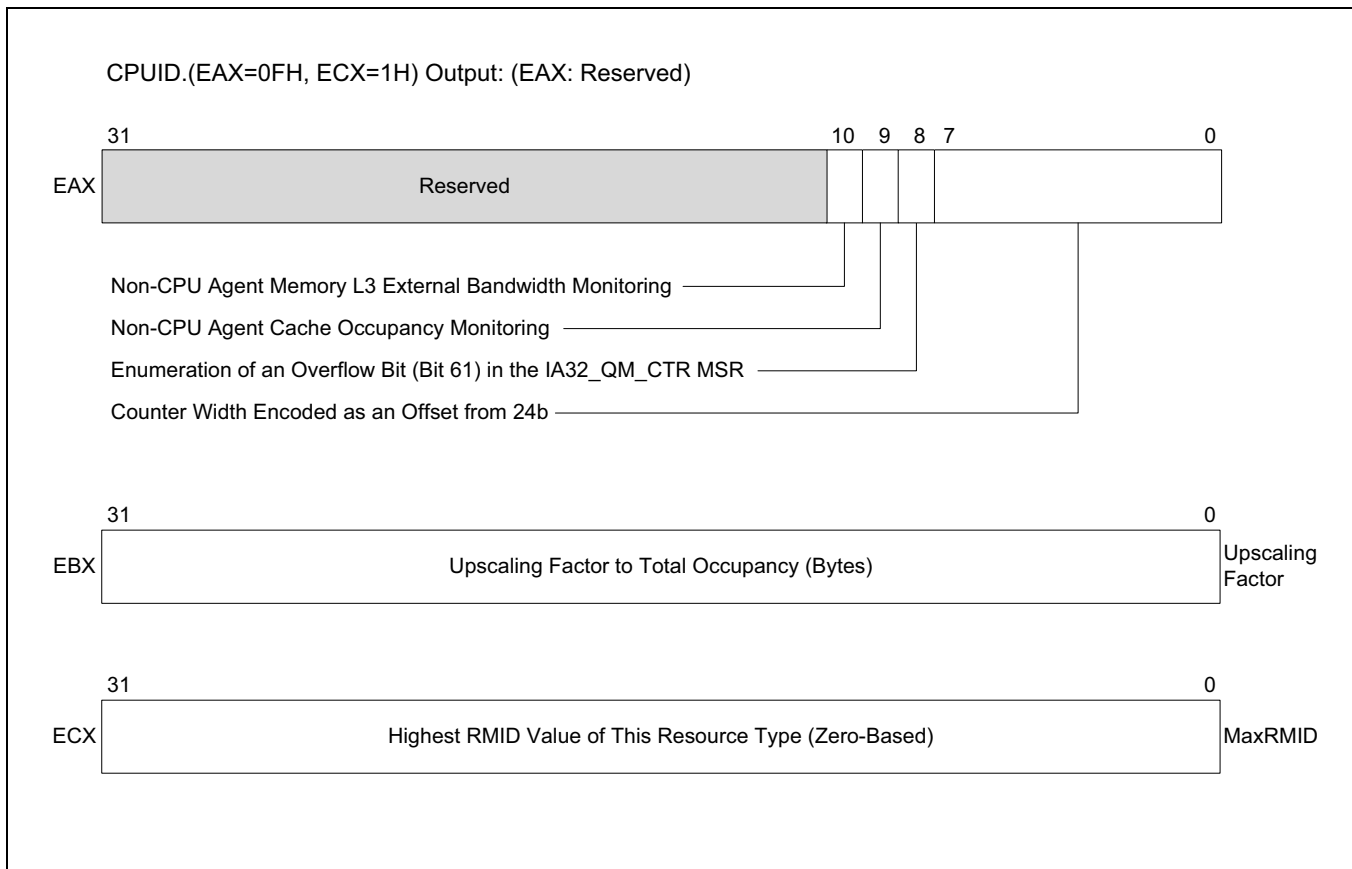


Figure 20-21. L3 Cache Monitoring Capability Enumeration Data (CPUID.0FH.01H)

CPUID.0FH.01H:EAX[7:0]: Encode counter width as offset from 24b. See Section 20.18.5.2 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[8]: If 1, indicates the presence of an overflow bit in the IA32_QM_CTR MSR. See Section 20.18.5.2 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[9]: If 1, indicates the presence of non-CPU agent Intel RDT CMT support. See Section 20.20 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[10]: If 1, indicates the presence of non-CPU agent Intel RDT MBM support. See Section 20.20 for details. Bits 31:11 of EAX are reserved.

For each supported Cache Monitoring resource type, hardware supports only a finite number of RMIDs.

CPUID.0FH.01H:ECX enumerates the highest RMID value that can be monitored with this resource type, see Figure 20-21.

CPUID.0FH.01H:EDX specifies a bit vector that is used to look up the EventID (See Figure 20-22 and Table 20-18) that software must program with IA32_QM_EVTSEL in order to retrieve event data. After software configures IA32_QMEVTSEL with the desired RMID and EventID, it can read the resulting data from IA32_QM_CTR. The raw numerical value reported from IA32_QM_CTR can be converted to the final value (occupancy in bytes or bandwidth in bytes per sampled time period) by multiplying the counter value by the value from CPUID.0FH.01H:EBX, see Figure 20-21.

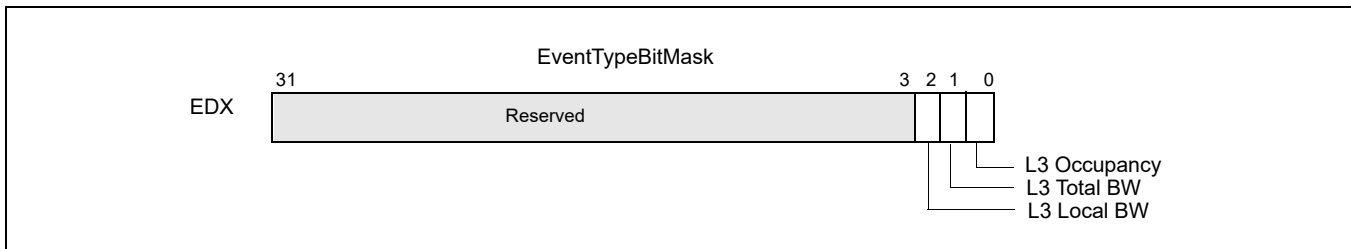


Figure 20-22. L3 Cache Monitoring Capability Enumeration Event Type Bit Vector (CPUID.0FH.01H)

20.18.5.1 Cache Monitoring Technology

On processors for which Cache Monitoring Technology supports the L3 cache occupancy event, CPUID.0FH.01H:EDX returns with bit 0 set. The corresponding event ID is shown in Table 20-18. The L3 occupancy data accumulated in the IA32_QM_CTR MSR can be converted to total occupancy (in bytes) by multiplying with CPUID.0FH.01H:EBX.

Event codes for Cache Monitoring Technology are discussed in the next section.

20.18.5.2 Memory Bandwidth Monitoring

On processors that monitoring supports Memory Bandwidth Monitoring using ResID=1 (L3), two additional bits are defined in the vector at CPUID.0FH.01H:EDX:

- CPUID.0FH.01H:EDX[1]: indicates the L3 total external bandwidth monitoring event is supported if set. This event monitors the L3 total external bandwidth to the next level of the cache hierarchy, including all demand and prefetch misses from the L3 to the next hierarchy of the memory system. In most platforms, this represents memory bandwidth.
- CPUID.0FH.01H:EDX[2]: indicates L3 local memory bandwidth monitoring event is supported if set. This event monitors the L3 external bandwidth satisfied by the local memory. In most platforms that support this event, L3 requests are likely serviced by a memory system with non-uniform memory architecture. This allows bandwidth to off-package memory resources to be tracked by subtracting local from total bandwidth (for instance, bandwidth over QPI to a memory controller on another physical processor could be tracked by subtraction). Note that it is not possible to read the local and total bandwidth atomically; multiple operations are needed. Because of this, it is possible for the counters to change in between the two reads.

The corresponding Event ID is shown in Table 20-18. The L3 bandwidth data accumulated in IA32_QM_CTR can be converted to total bandwidth (in bytes) using CPUID.0FH.01H:EBX.

Table 20-18. Monitoring Supported Event IDs

Event Type	Event ID	Context
L3 Cache Occupancy	01H	Cache Monitoring Technology
L3 Total External Bandwidth	02H	MBM
L3 Local External Bandwidth	03H	MBM
Reserved	All other event codes	N/A

A field is added to CPUID to enumerate the MBM counter width in platforms that support the extensible MBM counter width feature.

- CPUID.0FH.01H:EAX[7:0]: Encode counter width as offset from 24b in bits[7:0]. In EAX bits 7:0, the counter width is encoded as an offset from 24b. A value of zero in this field means 24-bit counters are supported. A value of 8 indicates that 32-bit counters are supported, as in the 3rd generation Intel Xeon Scalable Processor Family. With this enumerable counter width, the requirement that software polls at 1Hz is removed. Software may poll at a varying rate with a reduced risk of rollover. Under typical conditions, rollover will likely require hundreds of seconds (though this value is not explicitly specified and may vary and decrease in future processor generations as memory bandwidths increase). Suppose software seeks to ensure that rollover does not occur more than once between samples. In that case, sampling at 1Hz while consuming the enumerated counter widths' worth of data will provide this guarantee for a specific platform and counter width under all conditions.
- CPUID.0FH.01H:EAX[8]: Enumeration of the presence of an overflow bit in the IA32_QM_CTR MSR via EAX bit[8]. Software that uses the MBM event retrieval MSR interface should be updated to comprehend this new format, which enables up to 62-bit MBM counters to be provided by future platforms. Higher-level software that consumes the resulting bandwidth values is not expected to be affected. An overflow bit is defined in the IA32_QM_CTR MSR, bit 61, if CPUID.0FH.01H:EAX[8] is set. This rollover bit will be set on overflow of the MBM counters and reset upon read. Current processors do not support this capability.

20.18.6 Monitoring Resource RMID Association

After Monitoring and sub-features have been enumerated, software can begin using the monitoring features. The first step is to associate a given software thread (or multiple threads as part of an application, VM, group of applications or other abstraction) with an RMID.

Note that the process of associating an RMID with a given software thread is the same for all shared resource monitoring features (CMT, MBM), and a given RMID number has the same meaning from the viewpoint of any logical processors in a package. Stated another way, a thread may be associated in a 1:1 mapping with an RMID, and that RMID may allow cache occupancy, memory bandwidth information or other monitoring data to be read back later with monitoring event codes (retrieving data is discussed in a previous section).

The association of an application thread with an RMID requires an OS to program the per-logical-processor MSR IA32_PQR_ASSOC at context swap time (updates may also be made at any other arbitrary points during program execution such as application phase changes). The IA32_PQR_ASSOC MSR specifies the active RMID that monitoring hardware will use to tag internal operations, such as L3 cache requests. The layout of the MSR is shown in Figure 20-23. Software specifies the active RMID to monitor in the IA32_PQR_ASSOC.RMID field. The width of the RMID field can vary from one implementation to another, and is derived from $\text{Ceil}(\text{LOG}_2(1 + \text{CPUID.0FH.00H:EBX}[31:0]))$. The value of IA32_PQR_ASSOC after power-on is 0.

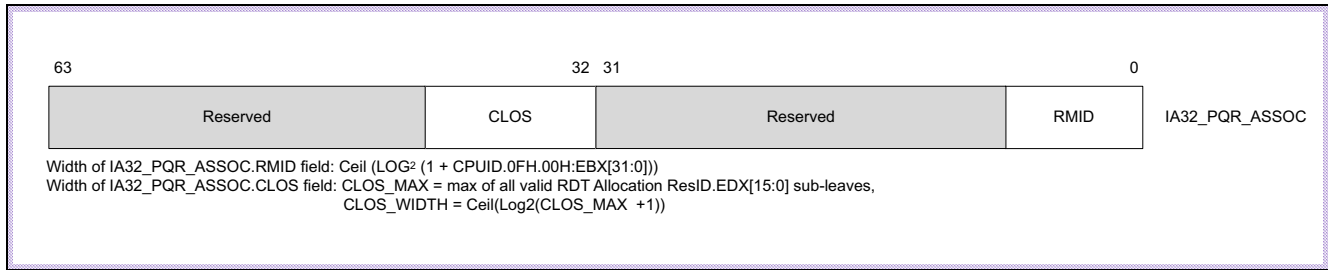


Figure 20-23. IA32_PQR_ASSOC MSR

In the initial implementation, the width of the RMID field is up to 10 bits wide, zero-referenced and fully encoded. However, software must use CPUID to query the maximum RMID supported by the processor. If a value larger than the maximum RMID is written to IA32_PQR_ASSOC.RMID, a #GP(0) fault will be generated.

RMIDs have a global scope within the physical package- if an RMID is assigned to one logical processor then the same RMID can be used to read multiple thread attributes later (for example, L3 cache occupancy or external bandwidth from the L3 to the next level of the cache hierarchy). In a multiple LLC platform the RMIDs are to be reassigned by the OS or VMM scheduler when an application is migrated across LLCs.

Note that in a situation where Monitoring supports multiple resource types, some upper range of RMIDs (e.g., RMID 31) may only be supported by one resource type but not by another resource type.

20.18.7 Monitoring Resource Selection and Reporting Infrastructure

The reporting mechanism for Cache Monitoring Technology and other related features is architecturally exposed as an MSR pair that can be programmed and read to measure various metrics such as the L3 cache occupancy (CMT) and bandwidths (MBM) depending on the level of Monitoring support provided by the platform. Data is reported back on a per-RMID basis. These events do not trigger based on event counts or trigger APIC interrupts (e.g., no Performance Monitoring Interrupt occurs based on counts). Rather, they are used to sample counts explicitly.

The MSR pair for the shared resource monitoring features (CMT, MBM) is separate from and not shared with architectural PerfMon counters, meaning software can use these monitoring features simultaneously with the PerfMon counters.

Access to the aggregated monitoring information is accomplished through the following programmable monitoring MSRs:

- **IA32_QM_EVTSEL:** This MSR provides a role similar to the event select MSRs for programmable performance monitoring described in Chapter 18. The simplified layout of the MSR is shown in Figure 20-24. IA32_QM_EVTSEL.EvtID (bits 7:0) specifies an event code of a supported resource type for hardware to report monitored data associated with IA32_QM_EVTSEL.RMID (bits 41:32). Software can configure IA32_QM_EVTSEL.RMID with any RMID that is active within the physical processor. The width of IA32_QM_EVTSEL.RMID matches that of IA32_PQR_ASSOC.RMID. Supported event codes for the IA32_QM_EVTSEL register are shown in Table 20-18. Note that valid event codes may not necessarily map directly to the bit position used to enumerate support for the resource via CPUID.

Software can program an RMID / Event ID pair into the IA32_QM_EVTSEL MSR bit field to select an RMID to read a particular counter for a given resource. The currently supported list of Monitoring Event IDs is discussed in Section 20.18.5, which covers feature-specific details.

Thread access to the IA32_QM_EVTSEL and IA32_QM_CTR MSR pair should be serialized (that is, treated as a critical section under lock) to avoid situations where one thread changes the RMID/EvtID just before another thread reads monitoring data from IA32_QM_CTR.

- **IA32_QM_CTR:** This MSR reports monitored data when available. It contains three bit fields. If software configures an unsupported RMID or event type in IA32_QM_EVTSEL, then IA32_QM_CTR.Error (bit 63) will be set, indicating there is no valid data to report. If IA32_QM_CTR.Unavailable (bit 62) is set, it indicates monitored data for the RMID is not available, and IA32_QM_CTR.data (bits 61:0) should be ignored. Therefore, IA32_QM_CTR.data (bits 61:0) is valid only if bits 63 and 62 are both clear. The IA32_QM_CTR.Overflow (bit 61) is present if CPUID.0FH.01H:EAX[8] is set. This bit is set on overflow of the MBM counters and will reset

upon read. For Cache Monitoring Technology, software can convert IA32_QM_CTR.data into cache occupancy or bandwidth metrics expressed in bytes by multiplying with the conversion factor from CPUID.0FH.01H:EBX.

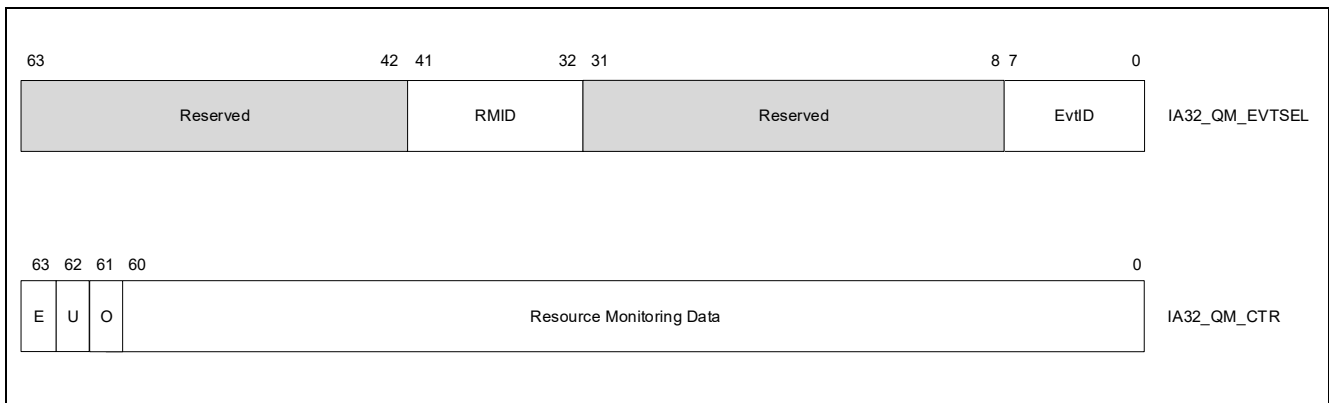


Figure 20-24. IA32_QM_EVTSEL and IA32_QM_CTR MSRs

20.18.8 Monitoring Programming Considerations

Figure 20-25 illustrates how system software can program IA32_QM_EVTSEL and IA32_QM_CTR to perform resource monitoring.

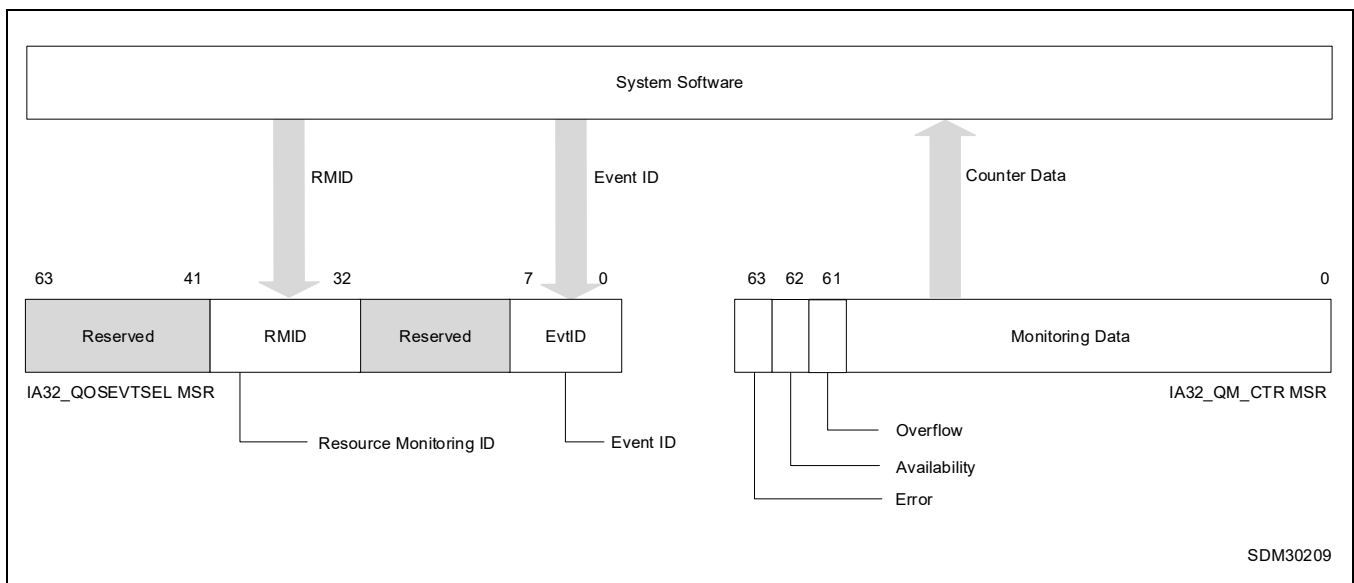


Figure 20-25. Software Usage of Cache Monitoring Resources

Though the field provided in IA32_QM_CTR allows for up to 62 bits of data to be returned, often a subset of bits are used. With Cache Monitoring Technology for instance, the number of bits used is the base-two logarithm of the total cache size divided by the Upscaling Factor from CPUID.

In Memory Bandwidth Monitoring, the initial counter size is 24 bits, and retrieving the value at 1Hz or faster is sufficient to ensure at most one rollover per sampling period. Any changes to counter width are enumerated to software; see Section 20.18.5.2 for details.

20.18.8.1 Monitoring Dynamic Configuration

Both the IA32_QM_EVTSEL and IA32_PQR_ASSOC registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,
- An RMID exceeding the maximum RMID is used.

20.18.8.2 Monitoring Operation With Power Saving Features

Some advanced power management features such as deep package C-states may shrink the L3 cache and cause CMT occupancy count to be reduced. MBM bandwidth counts may increase due to flushing cached data out of L3.

20.18.8.3 Monitoring Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and monitoring counter are unmodified across an SMI delivery. Thus, the execution of SMM handler code and SMM handler's data can manifest as spurious contribution in the monitored data.

It is possible for an SMM handler to minimize the impact on of spurious contribution in the QOS monitoring counters by reserving a dedicated RMID for monitoring the SMM handler. Such an SMM handler can save the previously configured QOS Monitoring state immediately upon entering SMM, and restoring the QOS monitoring state back to the prev-SMM RMID upon exit.

20.18.8.4 Monitoring Operation with RAS Features

In general, the Reliability, Availability, and Serviceability (RAS) features present in Intel Platforms are not expected to significantly affect shared resource monitoring counts. In cases where software RAS features cause memory copies or cache accesses, these may be tracked and may influence the shared resource monitoring counter values.

20.19 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) ALLOCATION FEATURES

The Intel Resource Director Technology (Intel RDT) feature set provides a set of allocation (resource control) capabilities including Cache Allocation Technology (CAT) and Code and Data Prioritization (CDP). The Intel Xeon processor E5 v4 family (and a subset of communication-focused processors in the Intel Xeon E5 v3 family) introduce capabilities to configure and make use of the Cache Allocation Technology (CAT) mechanisms on the L3 cache. Certain Intel Atom processors also provide support for control over the L2 cache, with capabilities as described below. The programming interface for Cache Allocation Technology and for the more general allocation capabilities are described in the rest of this chapter. The CAT and CDP capabilities, where architecturally supported, may be detected and enumerated in software using the CPUID instruction, as described in this chapter.

The Intel Xeon Scalable Processor Family introduces the Memory Bandwidth Allocation (MBA) feature which provides indirect control over the memory bandwidth available to CPU cores, and is discussed later in this chapter.

20.19.1 Introduction to Cache Allocation Technology (CAT)

Cache Allocation Technology enables an Operating System (OS), Hypervisor /Virtual Machine Manager (VMM) or similar system service management agent to specify the amount of cache space into which an application can fill (as a hint to hardware - certain features such as power management may override CAT settings). Specialized user-level implementations with minimal OS support are also possible, though not necessarily recommended (see notes below for OS/Hypervisor with respect to ring 3 software and virtual guests). Depending on the processor family, L2 or L3 cache allocation capability may be provided, and the technology is designed to scale across multiple cache levels and technology generations.

Software can determine which levels are supported in a given platform programmatically using CPUID as described in the following sections.

The CAT mechanisms defined in this document provide the following key features:

- A mechanism to enumerate platform Cache Allocation Technology capabilities and available resource types that provides CAT control capabilities. For implementations that support Cache Allocation Technology, CPUID provides enumeration support to query which levels of the cache hierarchy are supported and specific CAT capabilities, such as the max allocation bitmask size.
- A mechanism for the OS or Hypervisor to configure the amount of a resource available to a particular Class of Service via a list of allocation bitmasks.
- Mechanisms for the OS or Hypervisor to signal the Class of Service to which an application belongs.
- Hardware mechanisms to guide the LLC fill policy when an application has been designated to belong to a specific Class of Service.

Note that for many usages, an OS or Hypervisor may not want to expose Cache Allocation Technology mechanisms to Ring3 software or virtualized guests.

The Cache Allocation Technology feature enables more cache resources (i.e., cache space) to be made available for high priority applications based on guidance from the execution environment as shown in Figure 20-26. The architecture also allows dynamic resource reassignment during runtime to further optimize the performance of the high priority application with minimal degradation to the low priority app. Additionally, resources can be rebalanced for system throughput benefit across uses cases of OSes, VMMs, containers, and other scenarios by managing the CPUID and MSR interfaces. This section describes the hardware and software support required in the platform including what is required of the execution environment (i.e., OS/VMM) to support such resource control. Note that in Figure 20-26 the L3 Cache is shown as an example resource.

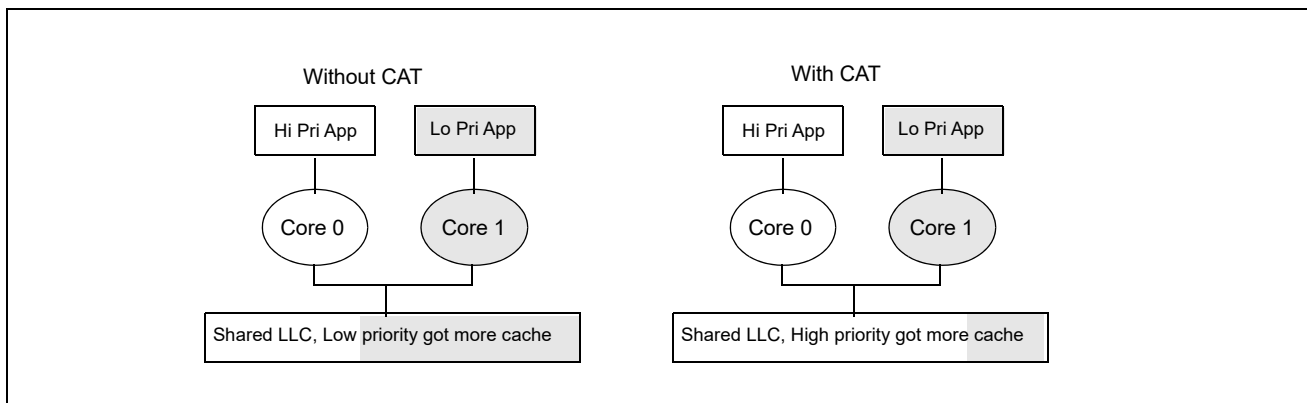


Figure 20-26. Cache Allocation Technology Enables Allocation of More Resources to High Priority Applications

20.19.2 Cache Allocation Technology Architecture

The fundamental goal of Cache Allocation Technology is to enable resource allocation based on application priority or Class of Service (CLOS). The processor exposes a set of Classes of Service into which applications (or individual threads) can be assigned. Cache allocation for the respective applications or threads is then restricted based on the class with which they are associated. Each Class of Service can be configured using capacity bitmasks (CBMs) which represent capacity and indicate the degree of overlap and isolation between classes. For each logical processor there is a register exposed (referred to here as the IA32_PQR_ASSOC MSR or PQR) to allow the OS/VMM to specify a CLOS when an application, thread or VM is scheduled.

The usage of Classes of Service (CLOS) are consistent across resources and a CLOS may have multiple resource control attributes attached, which reduces software overhead at context swap time. Rather than adding new types of CLOS tags per resource for instance, the CLOS management overhead is constant. Cache allocation for the indicated application/thread/container/VM is then controlled automatically by the hardware based on the class and the bitmask associated with that class. Bitmasks are configured via the IA32_resourceType_MASK_n MSRs, where resourceType indicates a resource type (e.g., "L3" for the L3 cache) and "n" indicates a CLOS number.

The basic ingredients of Cache Allocation Technology are as follows:

- An architecturally exposed mechanism using CPUID to indicate whether CAT is supported, and what resource types are available which can be controlled.

- For each available resourceType, CPUID also enumerates the total number of Classes of Services and the length of the capacity bitmasks that can be used to enforce cache allocation to applications on the platform.
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to configure the behavior of different classes of service using the bitmasks available.
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to assign a CLOS to an executing software thread (i.e., associating the active CR3 of a logical processor with the CLOS in IA32_PQR_ASSOC).
- Implementation-dependent mechanisms to indicate which CLOS is associated with a memory access and to enforce the cache allocation on a per CLOS basis.

A capacity bitmask (CBM) provides a hint to the hardware indicating the cache space an application should be limited to as well as providing an indication of overlap and isolation in the CAT-capable cache from other applications contending for the cache. The bit length of the capacity mask available generally depends on the configuration of the cache and is specified in the enumeration process for CAT in CPUID (this may vary between models in a processor family as well). Similarly, other parameters such as the number of supported CLOS may vary for each resource type, and these details can be enumerated via CPUID.

	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A	A	A	A	A	Default Bitmask
CLOS1	A	A	A	A	A	A	A	A	
CLOS2	A	A	A	A	A	A	A	A	
CLOS3	A	A	A	A	A	A	A	A	
	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A	A	A	A	A	Overlapped Bitmask
CLOS1					A	A	A	A	
CLOS2							A	A	
CLOS3								A	
	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A					Isolated Bitmask
CLOS1					A	A			
CLOS2							A		
CLOS3								A	

Figure 20-27. Examples of Cache Capacity Bitmasks

Sample cache capacity bitmasks for a bit length of 8 are shown in Figure 20-27. Note that all (and only) contiguous '1' combinations are allowed (e.g., FFFFH, 0FF0H, 003CH, etc.), unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type. Attempts to program a value without contiguous '1's (including zero) will result in a general protection fault (#GP(0)). It is generally expected that in way-based implementations, one capacity mask bit corresponds to some number of ways in cache, but the specific mapping is implementation-dependent. In all cases, a mask bit set to '1' specifies that a particular Class of Service can allocate into the cache subset represented by that bit. A value of '0' in a mask bit specifies that a Class of Service cannot

allocate into the given cache subset. In general, allocating more cache to a given application is usually beneficial to its performance.

Figure 20-27 also shows three examples of sets of Cache Capacity Bitmasks. For simplicity these are represented as 8-bit vectors, though this may vary depending on the implementation and how the mask is mapped to the available cache capacity. The first example shows the default case where all 4 Classes of Service (the total number of CLOS are implementation-dependent) have full access to the cache. The second case shows an overlapped case, which would allow some lower-priority threads to share cache space with the highest priority threads. The third case shows various non-overlapped partitioning schemes. As a matter of software policy for extensibility, CLOS0 should typically be considered and configured as the highest priority CLOS, followed by CLOS1, and so on, though there is no hardware restriction enforcing this mapping. When the system boots all threads are initialized to CLOS0, which has full access to the cache by default.

Though the representation of the CBMs looks similar to a way-based mapping they are independent of any specific enforcement implementation (e.g., way partitioning.) Rather, this is a convenient manner to represent capacity, overlap, and isolation of cache space. For example, executing a POPCNT instruction (population count of set bits) on the capacity bitmask can provide the fraction of cache space that a class of service can allocate into. In addition to the fraction, the exact location of the bits also shows whether the class of service overlaps with other classes of service or is entirely isolated in terms of cache space used.

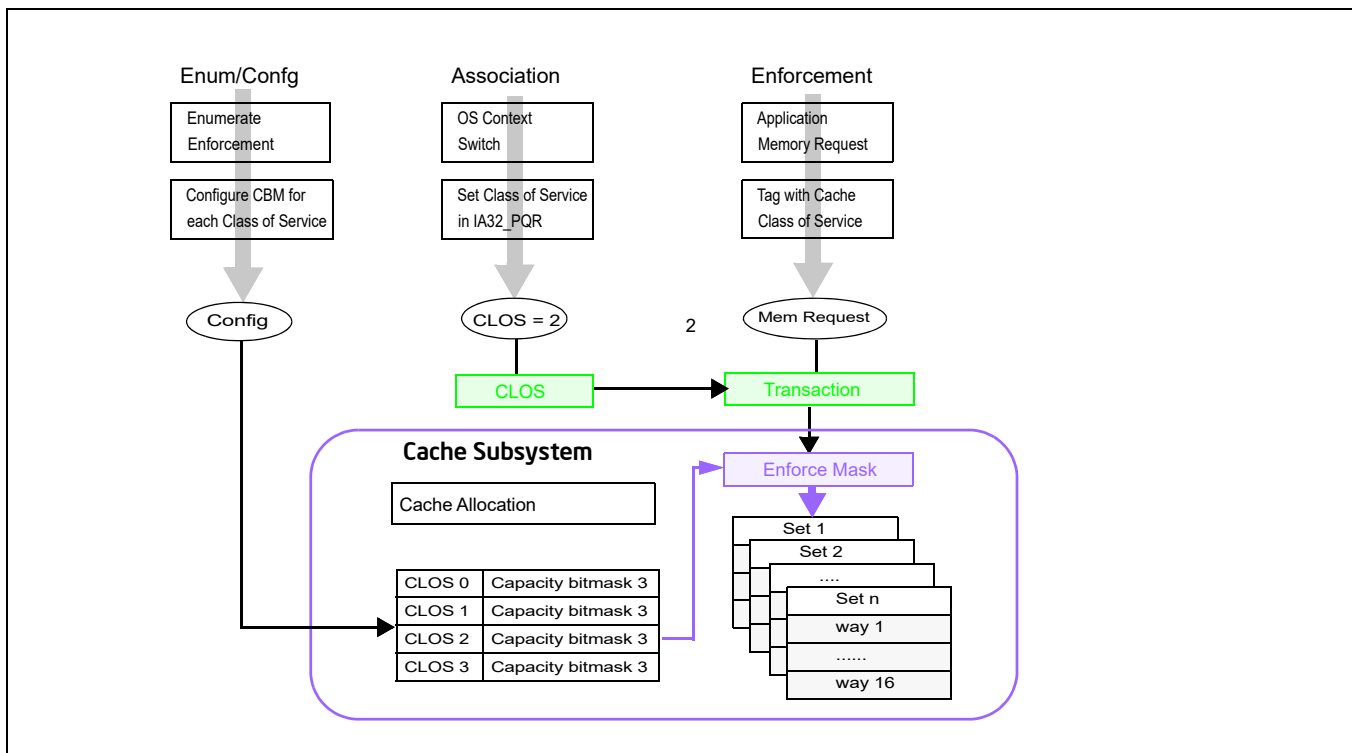


Figure 20-28. Class of Service and Cache Capacity Bitmasks

Figure 20-28 shows how the Cache Capacity Bitmasks and the per-logical-processor Class of Service are logically used to enable Cache Allocation Technology. All (and only) contiguous 1's in the CBM are permitted, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type. The length of a CBM may vary from resource to resource or between processor generations and can be enumerated using CPUID. From the available mask set and based on the goals of the OS/VMM (shared or isolated cache, etc.) bitmasks are selected and associated with different classes of service. For the available Classes of Service the associated CBMs can be programmed via the global set of CAT configuration registers (in the case of L3 CAT, via the IA32_L3_MASK_n MSRs, where "n" is the Class of Service, starting from zero). In all architectural implementations supporting CPUID it is possible to change the CBMs dynamically, during program execution, unless stated otherwise by Intel.

The currently running application's Class of Service is communicated to the hardware through the per-logical-processor PQR MSR (IA32_PQR_ASSOC MSR). When the OS schedules an application thread on a logical processor, the application thread is associated with a specific CLOS (i.e., the corresponding CLOS in the PQR) and all requests to the CAT-capable resource from that logical processor are tagged with that CLOS (in other words, the application thread is configured to belong to a specific CLOS). The cache subsystem uses this tagged request information to enforce QoS. The capacity bitmask may be mapped into a way bitmask (or a similar enforcement entity based on the implementation) at the cache before it is applied to the allocation policy. For example, the capacity bitmask can be an 8-bit mask and the enforcement may be accomplished using a 16-way bitmask for a cache enforcement implementation based on way partitioning.

The following sections describe extensions of CAT such as Code and Data Prioritization (CDP), followed by details on specific features such as L3 CAT, L3 CDP, L2 CAT, and L2 CDP. Depending on the specific processor a mix of features may be supported, and CPUID provides enumeration capabilities to enable software to dynamically detect the set of supported features.

20.19.3 Code and Data Prioritization (CDP) Technology

Code and Data Prioritization Technology is an extension of CAT. CDP enables isolation and separate prioritization of code and data fetches to the L2 or L3 cache in a software configurable manner, depending on hardware support, which can enable workload prioritization and tuning of cache capacity to the characteristics of the workload. CDP extends Cache Allocation Technology (CAT) by providing separate code and data masks per Class of Service (CLOS). Support for the L2 CDP feature and the L3 CDP features are separately enumerated (via CPUID) and separately controlled (via remapping the L2 CAT MSRs or L3 CAT MSRs respectively). Section 20.19.6.3 and Section 20.19.7 provide details on enumerating, controlling, and enabling L3 and L2 CDP respectively, while this section provides a general overview.

The L3 CDP feature was first introduced on the Intel Xeon E5 v4 family of server processors, as an extension to L3 CAT. The L2 CDP feature is first introduced on future Intel Atom family processors, as an extension to L2 CAT.

By default, CDP is disabled on the processor. If the CAT MSRs are used without enabling CDP, the processor operates in a traditional CAT-only mode. When CDP is enabled:

- The CAT mask MSRs are re-mapped into interleaved pairs of mask MSRs for data or code fetches (see Figure 20-29).
- The range of CLOS for CAT is re-indexed, with the lower-half of the CLOS range available for CDP.

Using the CDP feature, virtual isolation between code and data can be configured on the L2 or L3 cache if desired, similar to how some processor cache levels provide separate L1 data and L1 instruction caches.

Like the CAT feature, CDP may be dynamically configured by privileged software at any point during normal system operation, including dynamically enabling or disabling the feature provided that certain software configuration requirements are met (see Section 20.19.5).

An example of the operating mode of CDP is shown in Figure 20-29. Shown at the top are traditional CAT usage models where capacity masks map 1:1 with a CLOS number to enable control over the cache space which a given CLOS (and thus applications, threads or VMs) may occupy. Shown at the bottom are example mask configurations where CDP is enabled, and each CLOS number maps 1:2 to two masks, one for code and one for data. This enables code and data to be either overlapped or isolated to varying degrees either globally or on a per-CLOS basis, depending on application and system needs.

Example of CAT-Only Usage - 16 bit Capacity Masks															
CLOS0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
CLOS1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
CLOS2	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
CLOS3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

Traditional CAT

Example of Code/Data Prioritization Usage - 16 bit Capacity Masks															
CLOS0.Data	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
CLOS0.Code	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
CLOS1.Data	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0
CLOS1.Code	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
Other CLOS.Data	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Other CLOS.Code	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

CAT with CDP

Figure 20-29. Code and Data Capacity Bitmasks of CDP

When CDP is enabled, the existing mask space for CAT-only operation is split. As an example if the system supports 16 CAT-only CLOS, when CDP is enabled the same MSR interfaces are used, however half of the masks correspond to code, half correspond to data, and the effective number of CLOS is reduced by half. Code/Data masks are defined per-CLOS and interleaved in the MSR space as described in subsequent sections.

In cases where CPUID exposes a non-even number of supported Classes of Service for the CAT or CDP features, software using CDP should use the lower matched pairs of code/data masks, and any upper unpaired masks should not be used. As an example, if CPUID exposes 5 CLOS, when CDP is enabled then two code/data pairs are available (masks 0/1 for CLOS[0] data/code and masks 2/3 for CLOS[1] data/code), however the upper un-paired mask should not be used (mask 4 in this case) or undefined behavior may result.

20.19.4 Enabling Cache Allocation Technology Usage Flow

Figure 20-30 illustrates the key steps for OS/VMM to detect support of Cache Allocation Technology and enable priority-based resource allocation for a CAT-capable resource.

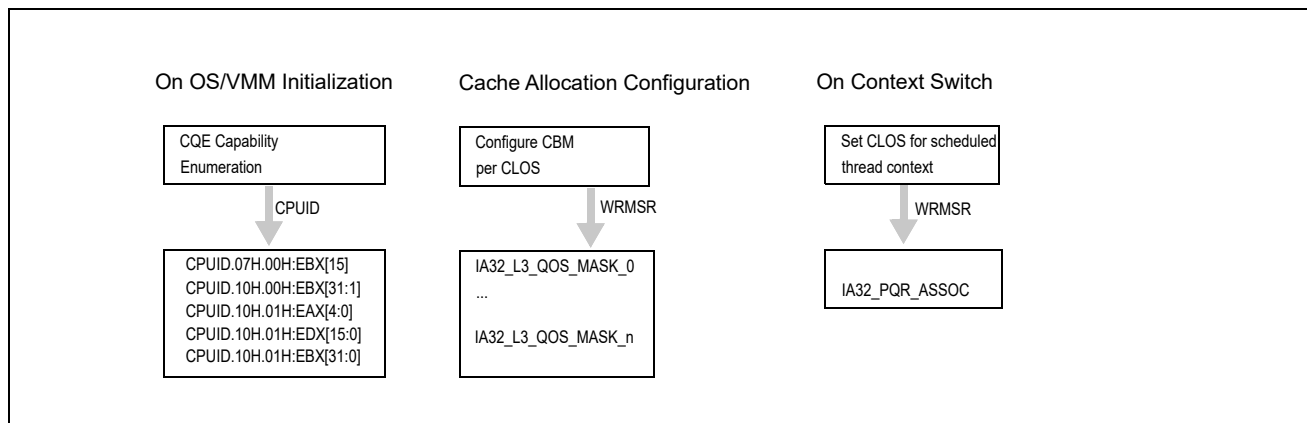


Figure 20-30. Cache Allocation Technology Usage Flow

Enumeration and configuration of L2 CAT is similar to L3 CAT, however CPUID details and MSR addresses differ. Common CLOS are used across the features.

20.19.4.1 Enumeration and Detection Support of Cache Allocation Technology

Software can query processor support of CAT capabilities by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_A[15] reports 1, the processor supports software control over shared processor resources. Software must use CPUID.10H to enumerate additional details of available resource types, classes of services and capability bitmasks. The programming interfaces provided by Cache Allocation Technology include:

- CPUID.10H (Cache Allocation Technology Enumeration leaf) and its sub-functions provide information on available resource types, and CAT capability for each resource type (see Section 20.19.4.2).
- IA32_L3_MASK_n: A range of MSRs is provided for each resource type, each MSR within that range specifying a software-configured capacity bitmask for each class of service. For L3 with Cache Allocation support, the CBM is specified using one of the IA32_L3_QOS_MASK_n MSR, where 'n' corresponds to a number within the supported range of CLOS, i.e., the range between 0 and CPUID.10H.ResID:EDX[15:0], inclusive. See Section 20.19.4.3 for details.
- IA32_L2_MASK_n: A range of MSRs is provided for L2 Cache Allocation Technology, enabling software control over the amount of L2 cache available for each CLOS. Similar to L3 CAT, a CBM is specified for each CLOS using the set of registers, IA32_L2_QOS_MASK_n MSR, where 'n' ranges from zero to the maximum CLOS number reported for L2 CAT in CPUID. See Section 20.19.4.3 for details.

The L2 mask MSRs are scoped at the same level as the L2 cache (similarly, the L3 mask MSRs are scoped at the same level as the L3 cache). Software may determine which logical processors share an MSR (for instance local to a core, or shared across multiple cores) by performing a write to one of these MSRs and noting which logical threads observe the change. Example flows for a similar method to determine register scope are described in Section 18.5.2, “System Software Recommendation for Managing CMCI and Machine Check Resources.” Software may also use CPUID.04H to determine the maximum number of logical processor IDs that may share a given level of the cache.

- IA32_PQR_ASSOC.CLOS: The IA32_PQR_ASSOC MSR provides a CLOS field that OS/VMM can use to assign a logical processor to an available CLOS. The set of CLOS are common across all allocation features, meaning that multiple features may be supported in the same processor without additional software CLOS management overhead at context swap time. See Section 20.19.4.4 for details.

20.19.4.2 Cache Allocation Technology: Resource Type and Capability Enumeration

CPUID.10H (Cache Allocation Technology Enumeration leaf) provides two or more sub-functions:

- CAT Enumeration leaf sub-function 0 enumerates available resource types that support allocation control, i.e., by using CPUID.10H.00H. Each supported resource type is represented by a bit field in CPUID.10H.00H:EBX[31:1]. The bit position of each set bit corresponds to a Resource ID (ResID), for instance ResID=1 is used to indicate L3 CAT support, and ResID=2 indicates L2 CAT support. The ResID is also the sub-leaf index that software must use to query details of the CAT capability of that resource type (see Figure 20-31).

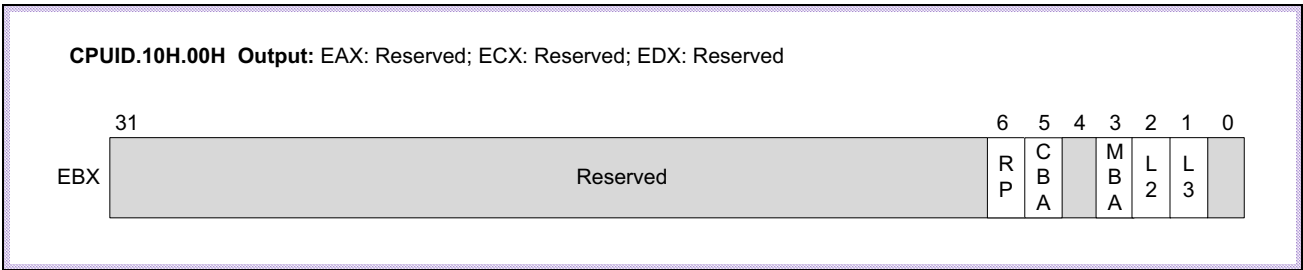


Figure 20-31. CPUID.10H.00H Available Resource Type Identification

- For ECX>0, EAX[4:0] reports the length of the capacity bitmask (ECX=1 or 2 for L3 CAT or L2 CAT respectively). Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- Sub-leaves of CPUID.10H with a non-zero ECX input matching a supported ResID enumerate the specific enforcement details of the corresponding ResID. The capabilities enumerated include the length of the capacity bitmasks and the number of Classes of Service for a given ResID. Software should query the capability of each available ResID that supports CAT from a sub-leaf of leaf 10H using the sub-leaf index reported by the corresponding non-zero bit in CPUID.10H.00H:EBX[31:1] in order to obtain additional feature details.

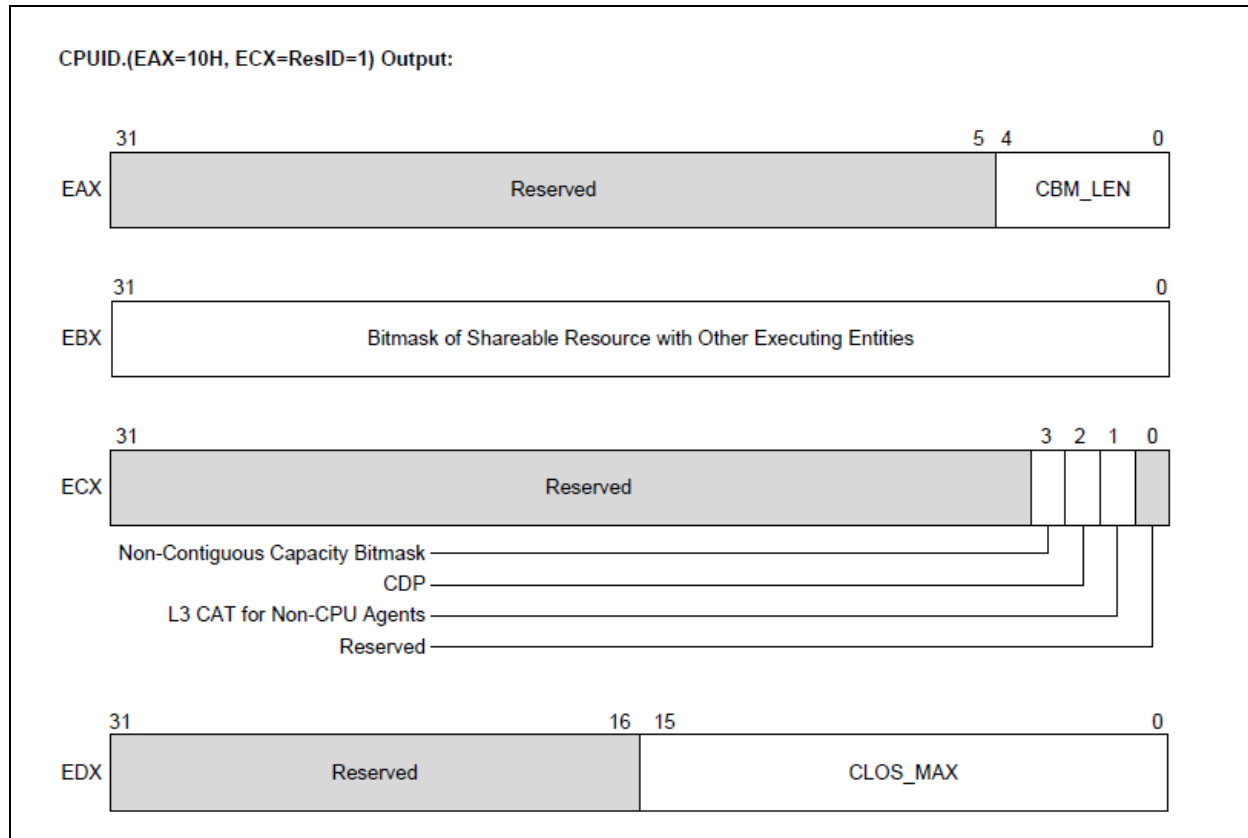


Figure 20-32. L3 Cache Allocation Technology and CDP Enumeration

- CAT capability for L3 is enumerated by CPUID.10H.01H, see Figure 20-32. The specific CAT capabilities reported by CPUID.10H.01H are:
 - CPUID.10H.ResID=1:EAX[4:0] reports the length of the capacity bitmask. Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
 - CPUID.10H.01H:EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L3 allocation may be used by other entities in the platform (e.g., an integrated graphics engine or hardware units outside the processor core and have direct access to L3). Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.
 - CPUID.10H.01H:ECX[1]: If 1, indicates L3 CAT for non-CPU agents is supported. Bits 0 and 31:4 of ECX are reserved. See section 18.20 for details.
 - CPUID.10H.01H:ECX.CDP[2]: If 1, indicates L3 Code and Data Prioritization Technology is supported (see Section 20.19.5). Bits 0 and 31:4 of ECX are reserved.

- CPUID.10H.01H:ECX[3]: If 1, indicates non-contiguous capacity bitmask is supported. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous. Bits 0 and 31:4 of ECX are reserved.
- CPUID.10H.01H:EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported CLOS). Bits 31:16 are reserved.

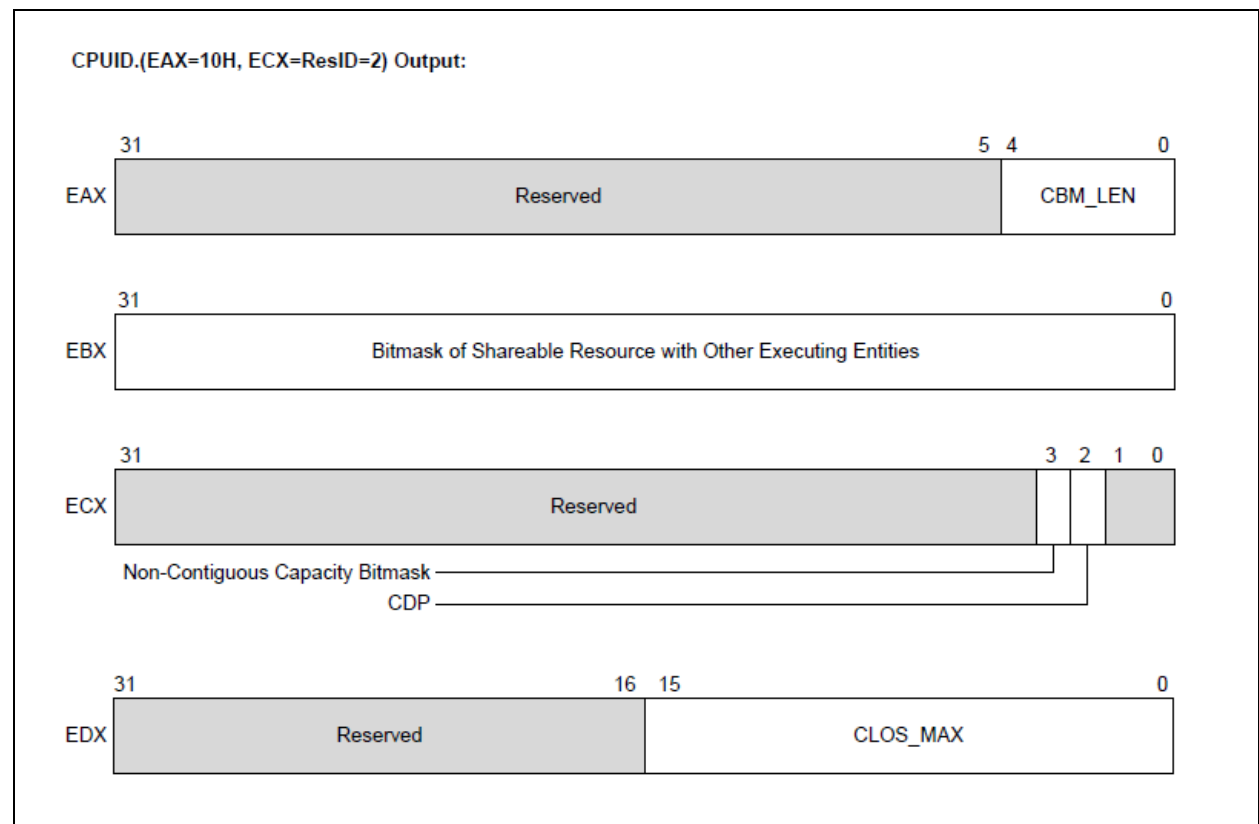


Figure 20-33. L2 Cache Allocation Technology

- CAT capability for L2 is enumerated by CPUID.10H.02H, see Figure 20-33. The specific CAT capabilities reported by CPUID.10H.02H are:
 - CPUID.10H.ResID=2:EAX[4:0] reports the length of the capacity bitmask. Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
 - CPUID.10H.02H:EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L2 allocation may be used by other entities in the platform. Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.
 - CPUID.10H.02H:ECX.CDP[2]: If 1, indicates L2 Code and Data Prioritization Technology is supported (see Section 17.19.6). Bits 1:0 and 31:4 of ECX are reserved.
 - CPUID.10H.02H:ECX[3]: If 1, indicates non-contiguous capacity bitmask is supported. The bits which are set in the various IA32_L2_MASK_n registers do not have to be contiguous. Bits 1:0 and 31:4 of ECX are reserved.

- CPUID.10H.02H:EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported CLOS). Bits 31:16 are reserved.

A note on migration of Classes of Service (CLOS): Software should minimize migrations of CLOS across logical processors (across threads or cores), as a reduction in the performance of the Cache Allocation Technology feature may result if CLOS are migrated frequently. This is aligned with the industry-standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and CLOS migration across processor logical threads and processor cores.

20.19.4.3 Cache Allocation Technology: Cache Mask Configuration

After determining the length of the capacity bitmasks (CBM) and number of CLOS supported using CPUID (see Section 20.19.4.2), each CLOS needs to be programmed with a CBM to dictate its available cache via a write to the corresponding IA32_resourceType_MASK_n register, where 'n' corresponds to a number within the supported range of CLOS, i.e., the range between 0 and CPUID.10H.ResID:EDX[15:0], inclusive, and 'resourceType' corresponds to a specific resource as enumerated by the set bits of CPUID.10H.00H:EBX[31:1], for instance, 'L2' or 'L3' cache.

A hierarchy of MSRs is reserved for Cache Allocation Technology registers of the form IA32_resourceType_MASK_n:

- From 0C90H through 0D8FH (inclusive), providing support for multiple sub-ranges to support varying resource types. The first supported resource type is 'L3', corresponding to the L3 cache in a platform. The MSRs range from 0C90H through 0D0FH (inclusive), enables support for up to 128 L3 CAT Classes of Service.

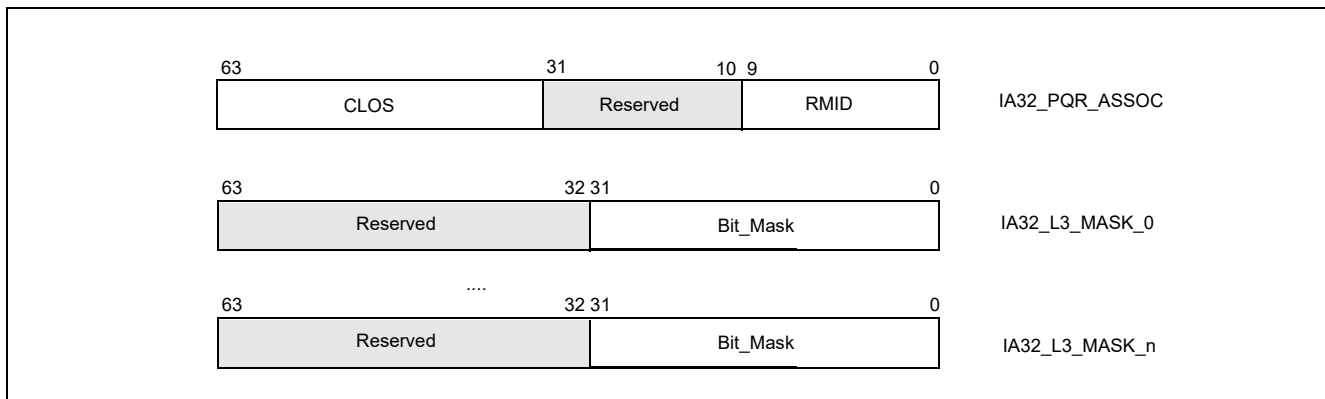


Figure 20-34. IA32_PQR_ASSOC, IA32_L3_MASK_n MSRs

- Within the same CAT range hierarchy, another set of registers is defined for resourceType 'L2', corresponding to the L2 cache in a platform, and MSRs IA32_L2_MASK_n are defined for n=[0,63] at addresses 0D10H through 0D4FH (inclusive).

Figure 20-34 and Figure 20-35 provide an overview of the relevant registers.

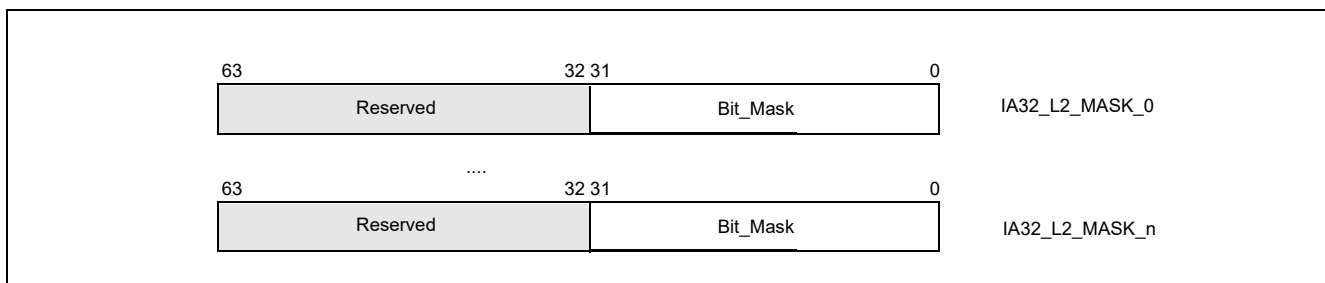


Figure 20-35. IA32_L2_MASK_n MSRs

All CAT configuration registers can be accessed using the standard RDMSR / WRMSR instructions.

Note that once L3 or L2 CAT masks are configured, threads can be grouped into Classes of Service (CLOS) using the IA32_PQR_ASSOC MSR as described in Section 20.19.4.4, “Class of Service to Cache Mask Association: Common Across Allocation Features.”

20.19.4.4 Class of Service to Cache Mask Association: Common Across Allocation Features

After configuring the available classes of service with the preferred set of capacity bitmasks, the OS/VMM can set the IA32_PQR_ASSOC.CLOS of a logical processor to the class of service with the desired CBM when a thread context switch occurs. This allows the OS/VMM to indicate which class of service an executing thread/VM belongs within. Each logical processor contains an instance of the IA32_PQR_ASSOC register at MSR location 0C8FH, and Figure 20-34 shows the bit field layout for this register. Bits[M:32] contain the CLOS field for each logical processor, where M can be determined by the following pseudo code:

```
CLOS_MAX = 0;
flags = CPUID.10H.00H:EBX;
for (ResID = 1; ResID <= 31; ++ResID){
    if (flags & (1 << ResID)){
        ResID_MAX = CPUID.10H.ResID:EDX & 0xFFFF;
        CLOS_MAX = max(CLOS_MAX, ResID_MAX);
    }
}
CLOS_WIDTH = Ceil(Log2(CLOS_MAX + 1));
M = (32 + CLOS_WIDTH - 1);
```

On platforms enumerating Resource Priority (Section 20.21.2) in CPUID as the only supported RDT Allocation feature, software should consult ACPI as described in Section 20.21.2.1.2 to determine maximum supported priority levels, which can be used (after subtracting one) as “CLOS_MAX” to determine CLOS field width.

Note that placing the RMID field within the same PQR register enables both RMID and CLOS to be swapped at context swap time for simultaneous use of monitoring and allocation features with a single register write for efficiency.

When CDP is enabled, Specifying a CLOS value in IA32_PQR_ASSOC.CLOS greater than MAX_CLOS_CDP = (CPUID.10H.01H:EDX[15:0] >> 1) will cause undefined performance impact to code and data fetches. In all cases, code and data masks for L2 and L3 CDP should be programmed with at least one bit set.

Note that if the IA32_PQR_ASSOC.CLOS is never written then the CAT capability defaults to using CLOS 0, which in turn is set to the default mask in IA32_L3_MASK_0 - which is all “1”s (on reset). This essentially disables the enforcement feature by default or for legacy operating systems and software.

See Section 20.19.7, “Introduction to Memory Bandwidth Allocation,” for important CLOS programming considerations including maximum values when using CAT and CDP.

20.19.5 Code and Data Prioritization (CDP): Enumerating and Enabling L3 CDP Technology

L3 CDP is an extension of L3 CAT. The presence of the L3 CDP feature is enumerated via CPUID.10H.01H:ECX.CDP[2] (see Figure 20-32). Most of the CPUID.10H.01H sub-leaf data that applies to CAT also apply to CDP. However, CPUID.10H.01H:EDX.CAT_MAX_CLOS specifies the maximum CLOS applicable to CAT-only operation. For CDP operations, CLOS_MAX_CDP is equal to (CPUID.10H.01H:EDX.CAT_MAX_CLOS >> 1).

If CPUID.10H.01H:ECX.CDP[2] = 1, the processor supports CDP and provides a new MSR IA32_L3_QOS_CFG at address 0C81H. The layout of IA32_L3_QOS_CFG is shown in Figure 20-36. The bit field definition of IA32_L3_QOS_CFG are:

- Bit 0: L3 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.CLOS is CLOS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

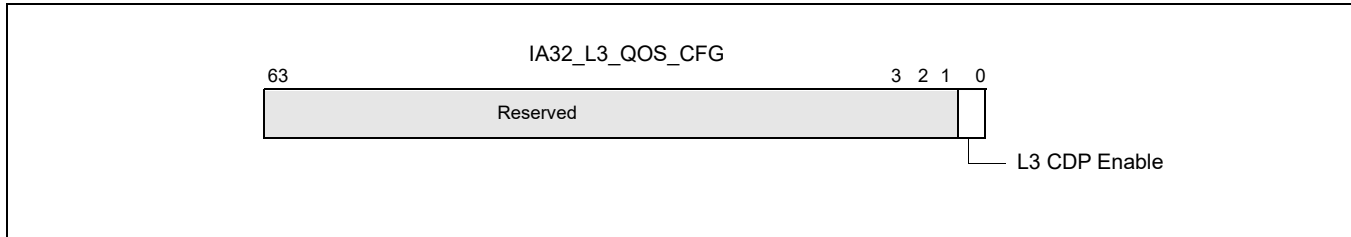


Figure 20-36. Layout of IA32_L3_QOS_CFG

IA32_L3_QOS_CFG default values are all 0s at RESET, the mask MSRs are all 1s. Hence, all logical processors are initialized in CLOS0 allocated with the entire L3 with CDP disabled, until software programs CAT and CDP. The scope of the IA32_L3_QOS_CFG MSR is defined to be the same scope as the L3 cache (e.g., typically per processor socket). Refer to Section 20.19.7 for software considerations while enabling or disabling L3 CDP.

20.19.5.1 Mapping Between L3 CDP Masks and CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per CLOS. The re-mapping is shown in Table 20-19.

Table 20-19. Re-indexing of CLOS Numbers and Mapping to CAT/CDP Mask MSRs

Mask MSR	CAT-only Operation	CDP Operation
IA32_L3_QOS_Mask_0	CLOS0	CLOS0.Data
IA32_L3_QOS_Mask_1	CLOS1	CLOS0.Code
IA32_L3_QOS_Mask_2	CLOS2	CLOS1.Data
IA32_L3_QOS_Mask_3	CLOS3	CLOS1.Code
IA32_L3_QOS_Mask_4	CLOS4	CLOS2.Data
IA32_L3_QOS_Mask_5	CLOS5	CLOS2.Code
....		
IA32_L3_QOS_Mask_‘2n’	CLOS‘2n’	CLOS‘n’.Data
IA32_L3_QOS_Mask_‘2n+1’	CLOS‘2n+1’	CLOS‘n’.Code

One can derive the MSR address for the data mask or code mask for a given CLOS number ‘n’ by:

- $\text{data\_mask\_address}(n) = \text{base} + (n \ll 1)$, where base is the address of IA32_L3_QOS_MASK_0.
- $\text{code\_mask\_address}(n) = \text{base} + (n \ll 1) + 1$.

When CDP is enabled, each CLOS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 20-29).

Mask MSR field definitions remain the same. Capacity masks must be formed of contiguous set bits, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003, and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

20.19.6 Code and Data Prioritization (CDP): Enumerating and Enabling L2 CDP Technology

L2 CDP is an extension of the L2 CAT feature. The presence of the L2 CDP feature is enumerated via CPUID.10H.02H:ECX.CDP[2] (see Figure 17-33). Most of the CPUID.10H.02H sub-leaf data that applies to CAT also

apply to CDP. However, CPUID.10H.02H:EDX.CAT_MAX_CLOS specifies the maximum CLOS applicable to CAT-only operation. For CDP operations, CLOS_MAX_CDP is equal to (CPUID.10H.02H:EDX.CAT_MAX_CLOS >> 1).

If CPUID.10H.02H:ECX.CDP[2] = 1, the processor supports L2 CDP and provides a new MSR IA32_L2_QOS_CFG at address 0C82H. The layout of IA32_L2_QOS_CFG is shown in Figure 20-37. The bit field definition of IA32_L2_QOS_CFG are:

- Bit 0: L2 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.CLOS is CLOS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

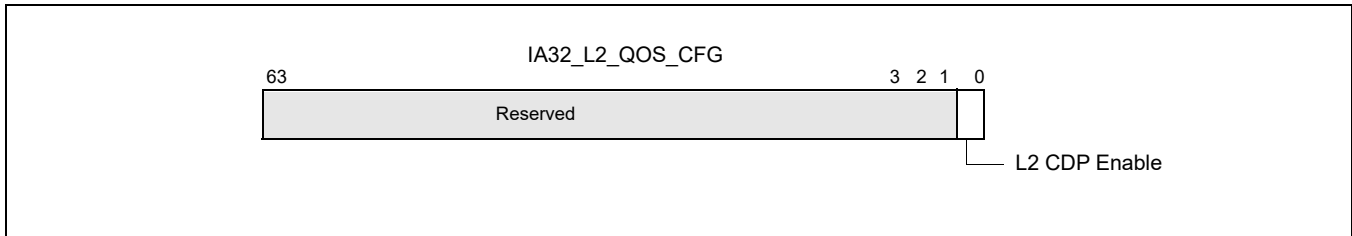


Figure 20-37. Layout of IA32_L2_QOS_CFG

IA32_L2_QOS_CFG default values are all 0s at RESET, and the mask MSRs are all 1s. Hence all logical processors are initialized in CLOS0 allocated with the entire L2 available and with CDP disabled, until software programs CAT and CDP. The IA32_L2_QOS_CFG MSR is defined at the same scope as the L2 cache, typically at the module level for Intel Atom processors for instance. In processors with multiple modules present it is recommended to program the IA32_L2_QOS_CFG MSR consistently across all modules for simplicity.

20.19.6.1 Mapping Between L2 CDP Masks and L2 CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per CLOS. This remapping is the same as the remapping shown in Table 20-19 for L3 CDP, but for the L2 MSR block (IA32_L2_QOS_MASK_n) instead of the L3 MSR block (IA32_L3_QOS_MASK_n). The same code / data mask mapping algorithm applies to remapping the MSR block between code and data masks.

As with L3 CDP, when L2 CDP is enabled, each CLOS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 20-29).

Mask MSR field definitions for L2 CDP remain the same as for L2 CAT. Capacity masks must be formed of contiguous set bits, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003, and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

20.19.6.2 Common L2 and L3 CDP Programming Considerations

Before enabling or disabling L2 or L3 CDP, software should write all 1's to all of the corresponding CAT/CDP masks to ensure proper behavior (e.g., the IA32_L3_QOS_Mask_n set of MSRs for the L3 CAT feature). When enabling CDP, software should also ensure that only CLOS number which are valid in CDP operation is used, otherwise undefined behavior may result. For instance in a case with 16 CAT CLOS, since CLOS are reduced by half when CDP is enabled, software should ensure that only CLOS 0-7 are in use before enabling CDP (along with writing 1's to all mask bits before enabling or disabling CDP).

Software should also account for the fact that mask interpretations change when CDP is enabled or disabled, meaning for instance that a CAT mask for a given CLOS may become a code mask for a different Class of Service when CDP is enabled. In order to simplify this behavior and prevent unintended remapping software should consider resetting all threads to CLOS[0] before enabling or disabling CDP.

20.19.6.3 Cache Allocation Technology Dynamic Configuration

All Intel Resource Director Technology (Intel RDT) interfaces including the IA32_PQR_ASSOC MSR, CAT/CDP masks, MBA delay values, and CQM/MBM registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,
- Accessing a QOS mask register outside the supported CLOS (the max CLOS number is specified in CPUID.10H.ResID:EDX[15:0]), or
- Writing a CLOS greater than the supported maximum (specified as the maximum value of CPUID.10H.ResID:EDX[15:0] for all valid ResID values) is written to the IA32_PQR_ASSOC.CLOS field.

When CDP is enabled, specifying a CLOS value in IA32_PQR_ASSOC.CLOS outside of the lower half of the CLOS space will cause undefined performance impact to code and data fetches due to MSR space re-indexing into code/data masks when CDP is enabled.

When reading the IA32_PQR_ASSOC register the currently programmed CLOS on the core will be returned.

When reading an IA32_resourceType_MASK_n register the current capacity bit mask for CLOS 'n' will be returned.

As noted previously, software should minimize migrations of CLOS across logical processors (across threads or cores), as a reduction in the accuracy of the Cache Allocation feature may result if CLOS are migrated frequently. This is aligned with the industry standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and CLOS migration across processor logical threads and processor cores.

20.19.6.4 Cache Allocation Technology Operation With Power Saving Features

Note that the Cache Allocation Technology feature cannot be used to enforce cache coherency, and that some advanced power management features such as C-states which may shrink or power off various caches within the system may interfere with CAT hints - in such cases the CAT bitmasks are ignored and the other features take precedence. If the highest possible level of CAT differentiation or determinism is required, disable any power-saving features which shrink the caches or power off caches. The details of the power management interfaces are typically implementation-specific, but can be found at Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

If software requires differentiation between threads but not absolute determinism then in many cases it is possible to leave power-saving cache shrink features enabled, which can provide substantial power savings and increase battery life in mobile platforms. In such cases when the caches are powered off (e.g., package C-states) the entire cache of a portion thereof may be powered off. Upon resuming an active state any new incoming data to the cache will be filled subject to the cache capacity bitmasks. Any data in the cache prior to the cache shrink or power off may have been flushed to memory during the process of entering the idle state, however, and is not guaranteed to remain in the cache. If differentiation between threads is the goal of system software then this model allows substantial power savings while continuing to deliver performance differentiation. If system software needs optimal determinism then power saving modes which flush portions of the caches and power them off should be disabled.

NOTE

IA32_PQR_ASSOC is saved and restored across C6 entry/exit. Similarly, the mask register contents are saved across package C-state entry/exit and are not lost.

20.19.6.5 Cache Allocation Technology Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and mask registers are unmodified across an SMI delivery. Thus, the execution of SMM handler code can interact with the Cache Allocation Technology resource and manifest some degree of non-determinism to the non-SMM software stack. An SMM handler may also perform certain system-level or power management practices that affect CAT operation.

It is possible for an SMM handler to minimize the impact on data determinism in the cache by reserving a CLOS with a dedicated partition in the cache. Such an SMM handler can switch to the dedicated CLOS immediately upon entering SMM, and switching back to the previously running CLOS upon exit.

20.19.6.6 Associating Threads with CAT/CDP Classes of Service

Threads are associated with Classes of Service (CLOS) via the per-logical-processor IA32_PQR_ASSOC MSR. The same CLOS concept applies to both CAT and CDP (for instance, CLOS[5] means the same thing whether CAT or CDP is in use, and the CLOS has associated resource usage constraint attributes including cache capacity masks). The mapping of CLOS to mask MSRs does change when CDP is enabled, according to the following guidelines:

- In CAT-only Mode - one set of bitmasks in one mask MSR control both code and data.
 - Each CLOS number map 1:1 with a capacity mask on the applicable resource (e.g., L3 cache).
- When CDP is enabled:
 - Two mask sets exist for each CLOS number, one for code, one for data.
 - Masks for code/data are interleaved in the MSR address space (see Table 20-19).

20.19.7 Introduction to Memory Bandwidth Allocation

The Memory Bandwidth Allocation (MBA) feature provides indirect and approximate control over memory bandwidth available per-core. It was introduced in the Intel Xeon Scalable Processor Family. This feature provides a method to control applications that may be over-utilizing bandwidth relative to their priority in environments such as the data-center.

The MBA feature uses existing constructs from the Intel RDT feature set, including Classes of Service (CLOS). A given CLOS used for L3 CAT, for instance, means the same thing as a CLOS used for MBA. Infrastructure, such as the MSR used to associate a thread with a CLOS (the IA32_PQR_ASSOC MSR) and some elements of the CPUID enumeration (such as CPUID.10H), are shared. Certain generations include advanced hardware controllers for efficiency. For more information, refer to the “Intel® Resource Director Technology Architecture Specification.”

The following sections describe CPU interfaces to Memory Bandwidth Allocation, such as CPUID enumeration and configuration interfaces (MSRs).

20.19.7.1 Memory Bandwidth Allocation Enumeration

Similar to other Intel RDT features, enumeration of the presence and details of the MBA feature is provided via a sub-leaf of the CPUID instruction.

Key components of the enumeration are as follows.

- Support for the MBA feature on the processor, and if MBA is supported, the following details:
 - Number of supported Classes of Service (CLOS) for the processor.
 - The maximum MBA delay value supported (which also implicitly provides a definition of the granularity).
 - An indication of whether the delay values which can be programmed are linearly spaced or not.

The presence of any of the Intel RDT features which enable control over shared platform resources is enumerated by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_A[15] reports 1, the processor supports software control over shared processor resources. Software may then use CPUID.10H to enumerate additional details on the specific controls provided.

Through CPUID.10H software may determine whether MBA is supported on the platform. Specifically, as shown in Figure 20-31, bit 3 of the EBX register indicates whether MBA is supported on the processor, and the bit position (3) constitutes a Resource ID (ResID) which allows enumeration of MBA details. For instance, if bit 3 is supported this implies the presence of CPUID.10H.[ResID=3] as shown in Figure 20-38 which provides the following details.

- CPUID.10H.ResID=3:EAX[11:0] reports the maximum MBA throttling value supported, minus one. For instance, a value of 89 indicates that a maximum throttling value of 90 is supported. Additionally, in cases where a linear interface (see below) is supported then one hundred minus the maximum throttling value indicates the granularity, 10% in this example.

- CPUID.10H.ResID=3:EBX is reserved.
- CPUID.10H.ResID=3:ECX[0] indicates that the MBA throttling values are applied per-logical-processor independently if set.
- CPUID.10H.ResID=3:ECX[2] reports whether the response of the delay values is linear (see text).
- CPUID.10H.ResID=3:EDX[15:0] reports the number of Classes of Service (CLOS) supported for the feature (minus one). For instance, a reported value of 15 implies a maximum of 16 supported MBA CLOS.

The number of CLOS supported for the MBA feature may or may not align with other resources such as L3 CAT. In cases where the Intel RDT features support different numbers of CLOS the lowest numerical CLOS support the common set of features, while higher CLOS may support a subset. For instance, if L3 CAT supports 8 CLOS while MBA supports 4 CLOS, all 8 CLOS would have L3 CAT masks available for cache control, but the upper 4 CLOS would not offer MBA support. In this case the upper 4 CLOS would not be subject to any throttling control. Software can manage supported resources / CLOS in order to either have consistent capabilities across CLOS by using the common subset or enable more flexibility by selectively applying resource control where needed based on careful CLOS and thread mapping. In all cases, CLOS[0] supports all Intel RDT resource control features present on the platform.

Discussion on the interpretation and usage of the MBA delay values is provided in Section 20.19.7.2 on MBA configuration.

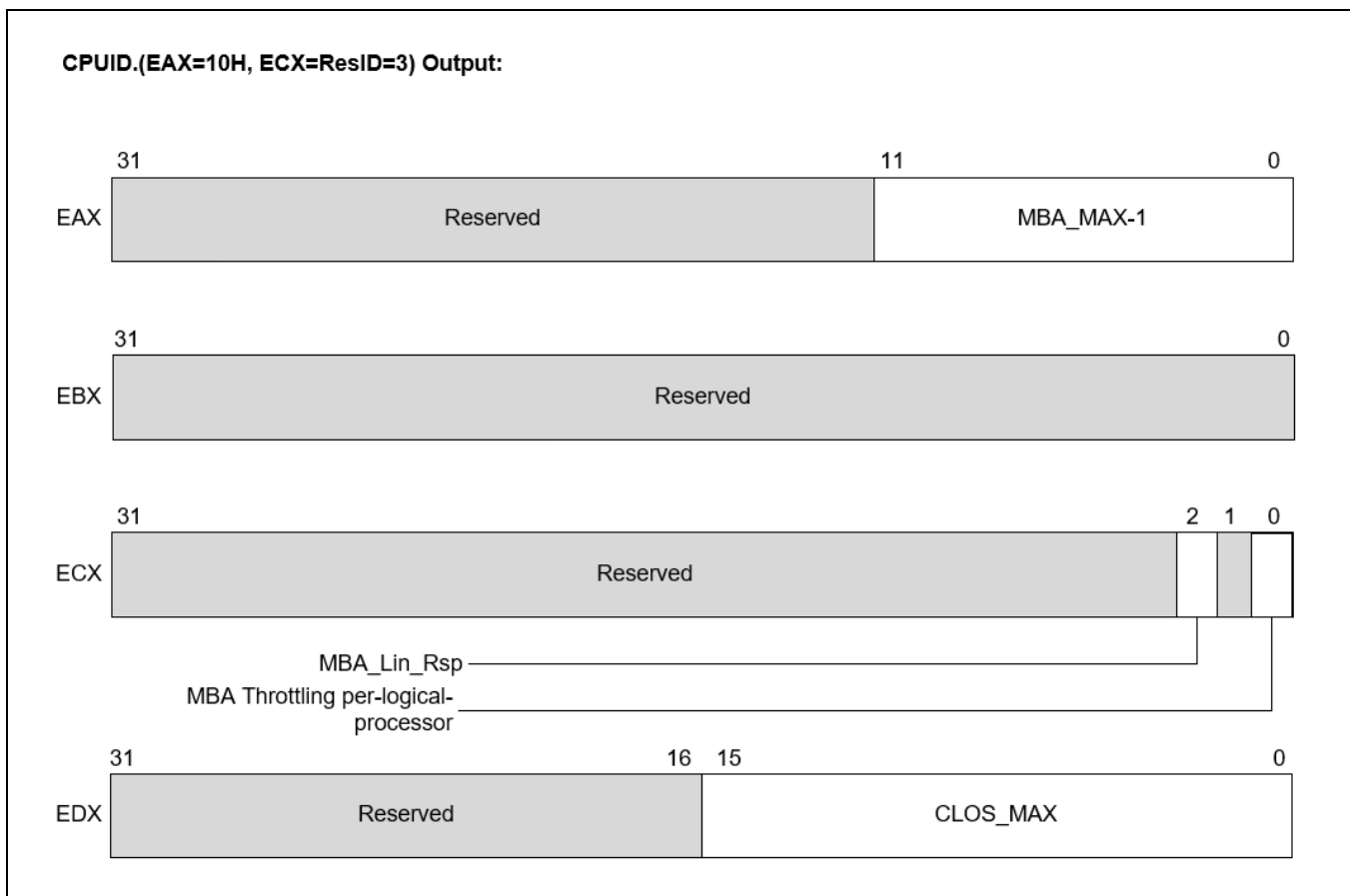


Figure 20-38. CPUID.10H.03H MBA Feature Details Identification

20.19.7.2 Memory Bandwidth Allocation Configuration

The configuration of MBA takes consists of two processes once enumeration is complete.

- Association of threads to Classes of Service (CLOS) - accomplished in a common fashion across Intel RDT features as described in Section 20.19.7.1 via the `IA32_PQR_ASSOC` MSR. As with features such as L3 CAT,

- software may update the CLOS field of the PQR MSR at context swap time in order to maintain the proper association of software threads to Classes of Service on the hardware. While logical processors may each be associated with independent CLOS, see Section 20.19.7.3 for important usage model considerations (initial versions of the MBA feature select the maximum delay value across threads).
- Configuration of the per-CLOS delay values, accomplished via the IA32_L2_QoS_Ext_BW_Thrtl_n MSR set shown in Table 20-20.

The MBA delay values which may be programmed range from zero (implying zero delay, and full bandwidth available) to the maximum (MBA_MAX) specified in CPUID as discussed in Section 20.19.7.1. The throttling values are approximate and do not sum to 100% across CLOS, rather they should be viewed as a maximum bandwidth “cap” per-CLOS.

Software may select an MBA delay value then write the value into one or more of the IA32_L2_QoS_Ext_BW_Thrtl_n MSRs to update the delay values applied for a specific CLOS. As shown in Table 20-20 the base address of the MSRs is at D50H, and the range corresponds to the maximum supported CLOS from CPUID.10H.ResID=1:EDX[15:0] as described in Section 20.19.7.1. For instance, if 16 CLOS are supported then the valid MSR range will extend from D50H through D5F inclusive.

Table 20-20. MBA Delay Value MSRs

Delay Value MSR	Address
IA32_L2_QoS_Ext_BW_Thrtl_0	D50H
IA32_L2_QoS_Ext_BW_Thrtl_1	D51H
IA32_L2_QoS_Ext_BW_Thrtl_2	D52H
....	
IA32_L2_QoS_Ext_BW_Thrtl_'CLOS_MAX'	D50H + CLOS_MAX from CPUID.10H.03H

- The definition for the MBA delay value MSRs is provided in Figure 17.39. The lower 16 bits are used for MBA delay values, and values from zero to the maximum from the CPUID MBA_MAX-1 value are supported. Values outside this range will generate #GP(0).
- If linear input throttling values are indicated by CPUID.10H.ResID=3:ECX[2] then values from zero through the MBA_MAX field from CPUID.10H.ResID=3:EAX[11:0] are supported as inputs. In the linear mode the input precision is defined as 100-(MBA_MAX). For instance, if the MBA_MAX value is 90, the input precision is 10%. Values not an even multiple of the precision (e.g., 12%) will be rounded down (e.g., to 10% delay applied).
- If linear values are not supported (CPUID.10H.ResID=3:ECX[2] = 0) then input delay values are powers-of-two from zero to the MBA_MAX value from CPUID. In this case any values not a power of two will be rounded down the next nearest power of two.

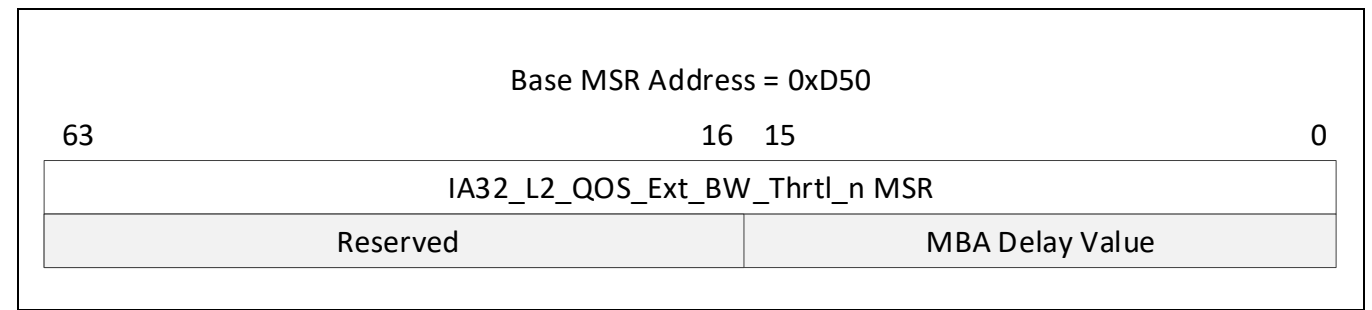


Figure 20-39. IA32_L2_QoS_Ext_BW_Thrtl_n MSR Definition

Note that the throttling values provided to software are calibrated through specific traffic patterns, however as workload characteristics may vary the response precision and linearity of the delay values will vary across products and should be treated as approximate values only.

20.19.7.3 Memory Bandwidth Allocation Usage Considerations

Different versions of Memory Bandwidth Allocation have various usage considerations and improving efficiency over time. See the “Intel® Resource Director Technology Architecture Specification” for additional details.

20.19.8 Cache Bandwidth Allocation (CBA)

20.19.8.1 Introduction to Cache Bandwidth Allocation

The Cache Bandwidth Allocation (CBA) feature provides control over bandwidth available between Level 1 (L1) caches, Level 2 (L2) Caches, and Level 3 (L3) Caches (as applicable) for each of the logical processors. Since reducing upstream bandwidth can also reduce bandwidth to external memory, this also provides an indirect control of memory bandwidth. The CBA feature, along with the MBA, provides a mechanism to control the bandwidth of different applications.

A given CLOS used for L3 CAT, for instance, means the same thing as a CLOS used for CBA. Infrastructure such as the MSR used to associate a logical processor with a CLOS (the IA32_PQR_ASSOC_MSR) and some elements of the CPUID enumeration (such as CPUID leaf 10H (Cache Allocation Technology Enumeration Leaf)) are shared. For more information, refer to the “Intel® Resource Director Technology Architecture Specification.”

The following sections describe the CPUID enumeration and configuration interfaces (Model-Specific Registers) applicable to the Cache Bandwidth Allocation feature.

20.19.8.2 Cache Bandwidth Allocation Enumeration

As with certain other Intel RDT features, enumeration of the presence and details of the CBA feature is provided via a sub-leaf of the CPUID instruction.

Key components of the enumeration include support for the CBA feature on the processor, and if CBA is supported, the following details:

- Number of supported Classes of Service for the processor.
- Scope of the CBA feature MSRs.
- The maximum CBA throttle Level supported.
- An indication of whether the throttle values that can be programmed are linearly spaced or not.

The presence of any of the Intel RDT features that enable control over shared platform resources is enumerated by executing CPUID instruction with EAX = 07H and ECX = 0H as input. If CPUID.07H.00H: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.

Using CPUID leaf 10H, software may determine whether CBA is supported on the platform. Specifically, as shown in Figure 20-31, bit 5 of the EBX register indicates whether CBA is supported on the processor, and the bit position (5) constitutes a Resource ID (ResID), which allows the enumeration of CBA details. For instance, if bit 5 is supported, this implies the presence of CPUID.10H.05H as shown in Figure 20-40. “CPUID.10H.05H, CBA Feature Details Identification,” which provides the following details (this information can also be found in Chapter 21 of Volume 1):

- CPUID.10H.05H:EAX
 - EAX[7:0] reports the maximum CBA throttling value supported.
 - EAX[11:8] reports the scope of CBA IA32_QoS_Core_BW_Thrtl_n MSRs. If EAX[11:8]=1, this indicates the logical processor scope of the MSRs.
 - EAX[31:12] is reserved.
- CPUID.10H.05H:EBX
 - EBX[31:0] is reserved.
- CPUID.10H.05H:ECX

- ECX[3] reports whether the response of the bandwidth control is approximately linear. If ECX[3] is 1, the response of the bandwidth control is approximately linear. If ECX[3] is 0, the response of the bandwidth control is non-linear.
- ECX[2:0] and ECX[31:4] are reserved.
- CPUID.10H.05H:EDX
 - EDX[15:0] reports the number of Classes of Service supported for the feature. Add one to the return value to get the result. For instance, a reported value of 15 implies a maximum of 16 supported CBA CLOS.
 - EDX[31:16] is reserved.

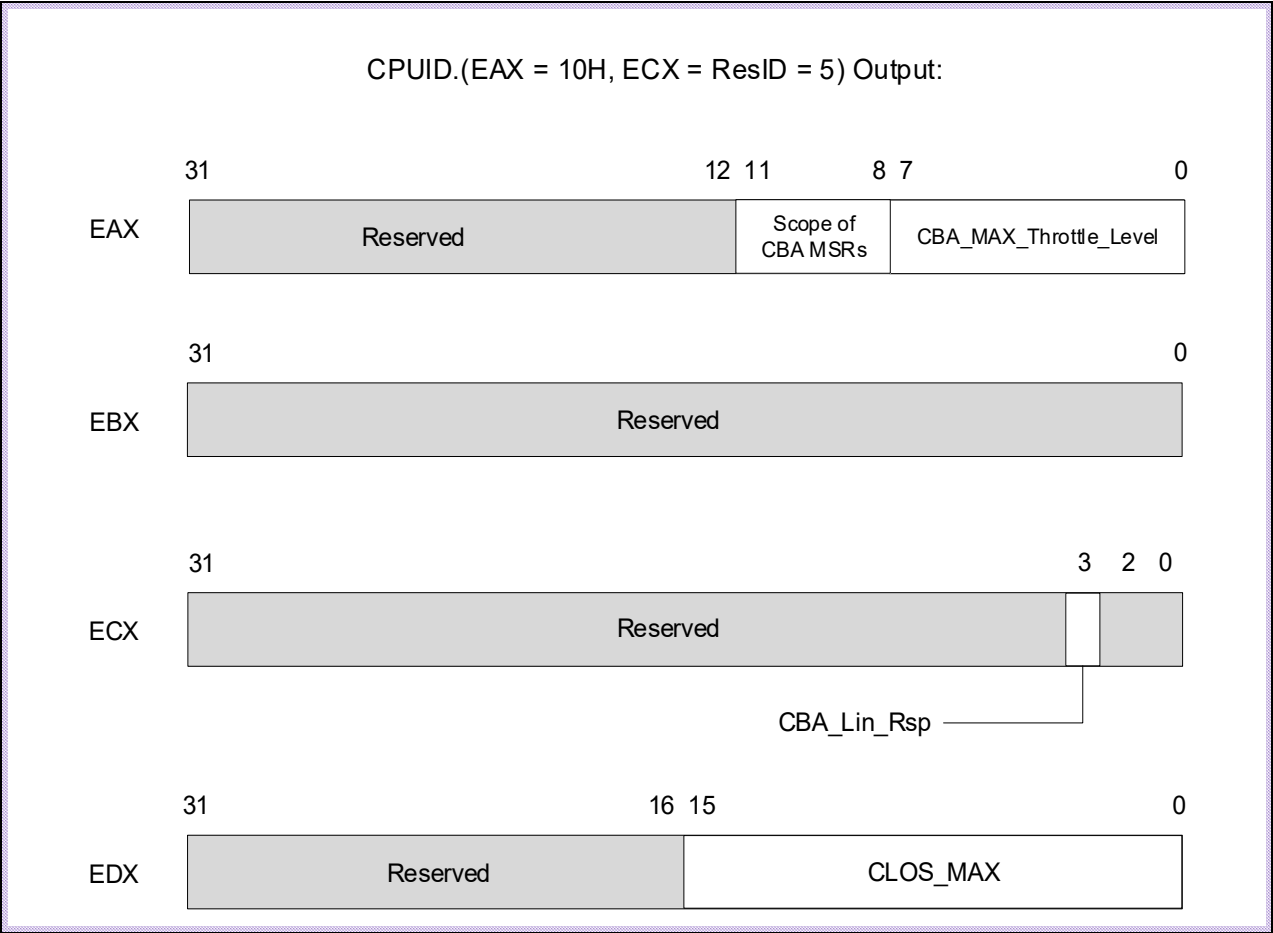


Figure 20-40. CPUID.10H.05H, CBA Feature Details Identification

20.19.8.3 Cache Bandwidth Allocation Configuration

The configuration of CBA consists of two processes once enumeration is complete:

- The association of logical processors to Classes of Service (CLOS) is accomplished commonly across Intel RDT features through the IA32_PQR_ASSOC MSR. Software may update the CLOS field of the PQR MSR dynamically, including at context swap time, to maintain the proper association of logical processors to Classes of Service on the hardware.
- A new set of architectural MSRs is added to enable software to communicate the memory bandwidth QoS requirements of the application running on the logical processor. The scope of these MSRs, IA32_QoS_Core_BW_Thrtl_n, is per logical processor. Each MSR encodes packed 8-bit fields indexed by Class of Service, which specify the Throttle Level for each CLOS.

The CLOS field value of the IA32_PQR_ASSOC MSR is used to index into the MSRs and select the software-specified Throttle Level. Each logical processor uses this level to control the bandwidth across the cache hierarchy. The hardware ensures coordination with the MBA feature (where present). The reset value of the CLOS[i].Level=0 indicates unthrottled bandwidth. This field may be programmed from 0 to CBA_MAX_Throttle_Level (see Figure 20-40). Any values outside this range will generate a #GP(0). A higher value of CLOS[i].Level implies a higher level of bandwidth throttling, and a lower number indicates lesser throttling. The number of supported CLOS for a given logical processor is enumerated in CPUID.10H.05H:EDX. In an example where a logical processor supports 16 CLOS, two 64-bit MSRs with packed Throttling Levels (TLs) are defined, IA32_QoS_Core_BW_Thrtl_0 (defining packed TLs for CLOS[7:0]) and IA32_QoS_Core_BW_Thrtl_1 (defining TLs for CLOS[15:8]). For example, within the MSR IA32_QoS_Core_BW_Thrtl_0, bits [7:0] define the TL field for CLOS 0 (see Figure 20-41).

Advanced versions of the MBA feature may manage the external memory bandwidth associated with the CLOS by dynamically increasing or decreasing the bandwidth, under software guidance, to maintain throttling priorities while maximizing system performance as described in the Intel Software Developer's Manual and the Intel RDT Architecture Specification. The CBA feature, along with the MBA, provides a mechanism to control the bandwidth of different applications. Software should understand that the effective throttling of an application may be whichever of the CBA or MBA specifies more throttling. Software may use CBA, MBA, or a combination to achieve bandwidth and performance management goals if supported on a processor.

Table 20-21. Cache Bandwidth Allocation (CBA) MSRs

Delay Value MSR	Address
IA32_QoS_Core_BW_Thrtl_0	E00H
IA32_QoS_Core_BW_Thrtl_1	E01H
IA32_QoS_Core_BW_Thrtl_2	E02H
...	...
IA32_QoS_Core_BW_Thrtl_'(((COS_MAX+1)/8) - 1)'	E00H + (((COS_MAX from CPUID.10H.5 + 1)/8 - 1)

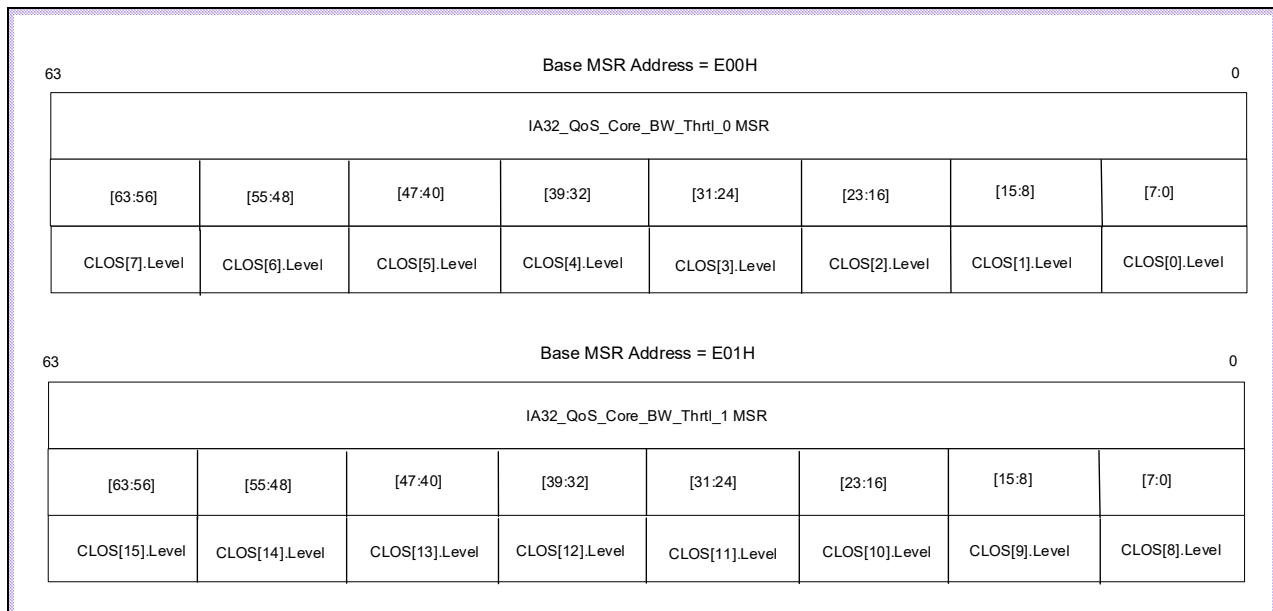


Figure 20-41. IA32_QoS_Core_BW_Thrtl_n MSR Definition

Note that the throttling values provided to the software are calibrated through specific traffic patterns; however, as workload characteristics may vary, the response precision and linearity of the bandwidth threshold values will vary across products and should be treated as approximate values only.

20.19.8.4 Cache Bandwidth Allocation Usage Considerations

Cache Bandwidth Allocation has various software usage considerations and improves efficiency over product generations. See the “Intel® Resource Director Technology Architecture Specification” for additional details.

20.20 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) FOR NON-CPU AGENTS

This section describes Intel RDT features for non-CPU agents. CPU agents are threads running on IA cores. Non-CPU agents include PCIe and CXL devices and integrated accelerators, thus broadly encompassing the set of agents that read from and write to either caches or memory, excluding IA cores. The non-CPU agent Intel RDT features enable monitoring of I/O device shared cache and memory bandwidth and cache allocation control. This provides features for I/O devices equivalent to the CPU agent Intel RDT capabilities CMT, MBM, and CAT (discussed in Section 20.18 and Section 20.19). Refer to the “Intel® Resource Director Technology Architecture Specification” regarding design goals, use cases, software architecture, ACPI enumeration, and MMIO register interfaces.

“Non-CPU agent Intel RDT” refers to capabilities that monitor and control non-CPU agents’ resource utilization, including PCIe and CXL devices and integrated accelerators. Non-CPU agent Intel RDT may be called I/O RDT in some literature. In this document, the term “non-CPU agent Intel RDT” is used.

20.20.1 Non-CPU Agent Intel® RDT Features Enumeration Details

CPU agent Intel RDT features use the CPUID instruction to enumerate supported features and the level of support. Architectural Model-Specific Registers (MSRs) are interfaces to the monitoring and allocation features, as described in Sections 18.18 and 18.19.

Non-CPU agent Intel RDT builds on CPU agent Intel RDT by extending CPUID to indicate the presence and integration of non-CPU agent Intel RDT and by providing rich enumeration information in vendor-specific extensions to the Advanced Configuration and Power Interface (ACPI), in particular in the I/O RDT (IRDT) table. The ACPI extensions detailed in the “Intel® Resource Director Technology Architecture Specification” provide mechanisms to comprehend the structure of devices attached behind I/O blocks to particular links and what forms of tagging are supported on a per-link basis.

It is recommended that software parse CPUID and ACPI to obtain a detailed understanding of platform support and capabilities before attempting to use non-CPU agent Intel RDT.

20.20.1.1 CPUID-Based Enumeration for Non-CPU Agent Intel® RDT Feature

CPUID-based enumeration provides a method by which all architectural Intel RDT features may be enumerated.

For CPU agent Intel RDT, monitoring details are enumerated in a CPUID sub-leaf denoted as CPUID.0FH.ResID, where ResID corresponds to a resource ID bit index from the CPUID.0FH.00H sub-leaf. Similarly, Intel RDT allocation features are described in CPUID.10H.ResID. (Note that the ResID bit positions are not guaranteed to be symmetric or have the same encodings.)

No CPUID leaves or sub-leaves are created for non-CPU agent Intel RDT. Rather, non-CPU agent Intel RDT extends the existing Intel RDT CPUID sub-leaves with a bit per resource type, indicating whether non-CPU agent Intel RDT monitoring or control is present. CPUID.0FH.ResID=1:EAX[10:9] represents the presence of CMT and MBM features for non-CPU agents. CPUID.10H.ResID=1:ECX[1] represents the presence of the CAT feature for non-CPU agents.

Specifically, for non-CPU Agent Intel RDT Monitoring (see Figure 20-21):

- Bits are added in the CPU Agent Intel RDT CMT/MBM leaf: CPUID.0FH.ResID=1:EAX[10:9].
 - EAX[bit 9]: If set, indicates the presence of non-CPU Agent Cache Occupancy Monitoring (the equivalent of CPU Agent Intel RDT’s CMT feature).

- EAX[bit 10]: If set, indicates the presence of non-CPU Agent memory L3 external BW monitoring (the equivalent of CPU Agent Intel RDT's MBM feature).

For non-CPU Agent Intel RDT Allocation (see Figure 20-32):

- New bit in L3 CAT leaf: CPUID.10H.ResID=1:ECX[1].
 - ECX[bit 1]: If set, indicates the presence of non-CPU Agent Cache Allocation Technology (the equivalent of CPU Agent Intel RDT's L3 CAT feature).
- As before, ECX[bit 2] indicates that L3 CDP is supported if set.

Note that no equivalent bits are defined in CPUID.10H.ResID=2 as there is no ability for devices to fill into core L2 caches.

If any of these non-CPU agent Intel RDT enumeration bits are set, indicating that a monitoring feature or allocation feature is present, it also indicates the presence of the IA32_L3_IO_RDT_CFG architectural MSR. This MSR may be used to enable the non-CPU agent Intel RDT features. See Section 20.20.2 for MSR details.

The presence of Intel RDT is a prerequisite for using the equivalent non-CPU agent Intel RDT feature. If a particular CPU agent Intel RDT feature is absent, any attempt to use non-CPU agent Intel RDT equivalents will result in general protection faults in the MSR interface. Attempts to enable unsupported features in the I/O complex will result in writes to the corresponding MMIO enable or configuration interfaces being ignored.

Software may use the existing CPUID leaves to gather the maximum number of RMID and CLOS tags for each resource level (e.g., L3 cache), and non-CPU agent Intel RDT is also subject to these limits.

Some platforms may support a mix of features, for instance, supporting L3 CAT architectural controls and the non-CPU agent Intel RDT equivalent, but no CMT/MBM monitoring or non-CPU agent monitoring equivalent, and these capabilities should be enumerated on a per-platform basis.

20.20.1.2 ACPI Enumeration

When support for non-CPU agent Intel RDT features is detected using CPUID, ACPI may be consulted for further details on the level of feature support, device structures behind various I/O ports, and the specific MMIO interfaces used to control a given device.

Non-CPU agent Intel RDT enumeration is via the "IRDT" ACPI table. For more information, refer to the "Intel® Resource Director Technology Architecture Specification."

20.20.2 Non-CPU Agent Intel® RDT Feature Enable MSR

Before configuring non-CPU agent Intel RDT through MMIO, the feature should be enabled using the non-CPU agent Intel RDT Feature Enable MSR, IA32_L3_IO_RDT_CFG (MSR address 0C83H). As described in Section 20.20.1.1, the presence of one or more CPUID bits indicating support for one or more non-CPU agent Intel RDT features also indicates the presence of this MSR. This MSR may be used to enable the non-CPU agent Intel RDT features.

Two bits are defined in this MSR. Bit 0, when set, enables non-CPU agent RDT resource allocation features. Bit 1, when set, enables non-CPU agent Intel RDT monitoring features.

The L3 Non-CPU agent Intel RDT Monitoring Enable bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource monitoring features are present.

The L3 Non-CPU agent Intel RDT Allocation Enable bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource allocation features are present.

The default value is 0x0, so both classes of features are disabled by default. All bits not defined are reserved. Writing a non-zero value to any reserved bit will generate a General Protection Fault (#GP(0)).

This MSR is scoped at the L3 cache level and is cleared on system reset. It is expected that the software will configure this MSR consistently across all L3 caches that may be present on that package.

The definition of the IA32_L3_IO_RDT_CFG MSR is shown in Figure 20-42.

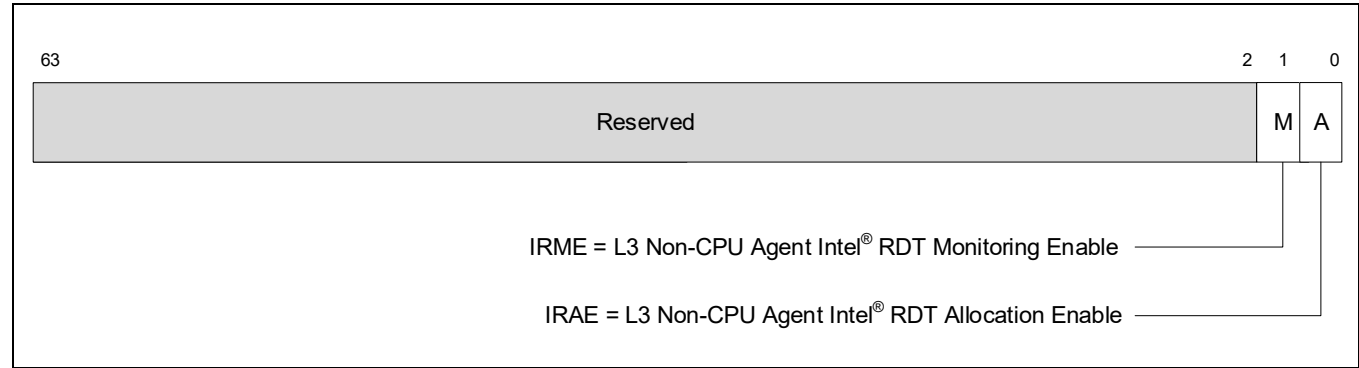


Figure 20-42. Layout of the IA32_L3_IO_RDT_CFG MSR for Enabling Non-CPU Agent Intel® RDT

20.20.3 Asymmetrical Enumeration of Intel® RDT Features

The enumeration of RDT has been extended to platforms that enumerate asymmetrical resources. For example, hybrid platforms (supporting multiple processor types) may contain multiple last-level caches (LLCs) with different capabilities, or some processors may not include an LLC at the same enumeration level. Software should consider the following changes to RDT enumeration to enable full RDT capabilities across all platforms. Asymmetrical RDT CPUID leaves 27H and 28H may enumerate distinct capabilities between logical processors on the platform, while CPUID RDT leaves 0FH and 10H may enumerate the common set of capabilities to ensure software compatibility. Asymmetrical RDT CPUID leaves should be enumerated per logical processor to identify differences in capabilities.

The enumeration of RDT technologies first consists of the following four CPUID leaf 07H bits in Table 20-22 and the corresponding CPUID leaves in Table 20-23.

Table 20-22. Asymmetrical Enumeration

CPUID Enumeration	Field Name	Description
CPUID.07H.00H:EBX.RDT_M[12]	RDT_M	Specifies that the RDT-M symmetrical enumeration is available on all logical processors.
CPUID.07H.00H:EBX.RDT_A[15]	RDT_A	Specifies that the RDT-A symmetrical enumeration is available on all logical processors.
CPUID.07H.01H:ECX.RDT_M_ASYM[0]	Asymmetric RDT-M	Specifies that logical processors on this platform may support RDT-M; but is not guaranteed.
CPUID.07H.01H:ECX.RDT_A_ASYM[1]	Asymmetric RDT-A	Specifies that logical processors on this platform may support RDT-A; but is not guaranteed.

Table 20-23. Asymmetrical CPUID Leaves

CPUID Enumeration	CPUID Leaf	Description
CPUID.07H.00H:EBX.RDT_M[12]	Leaf 0FH	This leaf provides the same values regardless of the logical processor querying leaf 0FH.
CPUID.07H.00H:EBX.RDT_A[15]	Leaf 10H	This leaf provides the same values regardless of the logical processor querying leaf 10H.
CPUID.07H.01H:ECX.RDT_M_ASYM[0]	Leaf 27H	This leaf can return different values between logical processors querying leaf 27H.
CPUID.07H.01H:ECX.RDT_A_ASYM[1]	Leaf 28H	This leaf can return different values between logical processors querying leaf 28H.

The Asymmetrical RDT-M leaf 27H and subleaves have the same definitions as RDT-M leaf 0FH. The Asymmetrical RDT-A leaf 28H and subleaves have the same definitions as RDT-A leaf 10H. Symmetrical platforms may enumerate the existence of asymmetrical leaves, and therefore cannot be used to determine a system is asymmetrical.

The data provided in the asymmetrical leaves 27H and 28H are supersets of leaves 0FH and 10H. The enumeration and capabilities returned through these leaves can be any of the following:

- Enumerating either leaves 27H or 28H is not indicative that leaves 0FH or 10H would be supported.
- Leaves 0FH and/or 10H may return a smaller subset than the intersection of features across resources where either leaves 27H or 28H exist on a platform.
- Leaves 0FH and/or 10H may return the intersection of features across resources where either leaves 27H or 28H exist on a platform.
- Leaves 0FH and/or 10H may be enumerated but contain no information where either leaves 27H or 28H exist on a platform.

The enumeration provided in leaves 0FH and 10H do not have a requirement to provide resource feature information on a platform where leaves 27H or 28H exist.

Software should use the following algorithm to detect support of Intel® RDT Monitoring:

```
IF CPUID.(EAX=07H,ECX=01H):ECX.Asymmetrical RDT-M[Bit 0] = 1 THEN
    // Perform asymmetrical enumeration using leaf 27H
ELSE IF CPUID.(EAX=07H,ECX=01H):EBX.RDT-M[Bit 12] = 1 THEN
    // Perform symmetrical enumeration using leaf 0FH
ELSE
    // Intel® Resource Directory Technology Monitoring is not supported.
ENDIF
ENDIF
```

Software should use the algorithm below to detect support of Intel® RDT Allocation:

```
IF CPUID.(EAX=07H,ECX=01H):ECX.Asymmetrical RDT-A[Bit 1] = 1 THEN
    // Perform asymmetrical enumeration using leaf 28H
ELSE IF CPUID.(EAX=07H,ECX=01H):EBX.RDT-A[Bit 15] = 1 THEN
    // Perform symmetrical enumeration using leaf 10H
ELSE
    // Intel® Resource Directory Technology Allocation is not supported.
ENDIF
ENDIF
```

The asymmetrical leaves are required to be read on each logical processor. Software may determine the intersection of features across all logical processors, or software may group logical processors with matching feature sets. Logical processors that share a resource will report the same feature support for that resource. An example would be to enumerate each set of logical processors sharing an L3 cache, reading CPUID leaf 28H for L3 CAT on one logical processor sharing each L3 cache and applying those settings to all logical processors sharing the same L3 cache.

20.21 INTEL® RDT DERIVATIVE TECHNOLOGIES

This section describes Intel RDT derivative technologies. These technologies are based on RDT technologies and may provide simplified hardware/software interfaces. See following sections for multiple technologies which build atop RDT.

20.21.1 Core Telemetry for Intel® Platform Monitoring Technology (PMT)

Core telemetry is part of the Intel® Platform Monitoring Technology (Intel® PMT) feature set. Processors that support core telemetry collect per-core telemetry during core execution (including the execution context of the core — e.g., Resource Monitoring ID or RMID), aggregate the data internally, and report them to external agents, enabling workload-level tracking. This telemetry **may** be used by external agents to monitor processor behavior, to improve placement of workloads in data centers, increase resource utilization, monitor energy consumption, etc. The telemetry collected is processor-specific. For information about Intel® PMT refer to the Intel® Platform Monitoring Technology External Specification.

To avoid creating a side channel that might leak information from trusted execution environments (TEEs), core telemetry collected during TEE execution is discarded by the processor. Specifically, out-of-band core telemetry is discarded if it was collected while the core was running in any of the following situations:

- During execution of an enclave supported by Intel® Software Guard Extensions (Intel® SGX). This is true even for enclave debug following an opt-in entry.
- During execution of an authenticated code module.
- During execution in SEAM root operation.
- During execution in SEAM non-root operation, unless the “allow SEAM-guest telemetry” VM-entry control is 1.

For information about SEAM (secure arbitration mode), see Chapter 35.

20.21.2 Resource Priority

Resource priority (RP) is a priority-based architectural interface that allows for biasing resource allocation of shared resources to higher priority threads.

RP provides software with a simplified means to indicate priority to RP-capable hardware without the need to manage specific allocation configurations that may vary with each generation of hardware, such as how much L3 cache is relevant.

The Resource Priority control logic in hardware evaluates per-thread priority values that are targeting a shared resource and applies work conserving algorithm to ensure resource is optimally allocated if higher priority thread is not using the shared resource.

20.21.2.1 Resource Priority Enumeration Details

As an RDT derivative technology, Resource Priority requires an enumeration flow which combines information from CPUID and the Advanced Configuration and Power Interface (ACPI) specification. Before using RP, it is recommended to consult both this documentation and the RDT Architecture Specification, version 1.4 or newer.

Resource Priority uses the CPUID instruction to enumerate supported hardware feature details and the level of support. Architectural Model-Specific Registers (MSRs) are used as part of the enable interface, as described in Section 20.21.2.2.

Resource Priority also requires firmware support in the form of BIOS pre-configuration of the mapping of amounts of resources to priority levels, in order to simplify OS usage. Firmware support on a given platform may be confirmed through ACPI, as described in detail in the RDT Architecture Specification, version 1.4 or newer.

Resource Priority builds on Intel RDT’s architectural constructs by extending CPUID to indicate the presence and integration of RP and by providing rich enumeration information in vendor-specific extensions to the Advanced Configuration and Power Interface (ACPI), in particular in the Collaborative Processor Performance Control (CPPC) table, documented in the Advanced Configuration and Power Interface (ACPI) Specification, version 6.6 or newer.

Usage details including the required flow that software should follow to first enumerate, verify pre-configuration of resource associations, then use the RP feature are described in the RDT Architecture Specification, version 1.4 or newer.

20.21.2.1.1 CPUID-Based Enumeration

Resource Priority is an RDT allocation technology enumerated by CPUID. The technology is present when all of the following are true.

- If CPUID.07H.00H:EBX.RDT_A[15] is 1, the processor supports software control over shared processor resources.
- If CPUID.10H.00H:EBX[6] is 1, the processor supports RP baseline capabilities and enumerates them in CPUID.10H.06H.

The RP baseline capabilities and requirements are reported by CPUID.10H.06H as follows. (Note the MSR definitions are described in a following section.)

- When CPUID.10H.06H:EAX.THREAD_ENABLE[0] is 1, software should enable RP in each logical processor's IA32_RESOURCE_PRIORITY MSR.
- When CPUID.10H.06H:EAX.PACKAGE_ENABLE[1] is 1, software should enable RP in the IA32_RESOURCE_PRIORITY_PKG MSR of each physical processor package.

The processor will provide uniform RP support across all logical processors in a platform. Thus, the contents of the CPUID.10H.06H and CPUID.28H.06H CPUID sub-leaves, if supported, will be identical.

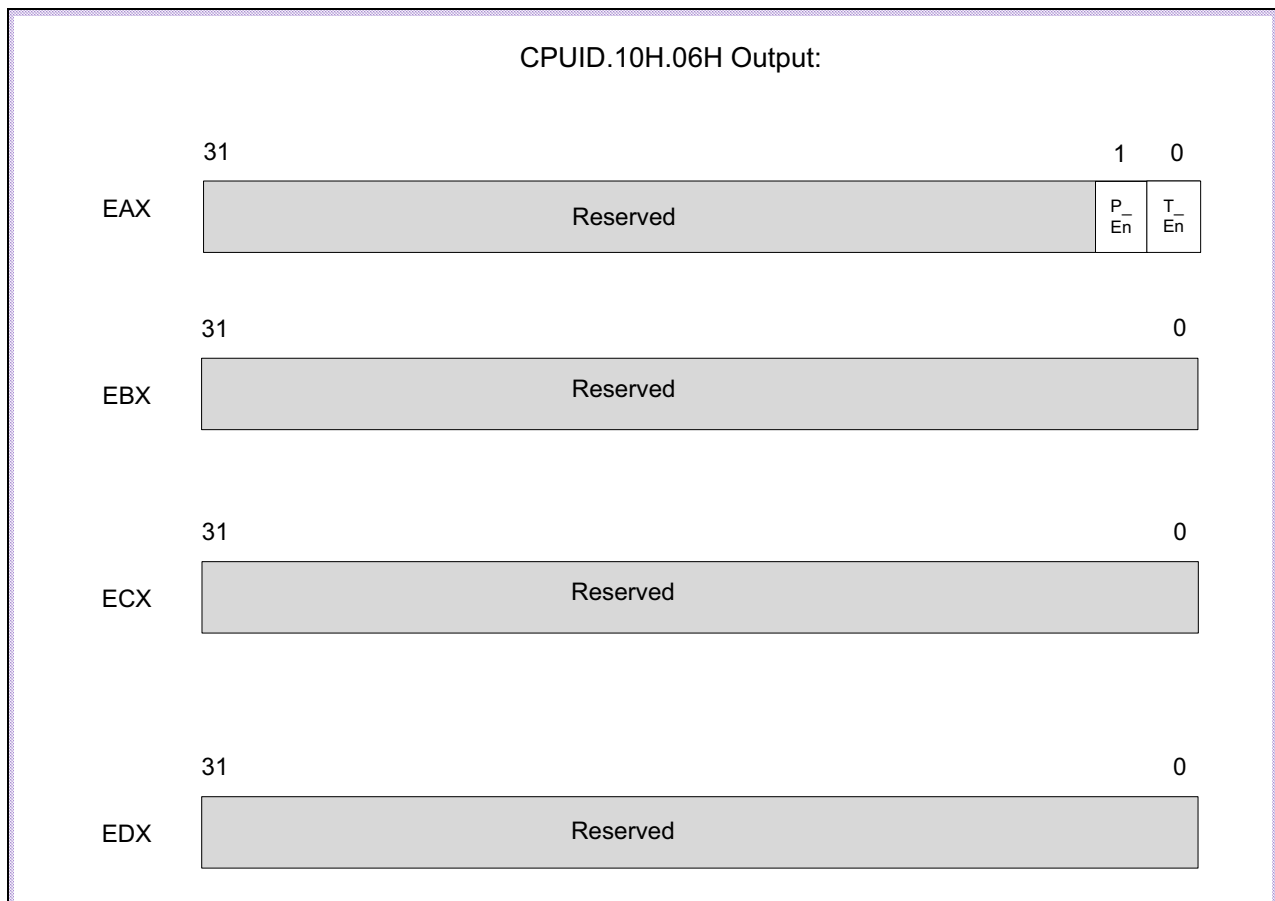


Figure 20-43. CPUID.10H.06H, RP Feature Details Identification

CPUID.10H.06H:EDX[15:0] will remain reserved and return zeroes to support the operation of the CLOS width formula in Section 20.19.4.4.

20.21.2.1.2 ACPI Enumeration

After support for RP is detected using CPUID, the Continuous Performance Control (“_CPC”) object method defined by the ACPI CPPC feature should be consulted to confirm BIOS support is present and that BIOS has preconfigured RP resource mappings. The “_CPC” object method then enumerates the number of priority levels and priority MSR which is used to set priority level for the thread. On Intel platforms, the priority MSR field contains the address of the IA32_PQR_ASSOC MSR.

RP firmware support enumeration is through ACPI (see ACPI Specification version 6.6 or newer for the definition of Collaborative Processor Performance Control (CPPC) and its component objects and data entries). See the Intel® Resource Director Technology Architecture Specification for additional details including ACPI usage flow.

The fields above are described in the Continuous Performance Control (“_CPC”) section of the ACPI specification, within the “Resource Priority Register Descriptor” field.

20.21.2.2 Resource Priority Feature Enable MSRs

When enumerated by CPUID.10H.06H as described previously, the per-thread IA32_RESOURCE_PRIORITY MSR controls the operation of RP for its associated logical processor. For each logical processor, software sets the MSR’s ENABLE field (bit 0) to 1 to enable RP.

In a similar manner, when enumerated by CPUID.10H.06H, the per-package IA32_RESOURCE_PRIORITY_PKG MSR controls the operation of RP for the associated physical processor package. All logical processors within a package share the same MSR. For each physical processor package, software sets the MSR’s ENABLE field (bit 0) to 1 to enable RP.

Before using RP, software should set the per-package enable bits in the IA32_RESOURCE_PRIORITY_PKG MSR, if supported, and then the per-logical-processor enable bits in IA32_RESOURCE_PRIORITY MSR, if supported.

While an overview of the CPUID, MSR, ACPI definitions are described in this section, more details including enable and disable flows are in the Intel® Resource Director Technology (Intel® RDT) Architecture Specification, version 1.4 or newer.

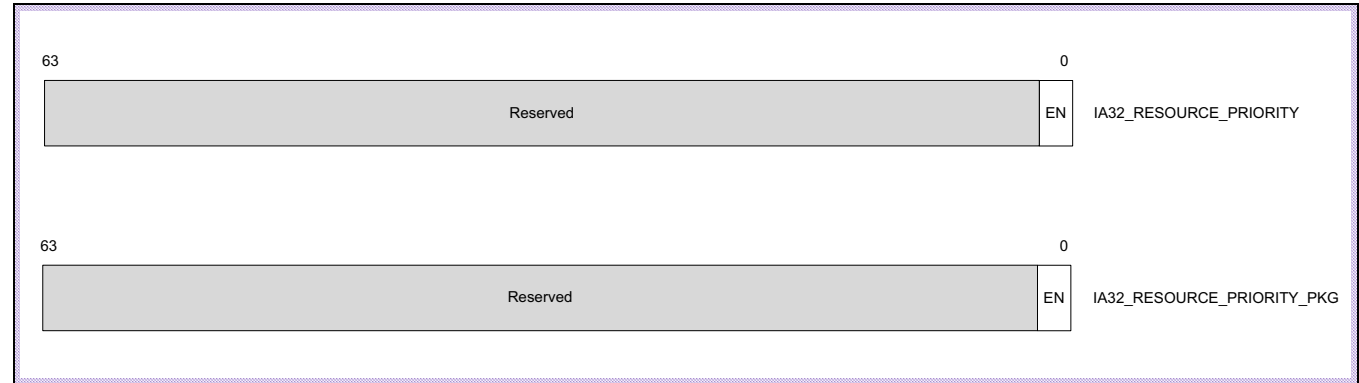


Figure 20-44. IA32_RESOURCE_PRIORITY and IA32_RESOURCE_PRIORITY_PKG Layouts

10. Updates to Chapter 22, Volume 3B

Change bars and violet text show changes to Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- Updated Section 22.10.5.1 for virtualization. This section was previously titled "Virtual Machine Extension (VMX)" and was re-titled to "Virtualization."

Intel 64 and IA-32 architectures provide facilities for monitoring performance via a PMU (Performance Monitoring Unit).

NOTE

Performance monitoring events can be found here: <https://perfmon-events.intel.com/>.

Additionally, performance monitoring event files for Intel processors are maintained by Intel. These files can be downloaded here: <https://github.com/intel/perfmon>.

22.1 PERFORMANCE MONITORING OVERVIEW

Performance monitoring was introduced in the Pentium processor with a set of model-specific performance-monitoring counter MSRs. These counters permit selection of processor performance parameters to be monitored and measured. The information obtained from these counters can be used for tuning system and compiler performance.

In Intel P6 family of processors, the performance monitoring mechanism was enhanced to permit a wider selection of events to be monitored and to allow greater control events to be monitored. Next, Intel processors based on Intel NetBurst microarchitecture introduced a distributed style of performance monitoring mechanism and performance events.

The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, and Intel processors based on Intel NetBurst microarchitecture are not architectural. They are all model specific (not compatible among processor families). Intel Core Solo and Intel Core Duo processors support a set of architectural performance events and a set of non-architectural performance events. Newer Intel processor generations support enhanced architectural performance events and non-architectural performance events.

Starting with Intel Core Solo and Intel Core Duo processors, there are two classes of performance monitoring capabilities. The first class supports events for monitoring performance using counting or interrupt-based event sampling usage. These events are non-architectural and vary from one processor model to another. They are similar to those available in Pentium M processors. These non-architectural performance monitoring events are specific to the microarchitecture and may change with enhancements. They are discussed in Section 22.6.3, "Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)." Non-architectural events for a given microarchitecture cannot be enumerated using CPUID; and they can be found at: <https://perfmon-events.intel.com/> or at <https://github.com/intel/perfmon/>.

The second class of performance monitoring capabilities is referred to as architectural performance monitoring. This class supports the same counting and Interrupt-based event sampling usages, with a smaller set of available events. The visible behavior of architectural performance events is consistent across processor implementations. Availability of architectural performance monitoring capabilities is enumerated using the CPUID.0AH. These events are discussed in Section 22.2.

See also:

- Section 22.2, "Architectural Performance Monitoring."
- Section 22.3, "Performance Monitoring (Intel® Core™ Processors and Intel® Xeon® Processors)."
 - Section 22.3.1, "Performance Monitoring for Processors Based on Nehalem Microarchitecture."
 - Section 22.3.2, "Performance Monitoring for Processors Based on Westmere Microarchitecture."
 - Section 22.3.3, "Intel® Xeon® Processor E7 Family Performance Monitoring Facility."
 - Section 22.3.4, "Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture."
 - Section 22.3.5, "3rd Generation Intel® Core™ Processor Performance Monitoring Facility."
 - Section 22.3.6, "4th Generation Intel® Core™ Processor Performance Monitoring Facility."

- Section 22.3.7, "5th Generation Intel® Core™ Processor and Intel® Core™ M Processor Performance Monitoring Facility."
- Section 22.3.8, "6th Generation, 7th Generation and 8th Generation Intel® Core™ Processor Performance Monitoring Facility."
- Section 22.3.9, "10th Generation Intel® Core™ Processor Performance Monitoring Facility."
- Section 22.3.10, "12th and 13th Generation Intel® Core™ Processors, and 4th and 5th Generation Intel® Xeon® Scalable Processor Family Performance Monitoring Facility."
- Section 22.3.11, "Intel® Series 2 Core™ Ultra Processor Performance Monitoring Facility."
- Section 22.4, "Performance monitoring (Intel® Xeon™ Phi Processors)."
- Section 22.4.1, "Intel® Xeon Phi™ Processor 7200/5200/3200 Performance Monitoring."
- Section 22.5, "Performance Monitoring (Intel Atom® Processors)."
- Section 22.5.1, "Performance Monitoring (45 nm and 32 nm Intel Atom® Processors)."
- Section 22.5.2, "Performance Monitoring for Silvermont Microarchitecture."
- Section 22.5.3, "Performance Monitoring for Goldmont Microarchitecture."
- Section 22.5.4, "Performance Monitoring for Goldmont Plus Microarchitecture."
- Section 22.5.5, "Performance Monitoring for Tremont Microarchitecture."
- Section 22.6, "Performance Monitoring (Legacy Intel Processors)."
- Section 22.6.1, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)."
- Section 22.6.2, "Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)."
- Section 22.6.3, "Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)."
- Section 22.6.4, "Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture."
- Section 22.6.4.5, "Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture."
- Section 22.6.5, "Performance Monitoring and Dual-Core Technology."
- Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."
- Section 22.6.7, "Performance Monitoring on L3 and Caching Bus Controller Sub-Systems."
- Section 22.6.8, "Performance Monitoring (P6 Family Processor)."
- Section 22.6.9, "Performance Monitoring (Pentium Processors)."
- Section 22.7, "Counting Clocks."
- Section 22.8, "IA32_PERF_CAPABILITIES MSR Enumeration."
- Section 22.9, "PEBS Facility."
- Section 22.10, "Architectural PEBS."

22.2 ARCHITECTURAL PERFORMANCE MONITORING

Performance monitoring events are architectural when they behave consistently across microarchitectures. Intel Core Solo and Intel Core Duo processors introduced architectural performance monitoring. The feature provides a mechanism for software to enumerate performance events and provides configuration and counting facilities for events.

Architectural performance monitoring does allow for enhancement across processor implementations. The CPUID.0AH leaf provides version ID for each enhancement. Intel Core Solo and Intel Core Duo processors support base level functionality identified by version ID of 1. Processors based on Intel Core microarchitecture support, at a minimum, the base level functionality of architectural performance monitoring. Intel Core 2 Duo processor T

7700 and newer processors based on Intel Core microarchitecture support both the base level functionality and enhanced architectural performance monitoring identified by version ID of 2.

45 nm and 32 nm Intel Atom processors and Intel Atom processors based on the Silvermont microarchitecture support the functionality provided by versionID 1, 2, and 3; CPUID.0AH:EAX[7:0] reports versionID = 3 to indicate the aggregate of architectural performance monitoring capabilities. Intel Atom processors based on the Airmont microarchitecture support the same performance monitoring capabilities as those based on the Silvermont microarchitecture. Intel Atom processors based on the Goldmont and Goldmont Plus microarchitectures support versionID 4. Intel Atom processors starting with processors based on the Tremont microarchitecture support versionID 5.

Intel Core processors and related Intel Xeon processor families based on the Nehalem through Broadwell microarchitectures support version ID 3. Intel processors based on the Skylake through Coffee Lake microarchitectures support versionID 4. Intel processors starting with processors based on the Ice Lake microarchitecture support versionID 5.

22.2.1 Architectural Performance Monitoring Version 1

Configuring an architectural performance monitoring event involves programming performance event select registers. There are a finite number of performance event select MSRs (IA32_PERFEVTSELx MSRs). The result of a performance monitoring event is reported in a performance monitoring counter (IA32_PMCx MSR). Performance monitoring counters are paired with performance monitoring select registers.

Performance monitoring select registers and counters are architectural in the following respects:

- The bit field layout of IA32_PERFEVTSELx is consistent across microarchitectures. A non-zero write of a field that is introduced after the initial implementation of architectural performance monitoring (Version 1) results in #GP if that field is not supported.
- Addresses of IA32_PERFEVTSELx MSRs remain the same across microarchitectures.
- Addresses of IA32_PMC MSRs remain the same across microarchitectures.
- Each logical processor has its own set of IA32_PERFEVTSELx and IA32_PMCx MSRs. Configuration facilities and counters are not shared between logical processors sharing a processor core.

Architectural performance monitoring provides a CPUID mechanism for enumerating the following information:

- Number of performance monitoring counters available to software in a logical processor (each IA32_PERFEVTSELx MSR is paired to the corresponding IA32_PMCx MSR).
- Number of bits supported in each IA32_PMCx.
- Number of architectural performance monitoring events supported in a logical processor.

Software can use CPUID to discover architectural performance monitoring availability (CPUID.0AH). The architectural performance monitoring leaf provides an identifier corresponding to the version number of architectural performance monitoring available in the processor.

The version identifier is retrieved by querying CPUID.0AH:EAX[7:0] (see Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). If the version identifier is greater than zero, architectural performance monitoring capability is supported. Software queries the CPUID.0AH for the version identifier first; it then analyzes the value returned in CPUID.0AH:EAX, CPUID.0AH:EBX to determine the facilities available.

In the initial implementation of architectural performance monitoring; software can determine how many IA32_PERFEVTSELx/ IA32_PMCx MSR pairs are supported per core, the bit-width of PMC, and the number of architectural performance monitoring events available.

Architectural performance monitoring facilities include a set of performance monitoring counters and performance event select registers. These MSRs have the following properties:

- IA32_PMCx MSRs start at address 0C1H and occupy a contiguous block of MSR address space; the number of MSRs per logical processor is reported using CPUID.0AH:EAX[15:8]. Note that this may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.

- IA32_PERFEVTSELx MSRs start at address 186H and occupy a contiguous block of MSR address space. Each performance event select register is paired with a corresponding performance counter in the 0C1H address block. Note the number of IA32_PERFEVTSELx MSRs may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.
- The bit width of an IA32_PMCx MSR is reported using the CPUID.0AH:EAX[23:16]. This the number of valid bits for read operation. On write operations, the lower-order 32 bits of the MSR may be written with any value, and the high-order bits are sign-extended from the value of bit 31.
- Bit field layout of IA32_PERFEVTSELx MSRs is defined architecturally.

See Figure 22-1 for the bit field layout of IA32_PERFEVTSELx MSRs. The bit fields are:

- **Event select field (bits 0 through 7)** — Selects the event logic unit used to detect microarchitectural conditions (see Table 22-3, for a list of architectural events and their 8-bit codes). The set of values for this field is defined architecturally; each value corresponds to an event logic unit for use with an architectural performance event. The number of architectural events is queried using CPUID.0AH:EAX. A processor may support only a subset of pre-defined values.

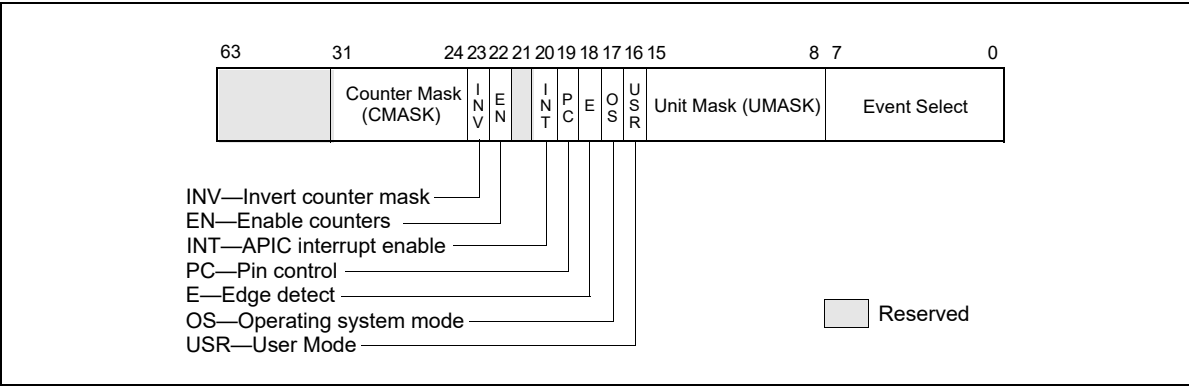


Figure 22-1. Layout of IA32_PERFEVTSELx MSRs

- **Unit mask (UMASK) field (bits 8 through 15)** — These bits qualify the condition that the selected event logic unit detects. Valid UMASK values for each event logic unit are specific to the unit. For each architectural performance event, its corresponding UMASK value defines a specific microarchitectural condition.

A pre-defined microarchitectural condition associated with an architectural event may not be applicable to a given processor. The processor then reports only a subset of pre-defined architectural events. Pre-defined architectural events are listed in Table 22-3; support for pre-defined architectural events is enumerated using CPUID.0AH:EBX.

- **USR (user mode) flag (bit 16)** — Specifies that the selected microarchitectural condition is counted when the logical processor is operating at privilege levels 1, 2 or 3. This flag can be used with the OS flag.
- **OS (operating system mode) flag (bit 17)** — Specifies that the selected microarchitectural condition is counted when the logical processor is operating at privilege level 0. This flag can be used with the USR flag.
- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of the selected microarchitectural condition. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by the other fields. The mechanism does not permit back-to-back assertions to be distinguished.

This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

- **PC (pin control) flag (bit 19)** — Beginning with Sandy Bridge microarchitecture, this bit is reserved (not writeable). On processors based on previous microarchitectures, the logical processor toggles the PMi pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PMi pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.

- **INT (APIC interrupt enable) flag (bit 20)** — When set, the logical processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — When set, performance counting is enabled in the corresponding performance-monitoring counter; when clear, the corresponding counter is disabled. The event logic unit for a UMASK must be disabled by setting `IA32_PERFEVTSELx[bit 22] = 0`, before writing to `IA32_PMCx`.
- **INV (invert) flag (bit 23)** — When set, inverts the counter-mask (CMASK) comparison, so that both greater than or equal to and less than comparisons can be made (0: greater than or equal; 1: less than). Note if counter-mask is programmed to zero, INV flag is ignored.
- **Counter mask (CMASK) field (bits 24 through 31)** — When this field is not zero, a logical processor compares this mask to the events count of the detected microarchitectural condition during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented.

This mask is intended for software to characterize microarchitectural conditions that can count multiple occurrences per cycle (for example, two or more instructions retired per clock; or bus queue occupations). If the counter-mask field is 0, then the counter is incremented each cycle by the event count associated with multiple occurrences.

NOTE

A non-zero write of a field that is introduced in a later architectural performance monitoring version results in a general-protection (#GP) exception if that field is not supported by prior versions.

22.2.2 Architectural Performance Monitoring Version 2

The enhanced features provided by architectural performance monitoring version 2 include the following:

- **Fixed-function performance counter register and associated control register** — Three of the architectural performance events are counted using three fixed-function MSRs (`IA32_FIXED_CTR0` through `IA32_FIXED_CTR2`). Each of the fixed-function PMC can count only one architectural performance event.
Configuring the fixed-function PMCs is done by writing to bit fields in the MSR (`IA32_FIXED_CTR_CTRL`) located at address 38DH. Unlike configuring performance events for general-purpose PMCs (`IA32_PMCx`) via UMASK field in (`IA32_PERFEVTSELx`), configuring, programming `IA32_FIXED_CTR_CTRL` for fixed-function PMCs do not require any UMASK.
- **Simplified event programming** — Most frequent operation in programming performance events are enabling/disabling event counting and checking the status of counter overflows. Architectural performance event version 2 provides three architectural MSRs:
 - `IA32_PERF_GLOBAL_CTRL` allows software to enable/disable event counting of all or any combination of fixed-function PMCs (`IA32_FIXED_CTRx`) or any general-purpose PMCs via a single WRMSR.
 - `IA32_PERF_GLOBAL_STATUS` allows software to query counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single RDMSR.
 - `IA32_PERF_GLOBAL_OVF_CTRL` allows software to clear counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single WRMSR.
- **PMI Overhead Mitigation** — Architectural performance monitoring version 2 introduces two bit field interface in `IA32_DEBUGCTL` for PMI service routine to accumulate performance monitoring data and LBR records with reduced perturbation from servicing the PMI. The two bit fields are:
 - `IA32_DEBUGCTL.Freeze_LBR_On_PMI(bit 11)`. In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 20.4.7 for details of the legacy Freeze LBRs on PMI control.
 - `IA32_DEBUGCTL.Freeze_PerfMon_On_PMI(bit 12)`. In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 20.4.7 for details of the legacy Freeze LBRs on PMI control.

The facilities provided by architectural performance monitoring version 2 can be queried from `CPUID.0AH` by examining the content of register EDX:

- Bits 0 through 4 of CPUID.0AH:EDX indicates the number of fixed-function performance counters available per core,
- Bits 5 through 12 of CPUID.0AH:EDX indicates the bit-width of fixed-function performance counters. Bits beyond the width of the fixed-function counter are reserved and must be written as zeros.

NOTE

Early generation of processors based on Intel Core microarchitecture may report in CPUID.0AH:EDX of support for version 2 but indicating incorrect information of version 2 facilities.

The IA32_FIXED_CTR_CTRL MSR include multiple sets of 4-bit field, each 4 bit field controls the operation of a fixed-function performance counter. Figure 22-2 shows the layout of 4-bit controls for each fixed-function PMC. Two sub-fields are currently defined within each control. The definitions of the bit fields are:

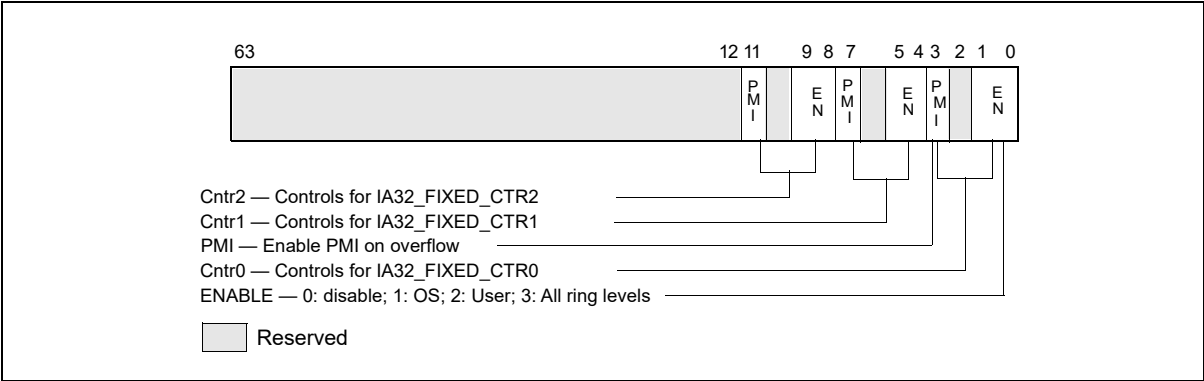


Figure 22-2. Layout of IA32_FIXED_CTR_CTRL MSR

- **Enable field (lowest 2 bits within each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring 0. When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring greater than 0. Writing 0 to both bits stops the performance counter. Writing a value of 11B enables the counter to increment irrespective of privilege levels.
- **PMI field (the fourth bit within each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

IA32_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting of each performance counter. Figure 22-3 shows the layout of IA32_PERF_GLOBAL_CTRL. Each enable bit in IA32_PERF_GLOBAL_CTRL is ANDed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the ANDed results is true; counting is disabled when the result is false.

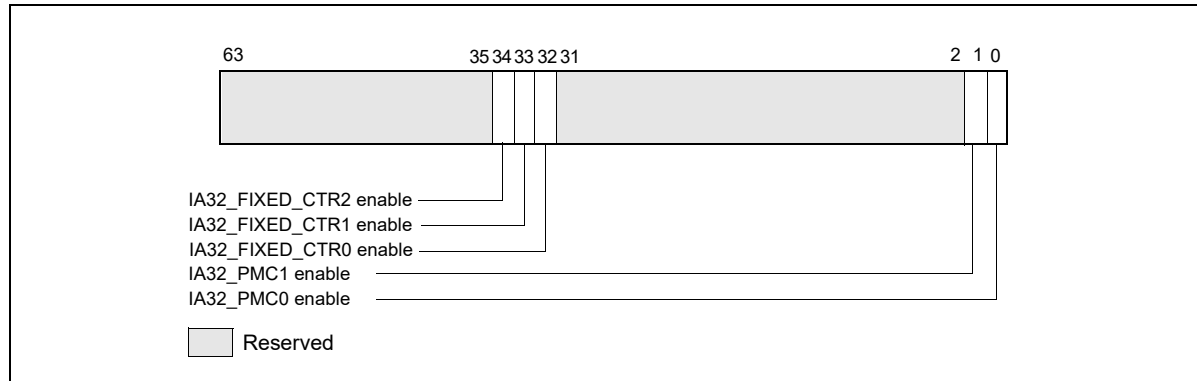


Figure 22-3. Layout of IA32_PERF_GLOBAL_CTRL MSR

The behavior of the fixed function performance counters supported by architectural performance version 2 is expected to be consistent on all processors that support those counters, and is defined as follows.

Table 22-1. Association of Fixed-Function Performance Counters with Architectural Performance Events

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
IA32_FIXED_CTR0	309H	INST_RETIRED.ANY	This event counts the number of instructions that retire execution. For instructions that consist of multiple uops, this event counts the retirement of the last uop of the instruction. The counter continues counting during hardware interrupts, traps, and in-side interrupt handlers.
IA32_FIXED_CTR1	30AH	CPU_CLK_UNHALTED.THREAD CPU_CLK_UNHALTED.CORE	The CPU_CLK_UNHALTED.THREAD event counts the number of core cycles while the logical processor is not in a halt state. If there is only one logical processor in a processor core, CPU_CLK_UNHALTED.CORE counts the unhalted cycles of the processor core. The core frequency may change from time to time due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason this event may have a changing ratio with regards to time.
IA32_FIXED_CTR2	30BH	CPU_CLK_UNHALTED.REF_TSC	This event counts the number of reference cycles at the TSC rate when the core is not in a halt state and not in a TM stop-clock state. The core enters the halt state when it is running the HLT instruction or the MWAIT instruction. This event is not affected by core frequency changes (e.g., P states) but counts at the same frequency as the time stamp counter. This event can approximate elapsed time while the core was not in a halt state and not in a TM stopclock state.
IA32_FIXED_CTR3	30CH	TOPDOWN.SLOTS	This event counts the number of available slots for an unhalted logical processor. The event increments by machine-width of the narrowest pipeline as employed by the Top-down Microarchitecture Analysis method. The count is distributed among unhalted logical processors (hyper-threads) who share the same physical core. Software can use this event as the denominator for the top-level metrics of the Top-down Microarchitecture Analysis method.

Table 22-1. Association of Fixed-Function Performance Counters with Architectural Performance Events

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
IA32_FIXED_CTR4 ¹	30DH	TOPDOWN_BAD_SPECULATION	This event counts Topdown slots that were not consumed by the backend due to a pipeline flush, such as a mispredicted branch or a machine clear. It provides a value equivalent to a general-purpose counter configured with UMask=00H and EventSelect=73H.
IA32_FIXED_CTR5 ¹	30EH	TOPDOWN_FE_BOUND	This event counts Topdown slots where uops were not provided to the backend due to frontend limitations, such as instruction cache/TLB miss delays or decoder limitations. It provides a value equivalent to a general purpose counter configured with UMask=01H and EventSelect=9CH.
IA32_FIXED_CTR6 ¹	30FH	TOPDOWN_RETIRING	This event counts Topdown slots that were committed (retired) by the backend. It provides a value equivalent to a general purpose counter configured with UMask=02H and EventSelect=C2H.

NOTES:

1. If this counter is supported, it will be accessible in the following MSRs: IA32_PERF_GLOBAL_STATUS (38EH), IA32_PERF_GLOBAL_CTRL (38FH), IA32_PERF_GLOBAL_STATUS_RESET (390H), and IA32_PERF_GLOBAL_STATUS_SET (391H).

IA32_PERF_GLOBAL_STATUS MSR provides single-bit status for software to query the overflow condition of each performance counter. IA32_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer. IA32_PERF_GLOBAL_STATUS[bit 63] provides a CondChgd bit to indicate changes to the state of performance monitoring hardware. Figure 22-4 shows the layout of IA32_PERF_GLOBAL_STATUS. A value of 1 in bits 0, 1, 32 through 34 indicates a counter overflow condition has occurred in the associated counter.

When a performance counter is configured for PEBS, overflow condition in the counter generates a performance-monitoring interrupt signaling a PEBS event. On a PEBS event, the processor stores data records into the buffer area, clears the counter overflow status., and sets the "OvfBuffer" bit in IA32_PERF_GLOBAL_STATUS.

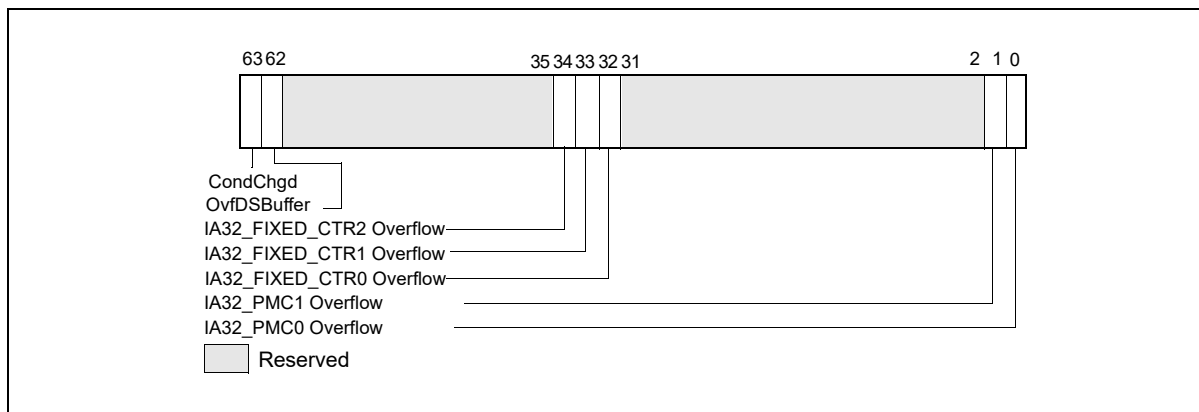


Figure 22-4. Layout of IA32_PERF_GLOBAL_STATUS MSR

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow indicator(s) of any general-purpose or fixed-function counters via a single WRMSR. Software should clear overflow indications when

- Setting up new values in the event select and/or UMASK field for counting or interrupt-based event sampling.
- Reloading counter values to continue collecting next sample.
- Disabling event counting or interrupt-based event sampling.

The layout of IA32_PERF_GLOBAL_OVF_CTL is shown in Figure 22-5.

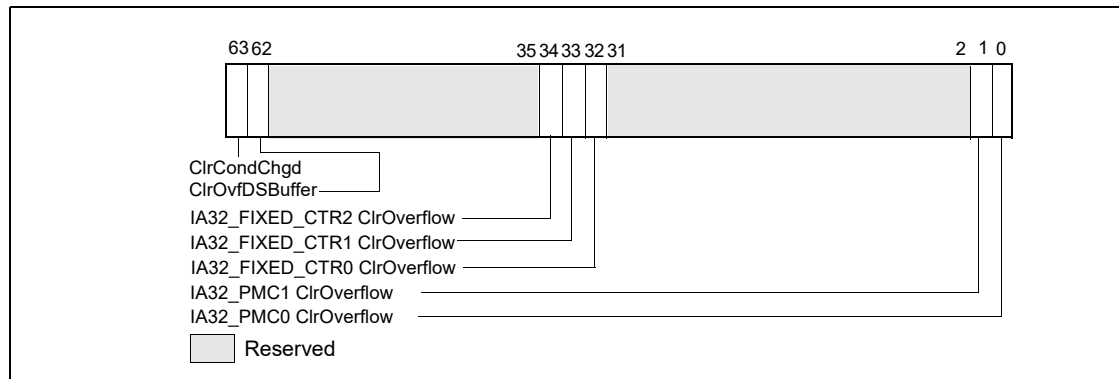


Figure 22-5. Layout of IA32_PERF_GLOBAL_OVF_CTRL MSR

22.2.3 Architectural Performance Monitoring Version 3

Processors supporting architectural performance monitoring version 3 also supports version 1 and 2, as well as capability enumerated by CPUID.0AH. Specifically, version 3 provides the following enhancement in performance monitoring facilities if a processor core comprising of more than one logical processor, i.e., a processor core supporting Intel Hyper-Threading Technology or simultaneous multi-threading capability:

- AnyThread counting for processor core supporting two or more logical processors. The interface that supports AnyThread counting include:
 - Each IA32_PERFEVTSELx MSR (starting at MSR address 186H) support the bit field layout defined in Figure 22-6.

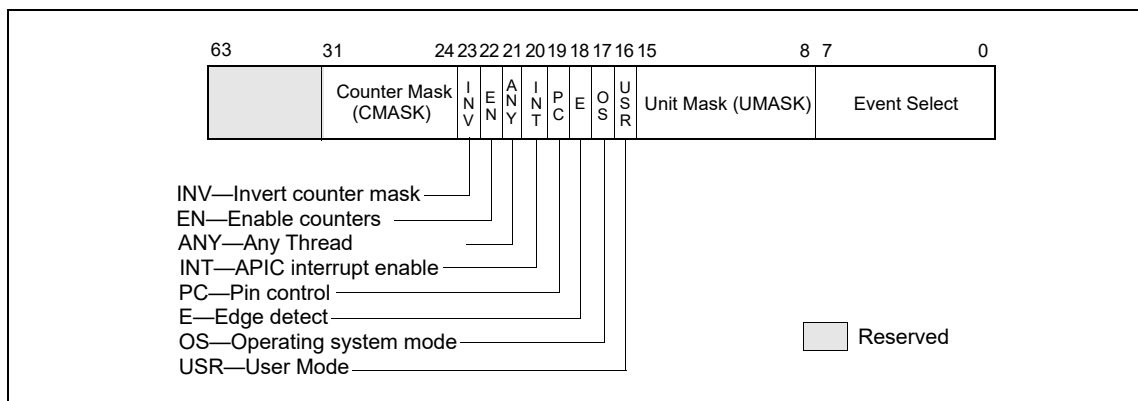


Figure 22-6. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

Bit 21 (AnyThread) of IA32_PERFEVTSELx is supported in architectural performance monitoring version 3 for processor core comprising of two or more logical processors. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring across all logical processors sharing a processor core. When bit 21 is 0, the counter only increments the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring in the logical processor which programmed the IA32_PERFEVTSELx MSR.

- Each fixed-function performance counter IA32_FIXED_CTRx (starting at MSR address 309H) is configured by a 4-bit control block in the IA32_PERF_FIXED_CTR_CTRL MSR. The control block also allows thread-specificity configuration using an AnyThread bit for fixed-function counters 0, 1, and 2. The layout of IA32_PERF_FIXED_CTR_CTRL MSR is shown.

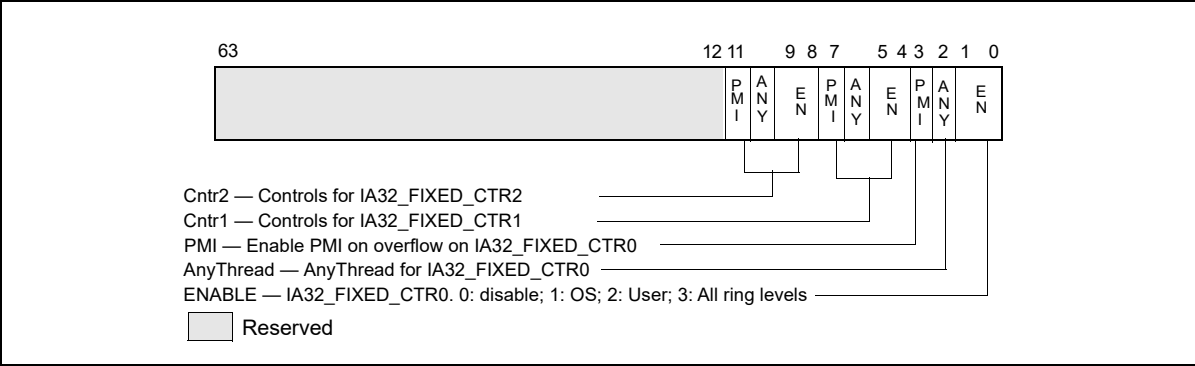


Figure 22-7. IA32_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3

- Each control block for a fixed-function performance counter provides an **AnyThread** (bit position $2 + 4 \cdot N$, $N = 0, 1$, etc.) bit. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the ENABLE setting of the corresponding control block of IA32_PERF_FIXED_CTR_CTRL) occurring across all logical processors sharing a processor core. When an **AnyThread** bit is 0 in IA32_PERF_FIXED_CTR_CTRL, the corresponding fixed-function counter only increments the associated event conditions occurring in the logical processor which programmed the IA32_PERF_FIXED_CTR_CTRL MSR.
- The IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_OVF_CTRL MSRs provide single-bit controls/status for each general-purpose and fixed-function performance counter. Figure 22-8 and Figure 22-9 show the layout of these MSRs for N general-purpose performance counters (where N is reported by CPUID.0AH:EAX[15:8]) and three fixed-function counters.

NOTE

The number of general-purpose performance monitoring counters (i.e., N in Figure 22-9) can vary across processor generations within a processor family, across processor families, or could be different depending on the configuration chosen at boot time in the BIOS regarding Intel Hyper Threading Technology, (e.g., $N=2$ for 45 nm Intel Atom processors; $N=4$ for processors based on the Nehalem microarchitecture; for processors based on the Sandy Bridge microarchitecture, $N=4$ if Intel Hyper Threading Technology is active and $N=8$ if not active). In addition, the number of counters may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.

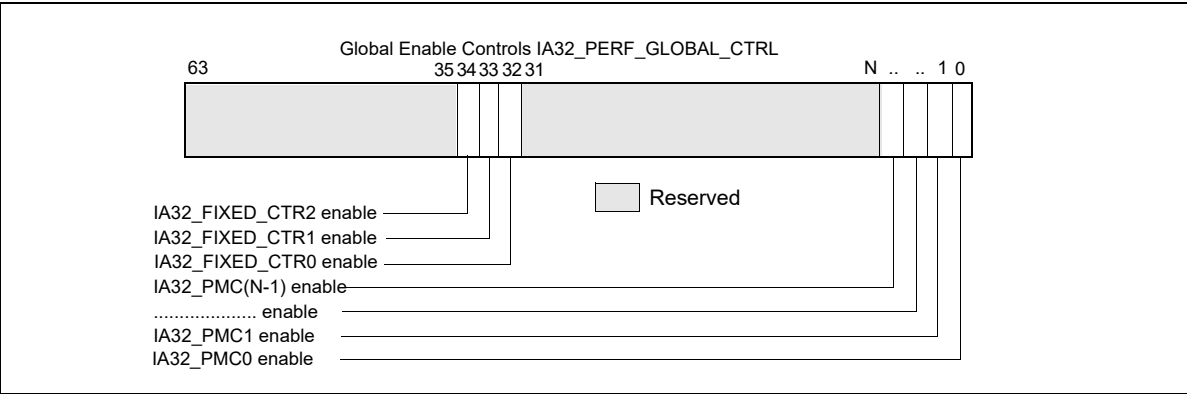


Figure 22-8. Layout of Global Performance Monitoring Control MSR

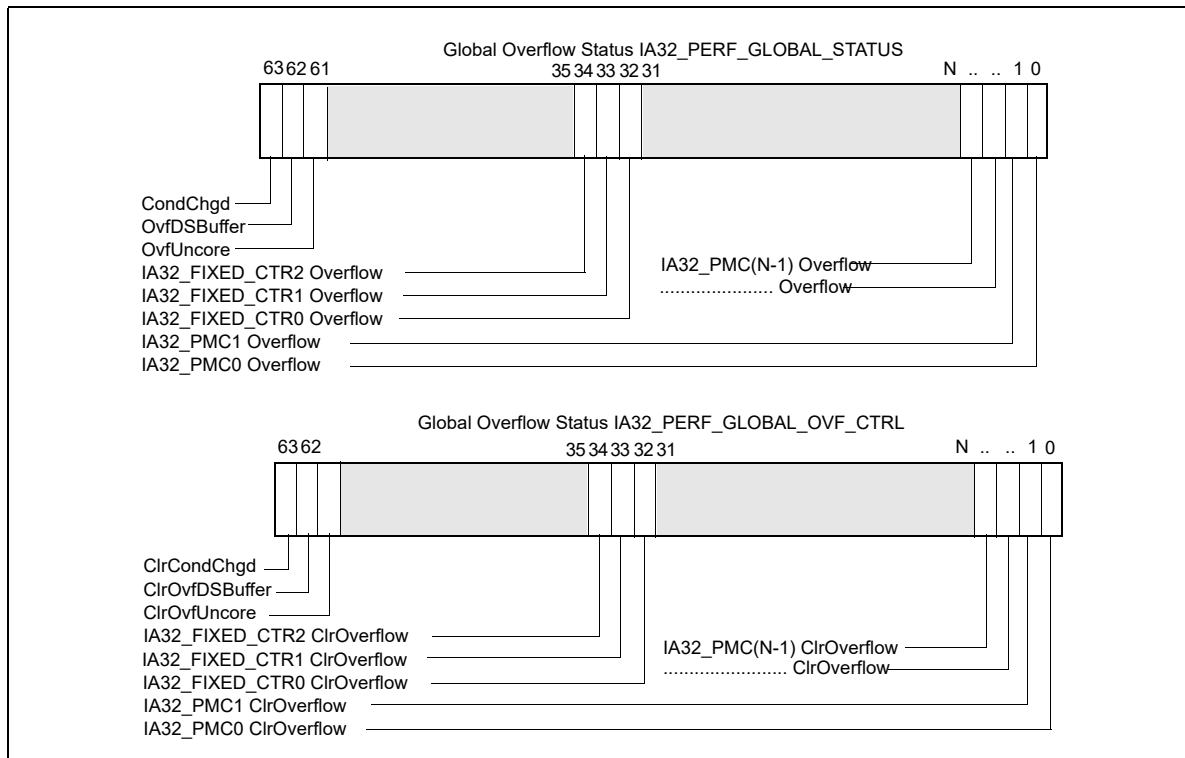


Figure 22-9. Global Performance Monitoring Overflow Status and Control MSRs

22.2.3.1 AnyThread Counting and Software Evolution

The motivation for characterizing software workload over multiple software threads running on multiple logical processors of the same processor core originates from a time earlier than the introduction of the AnyThread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL. While AnyThread counting provides some benefits in simple software environments of an earlier era, the evolution contemporary software environments introduce certain concepts and pre-requisites that AnyThread counting does not comply with.

One example is the proliferation of software environments that support multiple virtual machines (VM) under VMX (see Chapter 26, “Introduction to Virtual Machine Extensions”) where each VM represents a domain separated from one another.

A Virtual Machine Monitor (VMM) that manages the VMs may allow an individual VM to employ performance monitoring facilities to profiles the performance characteristics of a workload. The use of the Anythread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL is discouraged with software environments supporting virtualization or requiring domain separation.

Specifically, Intel recommends VMM:

- Configure the MSR bitmap to cause VM-exits for WRMSR to IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in VMX non-Root operation (see Chapter 27 for additional information),
- Clear the AnyThread bit of IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in the MSR-load lists for VM exits and VM entries (see Chapter 27, Chapter 29, and Chapter 30).

Even when operating in simpler legacy software environments which might not emphasize the pre-requisites of a virtualized software environment, the use of the AnyThread interface should be moderated and follow any event-specific guidance where explicitly noted.

22.2.4 Architectural Performance Monitoring Version 4

Processors supporting architectural performance monitoring version 4 also supports version 1, 2, and 3, as well as capability enumerated by CPUID.0AH. Version 4 introduced a streamlined PMI overhead mitigation interface that replaces the legacy semantic behavior but retains the same control interface in IA32_DEBUGCTL.Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI. Specifically version 4 provides the following enhancements:

- New indicators (LBR_FRZ, CTR_FRZ) in IA32_PERF_GLOBAL_STATUS, see Section 22.2.4.1.
- Streamlined Freeze/PMI Overhead management interfaces to use IA32_DEBUGCTL.Freeze_LBRs_On_PMI and IA32_DEBUGCTL.Freeze_PerfMon_On_PMI: see Section 22.2.4.1. Legacy semantics of Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI (applicable to version 2 and 3) are not supported with version 4 or higher.
- Fine-grain separation of control interface to manage overflow/status of IA32_PERF_GLOBAL_STATUS and read-only performance counter enabling interface in IA32_PERF_GLOBAL_STATUS: see Section 22.2.4.2.
- Performance monitoring resource in-use MSR to facilitate cooperative sharing protocol between perfmon-managing privilege agents.

22.2.4.1 Enhancement in IA32_PERF_GLOBAL_STATUS

The IA32_PERF_GLOBAL_STATUS MSR provides the following indicators with architectural performance monitoring version 4:

- IA32_PERF_GLOBAL_STATUS.LBR_FRZ[bit 58]: This bit is set due to the following conditions:
 - IA32_DEBUGCTL.FREEZE_LBR_ON_PMI has been set by the profiling agent, and
 - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently the LBR stack is frozen.

Effectively, the IA32_PERF_GLOBAL_STATUS.LBR_FRZ bit also serves as a control to enable capturing data in the LBR stack. To enable capturing LBR records, the following expression must hold with architectural PerfMon version 4 or higher:

— $(\text{IA32\_DEBUGCTL.LBR} \& (!\text{IA32\_PERF\_GLOBAL\_STATUS.LBR\_FRZ})) = 1$

- IA32_PERF_GLOBAL_STATUS.CTR_FRZ[bit 59]: This bit is set due to the following conditions:
 - IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI has been set by the profiling agent, and
 - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently, all the performance counters are frozen.

Effectively, the IA32_PERF_GLOBAL_STATUS.CTR_FRZ bit also serve as an read-only control to enable programmable performance counters and fixed counters in the core PMU. To enable counting with the performance counters, the following expression must hold with architectural PerfMon version 4 or higher:

- $(\text{IA32\_PERFEVTSELn.EN} \& \text{IA32\_PERF\_GLOBAL\_CTRL.PMCn} \& (!\text{IA32\_PERF\_GLOBAL\_STATUS.CTR\_FRZ})) = 1$ for programmable counter 'n', or
- $(\text{IA32\_PERF\_FIXED\_CTRL.ENi} \& \text{IA32\_PERF\_GLOBAL\_CTRL.FCi} \& (!\text{IA32\_PERF\_GLOBAL\_STATUS.CTR\_FRZ})) = 1$ for fixed counter 'i'

The read-only enable interface IA32_PERF_GLOBAL_STATUS.CTR_FRZ provides a more efficient flow for a PMI handler to use IA32_DEBUGCTL.Freeze_PerfMon_On_PMI to filter out data that may distort target workload analysis, see Table 20-3. It should be noted the IA32_PERF_GLOBAL_CTRL register continue to serve as the primary interface to control all performance counters of the logical processor.

For example, when the Freeze-On-PMI mode is not being used, a PMI handler would be setting IA32_PERF_GLOBAL_CTRL as the very last step to commence the overall operation after configuring the individual counter registers, controls, and PEBS facility. This does not only assure atomic monitoring but also avoids unnecessary complications (e.g., race conditions) when software attempts to change the core PMU configuration while some counters are kept enabled.

Additionally, IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55]: On processors that support Intel Processor Trace and configured to store trace output packets to physical memory using the ToPA scheme, bit 55 is set when a PMI occurred due to a ToPA entry memory buffer was completely filled.

IA32_PERF_GLOBAL_STATUS also provides an indicator to distinguish interaction of performance monitoring operations with other side-band activities, which apply Intel SGX on processors that support it (for additional information about Intel SGX, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D):

- IA32_PERF_GLOBAL_STATUS.ASCI[bit 60]: This bit is set when data accumulated in any of the configured performance counters (i.e., IA32_PMCx or IA32_FIXED_CTRx) may include contributions from direct or indirect operation of Intel SGX to protect an enclave (since the last time IA32_PERF_GLOBAL_STATUS.ASCI was cleared).

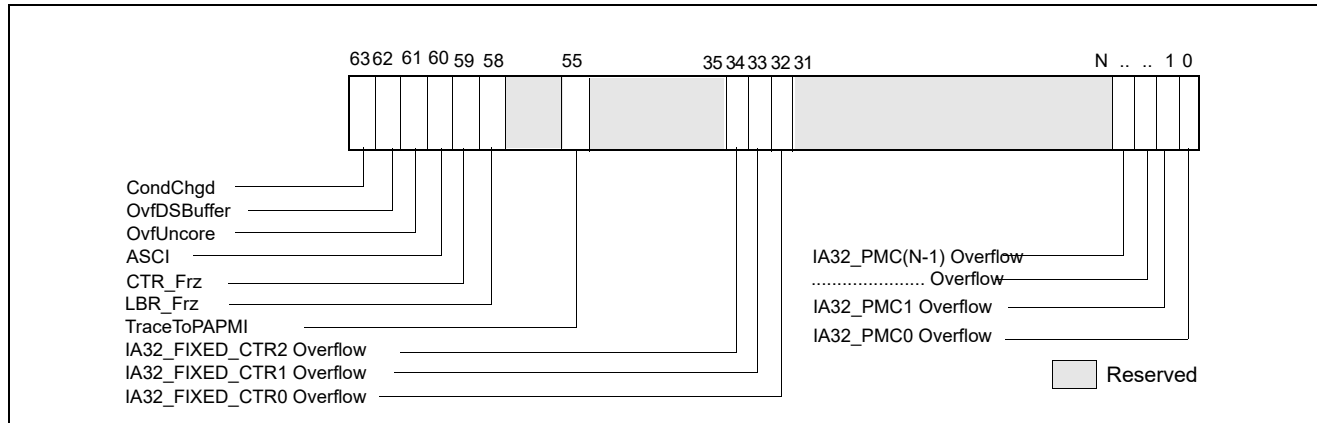


Figure 22-10. IA32_PERF_GLOBAL_STATUS MSR and Architectural PerfMon Version 4

Note, a processor's support for IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55] is enumerated as a result of CPUID enumerated capability of Intel Processor Trace and the use of the ToPA buffer scheme. Support of IA32_PERF_GLOBAL_STATUS.ASCI[bit 60] is enumerated by the CPUID enumeration of Intel SGX.

22.2.4.2 IA32_PERF_GLOBAL_STATUS_RESET and IA32_PERF_GLOBAL_STATUS_SET MSRS

With architectural performance monitoring version 3 and lower, clearing of the set bits in IA32_PERF_GLOBAL_STATUS MSR by software is done via IA32_PERF_GLOBAL_OVF_CTRL MSR. Starting with architectural performance monitoring version 4, software can manage the overflow and other indicators in IA32_PERF_GLOBAL_STATUS using separate interfaces to set or clear individual bits.

The address and the architecturally-defined bits of IA32_PERF_GLOBAL_OVF_CTRL is inherited by IA32_PERF_GLOBAL_STATUS_RESET (see Figure 22-11). Further, IA32_PERF_GLOBAL_STATUS_RESET provides additional bit fields to clear the new indicators in IA32_PERF_GLOBAL_STATUS described in Section 22.2.4.1.

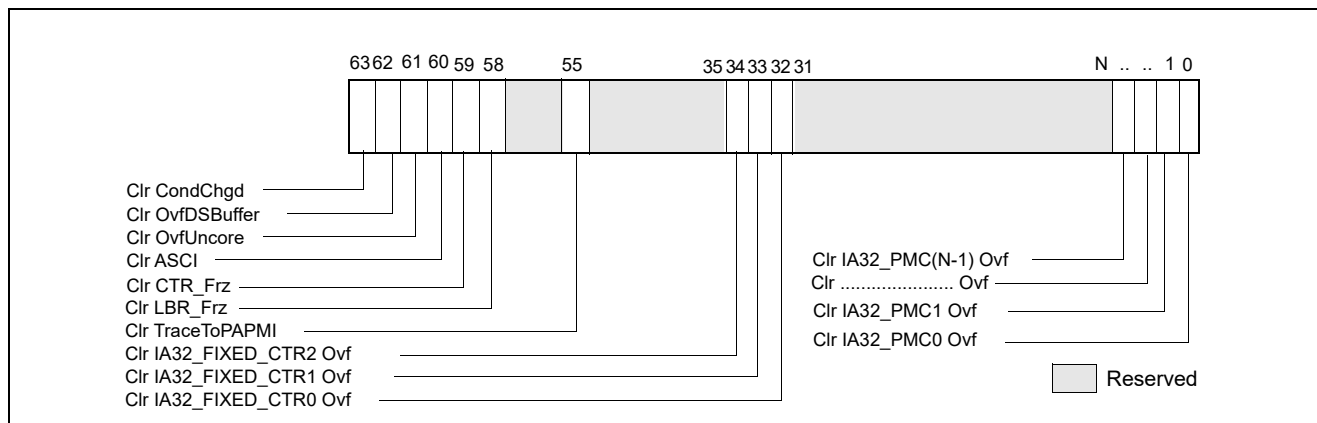


Figure 22-11. IA32_PERF_GLOBAL_STATUS_RESET MSR and Architectural PerfMon Version 4

The IA32_PERF_GLOBAL_STATUS_SET MSR is introduced with architectural performance monitoring version 4. It allows software to set individual bits in IA32_PERF_GLOBAL_STATUS. The IA32_PERF_GLOBAL_STATUS_SET interface can be used by a VMM to virtualize the state of IA32_PERF_GLOBAL_STATUS across VMs.

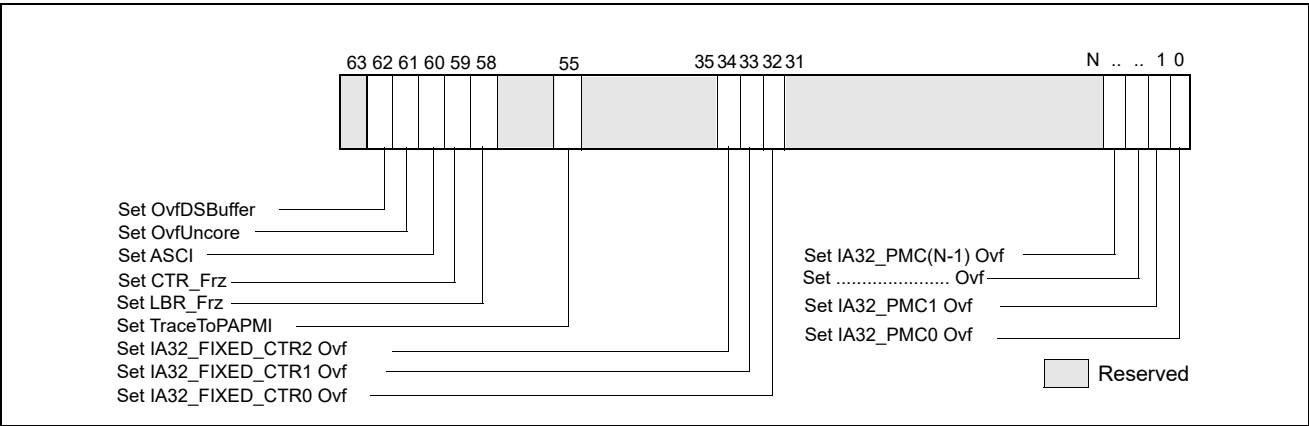


Figure 22-12. IA32_PERF_GLOBAL_STATUS_SET MSR and Architectural PerfMon Version 4

22.2.4.3 IA32_PERF_GLOBAL_INUSE MSR

In a contemporary software environment, multiple privileged service agents may wish to employ the processor’s performance monitoring facilities. The IA32_MISC_ENABLE.PERFMON_AVAILABLE[bit 7] interface could not serve the need of multiple agent adequately. A white paper, “Performance Monitoring Unit Sharing Guideline”¹, proposed a cooperative sharing protocol that is voluntary for participating software agents.

Architectural performance monitoring version 4 introduces a new MSR, IA32_PERF_GLOBAL_INUSE, that simplifies the task of multiple cooperating agents to implement the sharing protocol.

The layout of IA32_PERF_GLOBAL_INUSE is shown in Figure 22-13.

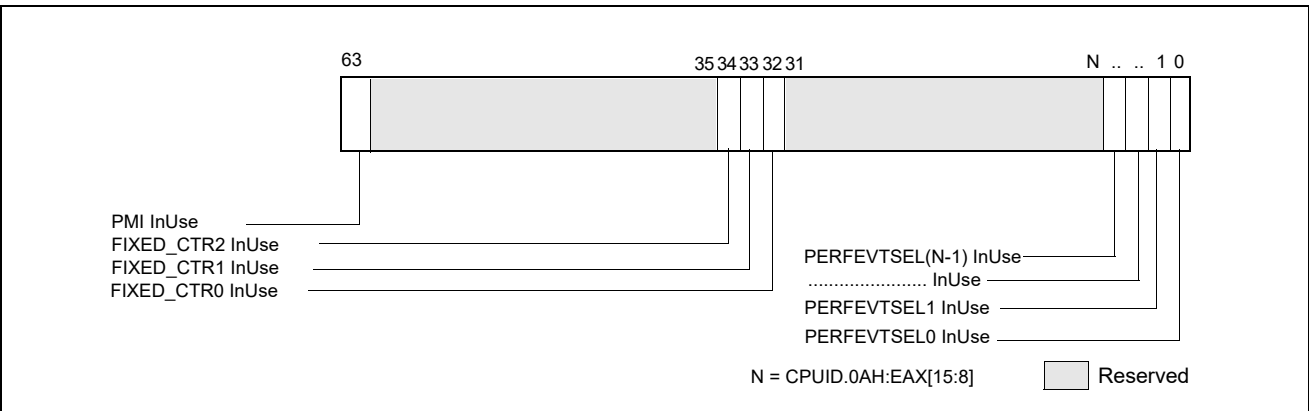


Figure 22-13. IA32_PERF_GLOBAL_INUSE MSR and Architectural PerfMon Version 4

The IA32_PERF_GLOBAL_INUSE MSR provides an “InUse” bit for each programmable performance counter and fixed counter in the processor. Additionally, it includes an indicator if the PMI mechanism has been configured by a profiling agent.

- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL0_InUse[bit 0]: This bit reflects the logical state of (IA32_PERFEVTSEL0[7:0] != 0).

1. Available at <http://www.intel.com/sdm>

- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL1_InUse[bit 1]: This bit reflects the logical state of (IA32_PERFEVTSEL1[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL2_InUse[bit 2]: This bit reflects the logical state of (IA32_PERFEVTSEL2[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSELn_InUse[bit n]: This bit reflects the logical state of (IA32_PERFEVTSELn[7:0] != 0), n < CPUID.0AH:EAX[15:8].
- IA32_PERF_GLOBAL_INUSE.FC0_InUse[bit 32]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[1:0] != 0).
- IA32_PERF_GLOBAL_INUSE.FC1_InUse[bit 33]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[5:4] != 0).
- IA32_PERF_GLOBAL_INUSE.FC2_InUse[bit 34]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[9:8] != 0).
- IA32_PERF_GLOBAL_INUSE.PMI_InUse[bit 63]: This bit is set if any one of the following bit is set:
 - IA32_PERFEVTSELn.INT[bit 20], n < CPUID.0AH:EAX[15:8].
 - IA32_FIXED_CTR_CTRL.ENi_PMI, i = 0, 1, 2.
 - Any IA32_PEBS_ENABLES bit which enables PEBS for a general-purpose or fixed-function performance counter.

22.2.5 Architectural Performance Monitoring Version 5

Processors supporting architectural performance monitoring version 5 also support versions 1, 2, 3, and 4, as well as capability enumerated by CPUID.0AH. Specifically, version 5 provides the following enhancements:

- Deprecation of AnyThread mode, see Section 22.2.5.1.
- Individual enumeration of Fixed counters in CPUID.0AH, see Section 22.2.5.2.
- Domain separation, see Section 22.2.5.3.

22.2.5.1 AnyThread Mode Deprecation

With Architectural Performance Monitoring Version 5, a processor that supports AnyThread mode deprecation is enumerated by CPUID.0AH:EDX[15]. If set, software will not have to follow guidelines in Section 22.2.3.1.

22.2.5.2 Fixed Counter Enumeration

With Architectural Performance Monitoring Version 5, register CPUID.0AH:ECX indicates Fixed Counter enumeration. It is a bit mask which enumerates the supported Fixed Counters in a processor. If bit 'i' is set, it implies that Fixed Counter 'i' is supported. Software is recommended to use the following logic to check if a Fixed Counter is supported on a given processor:

$$\text{FxCtr}[i]_{\text{is\_supported}} := \text{ECX}[i] \mid (\text{EDX}[4:0] > i);$$

22.2.5.3 Domain Separation

When the INV flag in IA32_PERFEVTSELx is used, a counter stops counting when the logical processor exits the C0 ACPI C-state.

22.2.6 Architectural Performance Monitoring Version 6

Processors supporting architectural performance monitoring version 6 also support versions 1, 2, 3, 4, and 5, as well as the capabilities enumerated by CPUID leaves 0AH and 23H. Specifically, version 6 provides the following enhancements:

- PerfMon MSRs Aliasing, see Section 22.2.6.1.

- UnitMask2, see Section 22.2.6.2.
- Equal flag, see Section 22.2.6.3.

22.2.6.1 Performance Monitoring MSR Aliasing

Architectural performance monitoring version 6 includes a new range for the counters' MSRs in the 19xxH address range¹. The new MSR range allows for scaling the number of general-purpose and fixed-function counters beyond the quantities in current products. Additionally, it banks registers of the same counter closer to each other.

All legacy and new counters, i.e., those enumerated in CPUID.23H.01H, will be supported in this new address range. Moving forward, newer counters may be supported in the new address range, but not in the legacy one.

Table 22-2. New Performance Monitoring MSR Naming Details

Register	General Counter n	Fixed Counter m
Counter	IA32_PMC_GPn_CTR	IA32_PMC_FXm_CTR
Event-Select	IA32_PMC_GPn_CFG_A	N/A
Reload Config	IA32_PMC_GPn_CFG_B	IA32_PMC_FXm_CFG_B
Event-Select Extended	IA32_PMC_GPn_CFG_C	IA32_PMC_FXm_CFG_C

An IA32_PMC_GPn_CTR MSR can be used to access the counter value for a GP (general-purpose) counter 'n.' On processors that support CPUID.23H, a general-purpose (GP) counter 'n' that is enumerated in both CPUID.23H and CPUID.0AH can be accessed through either IA32_PMC_GPn_CTR or the legacy MSR addresses (IA32_PMCn, IA32_A_PMCn). In contrast, a counter 'n' that is only enumerated in CPUID.23H can only be accessed through IA32_PMC_GPn_CTR. This guideline also applies to the other MSR aliases described in this section (i.e., IA32_PMC_GPn_CFG_A and IA32_PERFEVTSELn, IA32_PMC_FXm_CTR and IA32_FIXED_CTRm). The IA32_PMC_GPn_CTR MSR address² for counter 'n' is $1900H + 4 * n$, and this MSR has full-width write support.

The IA32_PMC_GPn_CFG_A MSR can be used to access the performance event select register for a GP counter 'n' and is at address³ $1901H + 4 * n$. The reload configuration MSRs for GP counter 'n,' IA32_PMC_GPn_CFG_B, is at MSR address $1902H + 4 * n$. There is no legacy MSR alias to this reload configuration register. Thus, the register only exists when enumerated in CPUID.23H. Similarly, no legacy MSR alias exists for the event-select extended registers, IA32_PMC_GPn_CFG_C, which are at MSR address $1903H + 4 * n$ for GP counter 'n.'

An IA32_PMC_FXm_CTR MSR can be used to access the counter value for a fixed-function counter 'm' if that counter is enumerated in CPUID.23H. The IA32_PMC_FXm_CTR MSR address for fixed-function counter 'm' is $1980H + 4 * m$. There is no alias for the fixed-function counters' reload configuration or event select extended registers (IA32_PMC_FXm_CFG_B at MSR addresses $1982H + 4 * m$ and IA32_PMC_FXm_CFG_C at MSR address $1983H + 4 * m$, respectively).

The available general-purpose and fixed-function counters are reported by CPUID.23H.01H:EAX and CPUID.23H.01H:EBX, respectively.⁴ Note that not all counters enumerated in CPUID.23H may have corresponding IA32_PMC_GPn_CFG_B, IA32_PMC_GPn_CFG_C, IA32_PMC_FXm_CFG_B, or IA32_PMC_FXm_CFG_C MSRs. The enumeration and usage of these MSRs are described in Section 22.11, "Auto Counter Reload." The enumeration in CPUID.23H is true-view, and thus, the enumeration may only be set on (and the MSRs/counters they enumerate only supported on) a subset of the logical processors of the system.

The architectural performance monitoring version 6 enhanced layout of the IA32_PERFEVTSELx MSR is shown in Figure 22-14.

1. This feature is also available in a subset of processors with a CPUID signature value of DisplayFamily_DisplayModel 06_C5H or 06_C6H (though they report IA32_PERF_CAPABILITIES.PEBS_FMT as 5).
2. As an example, the IA32_PMC_GP1_CTR MSR has MSR address 1904H. Note that the legacy full-width MSR addresses for the counters, IA32_A_PMCn MSRs, remains at MSR address $4C1H + n$.
3. As an example, the IA32_PMC_GP1_CFG_A MSR has MSR address 1905H. Note that the legacy MSR address for the event select registers, IA32_PERFEVTSELn MSRs, remain at MSR address $186H + n$.
4. The valid range of fixed-function counters is 0 through 15.

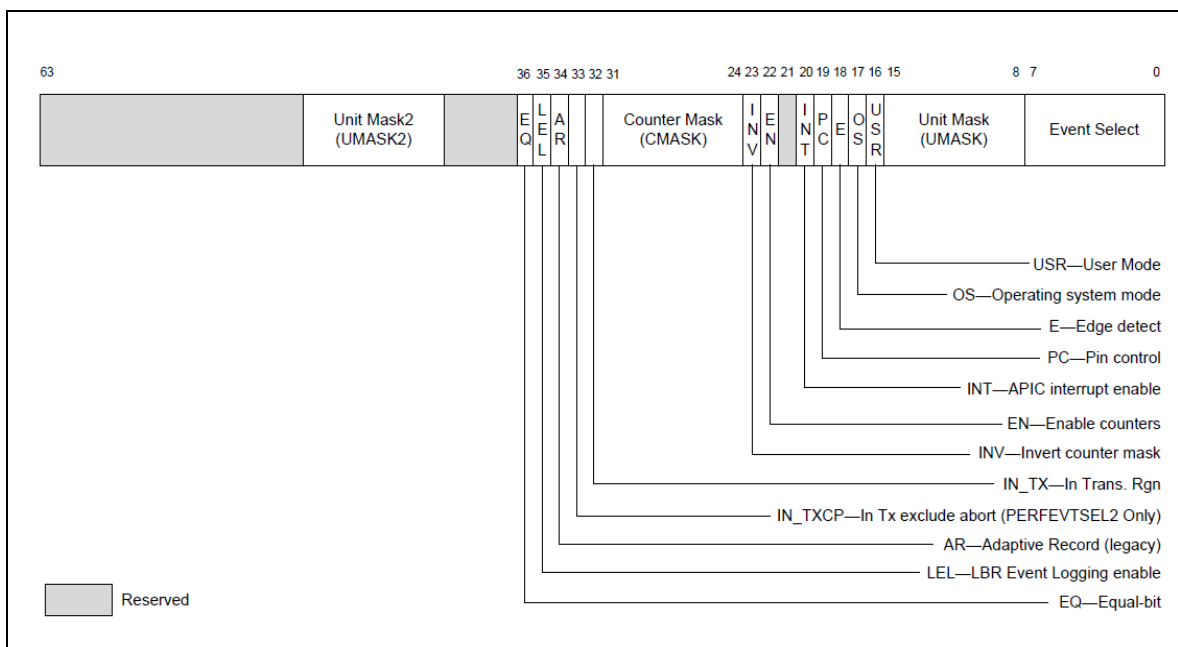


**Figure 22-14. IA32_PMC_GPx_CFG_A MSR (also known as IA32_PERFEVTSELx)
Supporting Architectural Performance Monitoring Version 6**

NOTES

The EQ bit and UMASK2 field are added in Architectural Performance Monitoring Version 6.

The IN_TX and IN_TXCP bits are available only on processors supporting Intel TSX.

The architectural performance monitoring version 6 enhanced layout of the IA32_FIXED_CTR_CTRL MSR is shown in Figure 22-15.

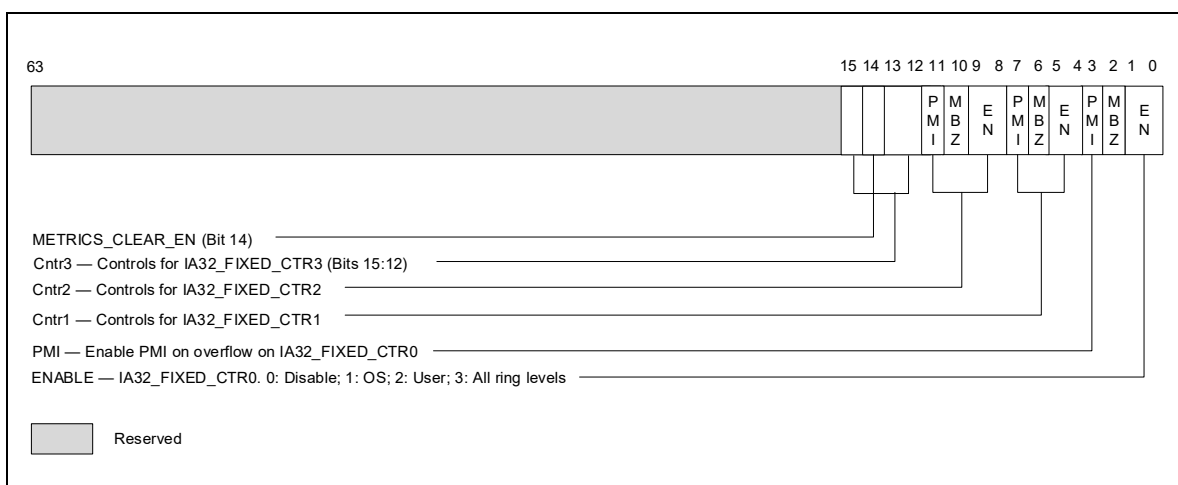


Figure 22-15. IA32_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 6

22.2.6.2 Unit Mask 2

Architectural performance monitoring version 6 introduces a new Unit Mask 2 (UMASK2) field in the IA32_PERFEVTSELx MSRs. It is supported if enumerated by CPUID.23H.00H:EBX[0].

- UMASK2 field (bits 40 through 47): These bits qualify the condition that the selected event logic unit detects. Valid UMASK2 values for each event logic unit are specific to the unit. The new UMASK2 field may also be used in conjunction with UMASK.

22.2.6.3 Equal Flag

Architectural performance monitoring version 6 introduces a new Equal (EQ) flag in the IA32_PERFVTSELx MSRs. It is supported if enumerated by CPUID.23H.00H:EBX[1].

- EQ flag (bit 36): When the EQ flag is set and the INV flag is clear, the comparison evaluates to true if the selected performance monitoring event (the event) is equal to the specified Counter Mask value (CMask). When the EQ flag is set and the INV flag is set, the comparison evaluates to true if the event is less than the CMask value and the event is not zero. Note that if the CMask is zero, the EQ flag is ignored.

22.2.7 Pre-defined Architectural Performance Events

Table Table 22-3. "UMask and Event Select Encodings for Pre-Defined Architectural Performance Events" lists architecturally defined events.

Table 22-3. UMask and Event Select Encodings for Pre-Defined Architectural Performance Events

Bit Position CPUID.0AH:EBX and CPUID.23H.03H:EAX	Event Name	UMask	Event Select
0	UnHalted Core Cycles	00H	3CH
1	Instruction Retired	00H	C0H
2	UnHalted Reference Cycles ¹	01H	3CH
3	LLC Reference	4FH	2EH
4	LLC Misses	41H	2EH
5	Branch Instruction Retired	00H	C4H
6	Branch Misses Retired	00H	C5H
7	Topdown Slots	01H	A4H
8	Topdown Backend Bound	02H	A4H
9	Topdown Bad Speculation	00H	73H
10	Topdown Frontend Bound	01H	9CH
11	Topdown Retiring	02H	C2H
12	LBR Inserts	01H	E4H

NOTES:

1. Implementations prior to the 12th generation Intel® Core™ processor P-cores count at core crystal clock, TSC, or bus clock frequency.

A processor that supports architectural performance monitoring may not support all the predefined architectural performance events (Table Table 22-3. "UMask and Event Select Encodings for Pre-Defined Architectural Performance Events"). The number of architectural events is reported through CPUID.0AH:EAX[31:24], while non-zero bits in CPUID.0AH:EBX indicate any architectural events that are not available.

The behavior of each architectural performance event is expected to be consistent on all processors that support that event. Minor variations between microarchitectures are noted below:

- UnHalted Core Cycles** — Event select 3CH, Umask 00H

This event counts core clock cycles when the clock signal on a specific core is running (not halted). The counter does not advance in the following conditions:

- An ACPI C-state other than C0 for normal operation.

- HLT.
- STPCLK# pin asserted.
- Being throttled by TM1.
- During the frequency switching phase of a performance state transition (see Chapter 17, “Power and Thermal Management”).

The performance counter for this event counts across performance state transitions using different core clock frequencies.

- **Instructions Retired** — Event select C0H, Umask 00H

This event counts the number of instructions at retirement. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. An instruction with a REP prefix counts as one instruction (not per iteration). Faults before the retirement of the last micro-op of a multi-ops instruction are not counted.

This event does not increment under VM-exit conditions. Counters continue counting during hardware interrupts, traps, and inside interrupt handlers.

- **UnHalted Reference Cycles** — Event select 3CH, Umask 01H

This event counts reference clock cycles at a fixed frequency while the clock signal on the core is running. The event counts at a fixed frequency, irrespective of core frequency changes due to performance state transitions. Processors may implement this behavior differently. Current implementations use the core crystal clock, TSC or the bus clock. Because the rate may differ between implementations, software should calibrate it to a time source with known frequency.

- **Last Level Cache References** — Event select 2EH, Umask 4FH

This event counts requests originating from the core that reference a cache line in the last level on-die cache. The event count includes speculation and cache line fills due to the first-level cache hardware prefetcher, but may exclude cache line fills due to other hardware-prefetchers.

Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Last Level Cache Misses** — Event select 2EH, Umask 41H

This event counts each cache miss condition for references to the last level on-die cache. The event count may include speculation and cache line fills due to the first-level cache hardware prefetcher, but may exclude cache line fills due to other hardware-prefetchers.

Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Branch Instructions Retired** — Event select C4H, Umask 00H

This event counts branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction.

- **All Branch Mispredict Retired** — Event select C5H, Umask 00H

This event counts mispredicted branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction in the architectural path of execution and experienced misprediction in the branch prediction hardware.

Branch prediction hardware is implementation-specific across microarchitectures; value comparison to estimate performance differences is not recommended.

- **Topdown Slots** — Event select A4H, Umask 01H

This event counts the total number of available slots for an unhalted logical processor.

The event increments by machine-width of the narrowest pipeline as employed by the Top-down Microarchitecture Analysis method. The count is distributed among unhalted logical processors (hyper-threads) who share the same physical core, in processors that support Intel Hyper-Threading Technology.

Software can use this event as the denominator for the top-level metrics of the Top-down Microarchitecture Analysis method.

NOTE

Programming decisions or software precisions on functionality should not be based on the event values or dependent on the existence of performance monitoring events.

- **Topdown Backend Bound** — Event select A4H, Umask 02H

This event counts a subset of the Topdown Slots event that was not consumed by the backend pipeline due to lack of backend resources, as a result of memory subsystem delays, execution unit limitations, or other conditions.

The count may be distributed among unhalted logical processors that share the same physical core, in processors that support Intel® Hyper-Threading Technology.

Software can use this event as the numerator for the Backend Bound metric (or top-level category) of the Topdown Microarchitecture Analysis method.

Software can also derive the Backend Bound Slots using the formula: Backend Bound Slots = (Total Slots - Bad Speculation Slots - Frontend Bound Slots - Retiring Slots).

- **Topdown Bad Speculation** — Event select 73H, Umask 00H

This event counts a subset of the Topdown Slots event that was wasted due to incorrect speculation as a result of incorrect control-flow or data speculation. Common examples include branch mispredictions and memory ordering clears.

The count may be distributed among impacted logical processors that share the same physical core, for some processors that support Intel Hyper-Threading Technology.

Software can use this event as the numerator for the Bad Speculation metric (or top-level category) of the Topdown Microarchitecture Analysis method.

Software can also derive the Bad Speculation Slots using the formula: Bad Speculation Slots = (Total Slots - Backend Bound Slots - Frontend Bound Slots - Retiring Slots).

- **Topdown Frontend Bound** — Event select 9CH, Umask 01H

This event counts a subset of the Topdown Slots event that had no operation delivered to the backend pipeline due to instruction fetch limitations when the backend could have accepted more operations. Common examples include instruction cache misses and x86 instruction decode limitations.

The count may be distributed among unhalted logical processors that share the same physical core, in processors that support Intel Hyper-Threading Technology.

Software can use this event as the numerator for the Frontend Bound metric (or top-level category) of the Topdown Microarchitecture Analysis method.

- **Topdown Retiring** — Event select C2H, Umask 02H

This event counts a subset of the Topdown Slots event that is utilized by operations that eventually get retired (committed) by the processor pipeline. Usually, this event positively correlates with higher performance as measured by the instructions-per-cycle metric.

Software can use this event as the numerator for the Retiring metric (or top-level category) of the Topdown Microarchitecture Analysis method.

- **LBR Inserts** — Event select E4H, Umask 01H

This event counts when an LBR (Last Branch Record) entry is inserted or removed. Inserted means an actual LBR buffer update has occurred, considering LBR configuration and filtering. An LBR entry is removed when a RET instruction is retired in LBR Call-stack mode.

Software may use this event in usages like profile-guided optimization (PGO) for profiling collections across Intel processors and in virtualized environments.

22.2.8 Full-Width Writes to Performance Counter Registers

The general-purpose performance counter registers IA32_PMCx are writable via WRMSR instruction. However, the value written into IA32_PMCx by WRMSR is the signed extended 64-bit value of the EAX[31:0] input of WRMSR.

A processor that supports full-width writes to the general-purpose performance counters enumerated by CPUID.0AH:EAX[15:8] will set IA32_PERF_CAPABILITIES[13] to enumerate its full-width-write capability. See Figure 22-67.

If IA32_PERF_CAPABILITIES.FW_WRITE[bit 13] = 1, each IA32_PMCi is accompanied by a corresponding alias address starting at 4C1H for IA32_A_PMC0.

The bit width of the performance monitoring counters is specified in CPUID.0AH:EAX[23:16].

If IA32_A_PMCi is present, the 64-bit input value (EDX:EAX) of WRMSR to IA32_A_PMCi will cause IA32_PMCi to be updated by:

```
COUNTERWIDTH = CPUID.0AH:EAX[23:16] bit width of the performance monitoring counter
IA32_PMCi[COUNTERWIDTH-1:32] := EDX[COUNTERWIDTH-33:0];
IA32_PMCi[31:0] := EAX[31:0];
EDX[63:COUNTERWIDTH] are reserved
```

22.2.9 Scalable Enumeration Architecture

The CPUID.23H Architectural Performance Monitoring Extended (ARCH_PERFMON_EXT) leaf provides enhanced enumeration of PMU architectural features. Additionally, the IA32_PERF_CAPABILITIES MSR enhances enumeration for PMU non-architectural features.

NOTE

CPUID.0AH continues to report useful attributes, such as architectural performance monitoring version ID and counter width (# bits).

CPUID.23H enhances previous enumeration of PMU capabilities:

- Supports CPUID sub-leaves to accommodate future PMU extensions.
- Exposes true-view resources per logical processor.
- Introduces a bitmap (true-view) enumeration of general-purpose counters availability.
- A bitmap (true-view) enumeration of fixed-function counters availability.
- A bitmap (true-view) enumeration of architectural performance monitoring events.

Processors that support this enhancement set CPUID.07H.01H:EAX.ARCH_PERFMON_EXT[8].

22.2.9.1 CPUID Sub-leaves

CPUID.23H contains additional architectural PMU capabilities. This leaf supports sub-leaves, providing each distinct PMU feature with an individual sub-leaf for enumerating its details.

The availability of sub-leaves is enumerated via CPUID.23H.00H:EAX. For each bit *n* set in this field, sub-leaf *n* under CPUID.23H is supported.

22.2.9.2 Reporting Per Logical Processor

CPUID.23H provides a true-view of per logical processor PMU capabilities. This leaf reports the actual support of the individual logical processor that the CPUID instruction was executed on; this applies to all sub-leaves.

Software must not make assumptions that CPUID.23H would report any value the same on another logical processor. It is required to read CPUID.23H on every logical processor and program that logical processor only according to the values returned by the CPUID.23H directly executed upon it. It is a requirement of software to compare and determine common features between logical processors if required by iterating over each logical processor's CPUID.23H.

Conversely, CPUID.0AH provides a maximum common set of capabilities across logical processors when a feature is not supported by all logical processors.

NOTE

Locating a PMU feature under CPUID.23H alerts software that the feature may not be supported uniformly across all logical processors.

22.2.9.3 General-Purpose Counters Bitmap

CPUID.23H.01H:EAX reports a bitmap for available general-purpose counters. (CPUID.0AH reports only the total number of general-purpose counters.)

This capability enables a virtual-machine monitor to reserve lower-index counters for its own use, while exposing higher-index counters to guest software. This is especially important should the general-purpose counters not be fully homogeneous.

Software should utilize the new bitmap reporting, including for detecting the number of available general-purpose counters. To facilitate this transition, the number of general-purpose counters in CPUID.0AH will not go beyond eight, even if the processor has support for more than eight general-purpose counters.

Note that general-purpose counters that are exclusively enumerated in CPUID.23H.01H:EAX may not support the legacy MSR address range; see Section 22.2.6.1, “Performance Monitoring MSR Aliasing,” for details.

22.2.9.4 Fixed-Function Counters True-View Bitmap

CPUID.23H.01H:EBX reports a bitmap for available fixed-function counters. (CPUID.0AH reports the common number of contiguous fixed-function counters in addition to a common bitmap of fixed-function counters availability.)¹

This capability enables privileged software to expose per logical processor enumeration of fixed-function counters. This is especially important should the fixed-function counters not be available on all logical processors.

Note that programmable counters that are exclusively enumerated in CPUID.23H.01H:EAX may not support the legacy MSR address range; see Section 22.2.6.1, “Performance Monitoring MSR Aliasing,” for details.

22.2.9.5 Architectural Performance Monitoring Events Bitmap

CPUID.23H.03H:EAX provides a true-view of per logical processor available architectural performance monitoring events. For each bit n set in this field, the processor supports Architectural Performance Monitoring Event of index n (positive polarity).

Conversely, CPUID.0AH:EBX provides a maximum common set of architectural performance monitoring events supported by all logical processors, where if bit n is set, it denotes the processor does not necessarily support Architectural Performance Monitoring Event of index n on all logical processors (negative polarity).

22.2.9.6 TMA Slots Per Cycle

CPUID.23H.00H:ECX[7:0] reports the number of TMA slots per cycle in a true-view per logical-processor fashion.

This number can be multiplied by the number of cycles (from CPU_CLK_UNHALTED.THREAD / CPU_CLK_UNHALTED.CORE or IA32_FIXED_CTR1) to determine the total number of TMA slots.

Because of microarchitectural reasons, some logical processors may be reporting TMA slots per cycle as 0. In such situations, software can use other methods, like programmable events or fixed counters, to understand the performance issues.

1. The valid range of fixed-function counters is 0 through 15.

22.3 PERFORMANCE MONITORING (INTEL® CORE™ PROCESSORS AND INTEL® XEON® PROCESSORS)

22.3.1 Performance Monitoring for Processors Based on Nehalem Microarchitecture

Intel Core i7 processor family¹ supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities. The Intel Core i7 processor family is based on Nehalem microarchitecture, and provides four general-purpose performance counters (IA32_PMC0, IA32_PMC1, IA32_PMC2, IA32_PMC3) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2) in the processor core.

Non-architectural performance monitoring in Intel Core i7 processor family uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>. Non-architectural performance monitoring events fall into two broad categories:

- Performance monitoring events in the processor core: These include many events that are similar to performance monitoring events available to processor based on Intel Core microarchitecture. Additionally, there are several enhancements in the performance monitoring capability for detecting microarchitectural conditions in the processor core or in the interaction of the processor core to the off-core sub-systems in the physical processor package. The off-core sub-systems in the physical processor package is loosely referred to as “uncore”.
- Performance monitoring events in the uncore: The uncore sub-system is shared by more than one processor cores in the physical processor package. It provides additional performance monitoring facility outside of IA32_PMCx and performance monitoring events that are specific to the uncore sub-system.

Architectural and non-architectural performance monitoring events in Intel Core i7 processor family support thread qualification using bit 21 of IA32_PERFEVTSELx MSR.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3.

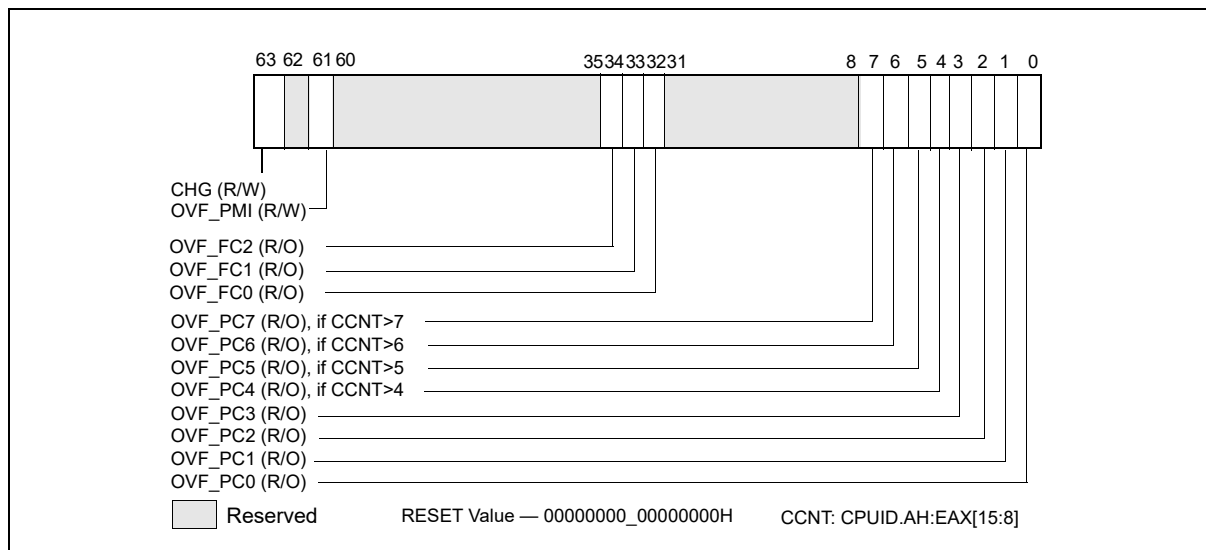


Figure 22-16. IA32_PERF_GLOBAL_STATUS MSR

1. Intel Xeon processor 5500 series and 3400 series are also based on Nehalem microarchitecture; the performance monitoring facilities described in this section generally also apply.

22.3.1.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- Four general purpose performance counters, IA32_PMCx, associated counter configuration MSRs, IA32_PERFVTSELx, and global counter control MSR supporting simplified control of four counters. Each of the four performance counter can support processor event based sampling (PEBS) and thread-qualification of architectural and non-architectural performance events. Width of IA32_PMCx supported by hardware has been increased. The width of counter reported by CPUID.0AH:EAX[23:16] is 48 bits. The PEBS facility in Nehalem microarchitecture has been enhanced to include new data format to capture additional information, such as load latency.
- Load latency sampling facility. Average latency of memory load operation can be sampled using load-latency facility in processors based on Nehalem microarchitecture. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches). This facility is used in conjunction with the PEBS facility.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFVTSELx.

NOTE

The number of counters available to software may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters. CPUID.0AH:EAX[15:8] reports the MSRs available to software; see Section 22.2.1.

22.3.1.1.1 Processor Event Based Sampling (PEBS)

All general-purpose performance counters, IA32_PMCx, can be used for PEBS if the performance event supports PEBS. Software uses IA32_MISC_ENABLE[7] and IA32_MISC_ENABLE[12] to detect whether the performance monitoring facility and PEBS functionality are supported in the processor. The MSR IA32_PEBS_ENABLE provides 4 bits that software must use to enable which IA32_PMCx overflow condition will cause the PEBS record to be captured.

Additionally, the PEBS record is expanded to allow latency information to be captured. The MSR IA32_PEBS_ENABLE provides 4 additional bits that software must use to enable latency data recording in the PEBS record upon the respective IA32_PMCx overflow condition. The layout of IA32_PEBS_ENABLE for processors based on Nehalem microarchitecture is shown in Figure 22-17.

When a counter is enabled to capture machine state (PEBS_EN_PMCx = 1), the processor will write machine state information to a memory buffer specified by software as detailed below. When the counter IA32_PMCx overflows from maximum count to zero, the PEBS hardware is armed.

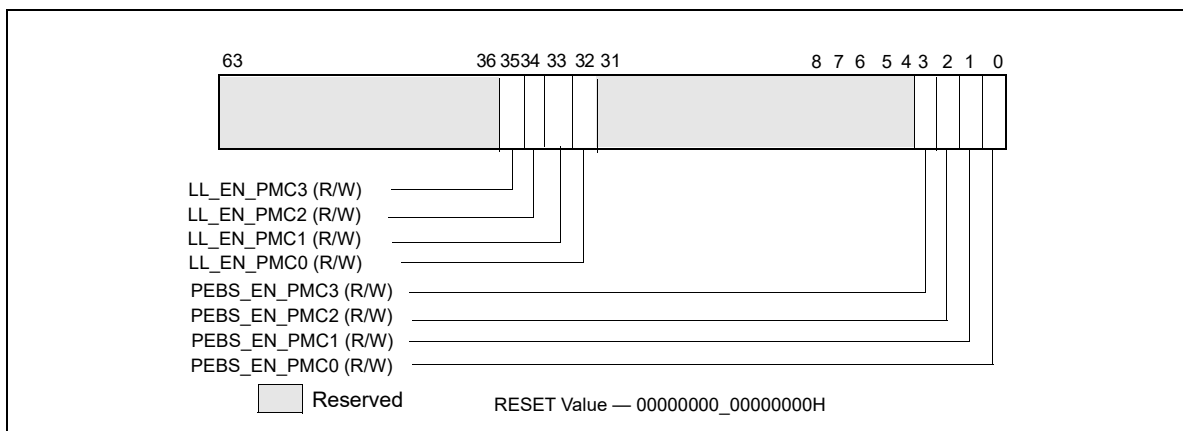


Figure 22-17. Layout of IA32_PEBS_ENABLE MSR

Upon occurrence of the next PEBS event, the PEBS hardware triggers an assist and causes a PEBS record to be written. The format of the PEBS record is indicated by the bit field IA32_PERF_CAPABILITIES[11:8] (see Figure 22-67).

The behavior of PEBS assists is reported by IA32_PERF_CAPABILITIES[6] (see Figure 22-67). The return instruction pointer (RIP) reported in the PEBS record will point to the instruction after (+1) the instruction that causes the PEBS assist. The machine state reported in the PEBS record is the machine state after the instruction that causes the PEBS assist is retired. For instance, if the instructions:

```
mov eax, [eax] ; causes PEBS assist
```

```
nop
```

are executed, the PEBS record will report the address of the nop, and the value of EAX in the PEBS record will show the value read from memory, not the target address of the read operation.

The PEBS record format is shown in Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family", and each field in the PEBS record is 64 bits long. The PEBS record format, along with debug/store area storage format, does not change regardless of IA-32e mode is active or not. CPUID.01H:ECX.DTES64[2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 22-4. PEBS Record Format for Intel Core i7 Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	58H	R9
08H	R/EIP	60H	R10
10H	R/EAX	68H	R11
18H	R/EBX	70H	R12
20H	R/ECX	78H	R13
28H	R/EDX	80H	R14
30H	R/ESI	88H	R15
38H	R/EDI	90H	IA32_PERF_GLOBAL_STATUS
40H	R/EBP	98H	Data Linear Address
48H	R/ESP	A0H	Data Source Encoding
50H	R8	A8H	Latency value (core cycles)

In IA-32e mode, the full 64-bit value is written to the register. If the processor is not operating in IA-32e mode, 32-bit value is written to registers with bits 63:32 zeroed. Registers not defined when the processor is not in IA-32e mode are written to zero.

Bytes AFH:90H are enhancement to the PEBS record format. Support for this enhanced PEBS record format is indicated by IA32_PERF_CAPABILITIES[11:8] encoding of 0001B.

The value written to bytes 97H:90H is the state of the IA32_PERF_GLOBAL_STATUS register before the PEBS assist occurred. This value is written so software can determine which counters overflowed when this PEBS record was written. Note that this field indicates the overflow status for all counters, regardless of whether they were programmed for PEBS or not.

Programming PEBS Facility

Only a subset of non-architectural performance events in the processor support PEBS. The subset of precise events are listed in Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture". In addition to using IA32_PERFEVTSELx to specify event unit/mask settings and setting the EN_PMCx bit in the IA32_PEBS_ENABLE register for the respective counter, the software must also initialize the DS_BUFFER_MANAGEMENT_AREA data structure in memory to support capturing PEBS records for precise events.

The recording of PEBS records may not operate properly if accesses to the linear addresses in the DS buffer management area or in the PEBS buffer (see below) cause page faults, VM exits, or the setting of accessed or dirty flags in the paging structures (ordinary or EPT). For that reason, system software should establish paging structures (both ordinary and EPT) to prevent such occurrences. Implications of this may be that an operating system should allocate this memory from a non-paged pool and that system software cannot do "lazy" page-table entry propagation for these pages. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

The beginning linear address of the DS_BUFFER_MANAGEMENT_AREA data structure must be programmed into the IA32_DS_AREA register. The layout of the DS_BUFFER_MANAGEMENT_AREA is shown in Figure 22-18.

- **PEBS Buffer Base:** This field is programmed with the linear address of the first byte of the PEBS buffer allocated by software. The processor reads this field to determine the base address of the PEBS buffer.
- **PEBS Index:** This field is initially programmed with the same value as the PEBS Buffer Base field, or the beginning linear address of the PEBS buffer. The processor reads this field to determine the location of the next PEBS record to write to. After a PEBS record has been written, the processor also updates this field with the address of the next PEBS record to be written. The figure above illustrates the state of PEBS Index after the first PEBS record is written.
- **PEBS Absolute Maximum:** This field represents the absolute address of the maximum length of the allocated PEBS buffer plus the starting address of the PEBS buffer. The processor will not write any PEBS record beyond the end of PEBS buffer, when **PEBS Index** equals **PEBS Absolute Maximum**. No signaling is generated when PEBS buffer is full. Software must reset the **PEBS Index** field to the beginning of the PEBS buffer address to continue capturing PEBS records.

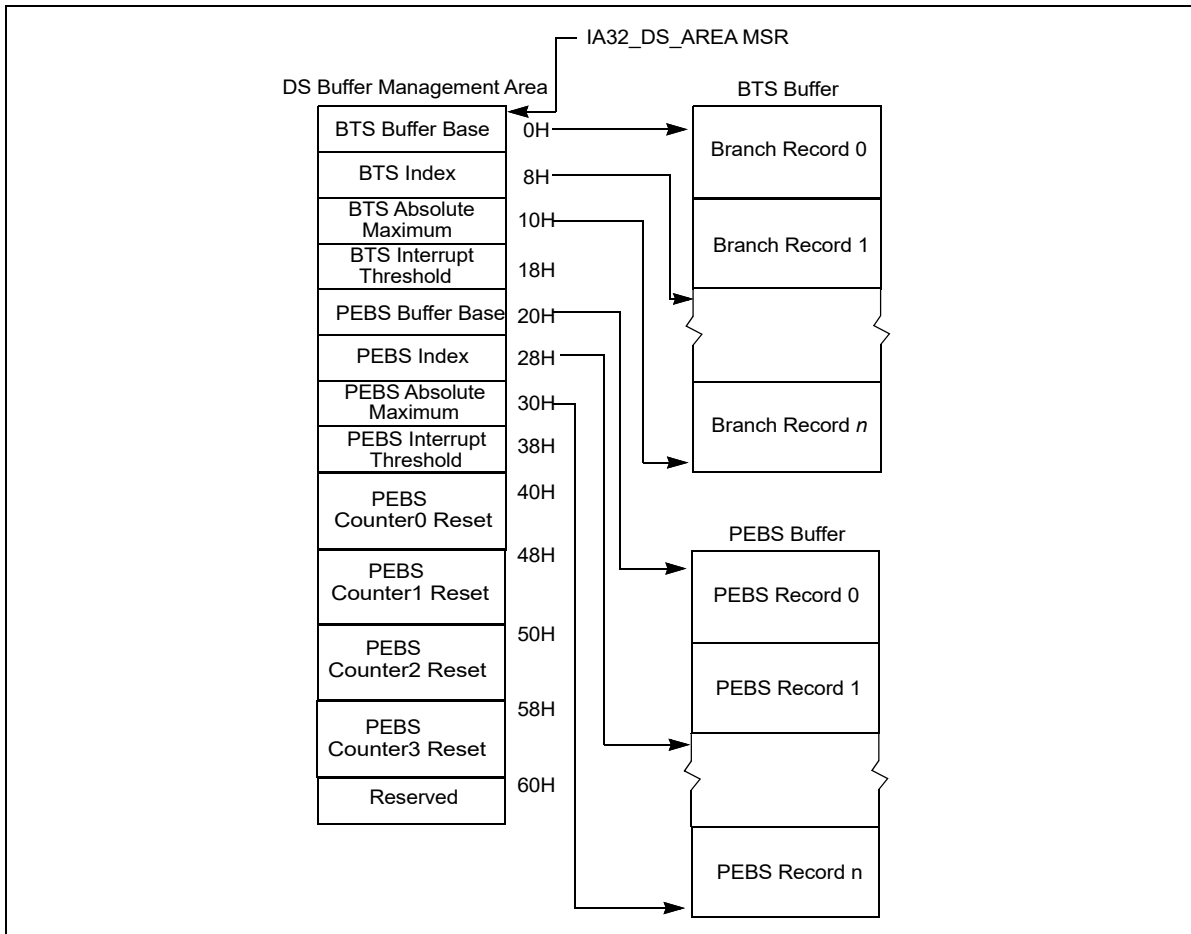


Figure 22-18. PEBS Programming Environment

- PEBS Interrupt Threshold:** This field specifies the threshold value to trigger a performance interrupt and notify software that the PEBS buffer is nearly full. This field is programmed with the linear address of the first byte of the PEBS record within the PEBS buffer that represents the threshold value of this field. After the processor writes a PEBS record and updates **PEBS Index**, if the **PEBS Index** reaches the threshold value of this field, the processor will generate a performance interrupt. This is the same interrupt that is generated by a performance counter overflow, as programmed in the Performance Monitoring Counters vector in the Local Vector Table of the Local APIC. When a performance interrupt due to PEBS buffer full is generated, the IA32_PERF_GLOBAL_STATUS.PEBS_Ovf bit will be set.
- PEBS CounterX Reset:** This field allows software to set up PEBS counter overflow condition to occur at a rate useful for profiling workload, thereby generating multiple PEBS records to facilitate characterizing the profile the execution of test code. After each PEBS record is written, the processor checks each counter to see if it overflowed and was enabled for PEBS (the corresponding bit in IA32_PEBS_ENABLED was set). If these conditions are met, then the reset value for each overflowed counter is loaded from the DS Buffer Management Area. For example, if counter IA32_PMC0 caused a PEBS record to be written, then the value of "PEBS Counter 0 Reset" would be written to counter IA32_PMC0. If a counter is not enabled for PEBS, its value will not be modified by the PEBS assist.

Performance Counter Prioritization

Performance monitoring interrupts are triggered by a counter transitioning from maximum count to zero (assuming IA32_PerfEvtSelX.INT is set). This same transition will cause PEBS hardware to arm, but not trigger. PEBS hardware triggers upon detection of the first PEBS event after the PEBS hardware has been armed (a 0 to 1 transition of the counter). At this point, a PEBS assist will be undertaken by the processor.

Performance counters (fixed and general-purpose) are prioritized in index order. That is, counter IA32_PMC0 takes precedence over all other counters. Counter IA32_PMC1 takes precedence over counters IA32_PMC2 and IA32_PMC3, and so on. This means that if simultaneous overflows or PEBS assists occur, the appropriate action will be taken for the highest priority performance counter. For example, if IA32_PMC1 cause an overflow interrupt and IA32_PMC2 causes an PEBS assist simultaneously, then the overflow interrupt will be serviced first.

The PEBS threshold interrupt is triggered by the PEBS assist, and is by definition prioritized lower than the PEBS assist. Hardware will not generate separate interrupts for each counter that simultaneously overflows. General-purpose performance counters are prioritized over fixed counters.

If a counter is programmed with a precise (PEBS-enabled) event and programmed to generate a counter overflow interrupt, the PEBS assist is serviced before the counter overflow interrupt is serviced. If in addition the PEBS interrupt threshold is met, the

threshold interrupt is generated after the PEBS assist completes, followed by the counter overflow interrupt (two separate interrupts are generated).

Uncore counters may be programmed to interrupt one or more processor cores (see Section 22.3.1.2). It is possible for interrupts posted from the uncore facility to occur coincident with counter overflow interrupts from the processor core. Software must check core and uncore status registers to determine the exact origin of counter overflow interrupts.

22.3.1.1.2 Load Latency Performance Monitoring Facility

The load latency facility provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family". This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).

To use this feature software must assure:

- One of the IA32_PERFEVTSELx MSR is programmed to specify the event unit MEM_INST_RETIRED, and the LATENCY_ABOVE_THRESHOLD event mask must be specified (IA32_PerfEvtSelX[15:0] = 100H). The corresponding counter IA32_PMCx will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in a MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with latencies greater than this value are eligible for counting and latency data reporting. The minimum value that may be programmed in this register is 3 (the minimum detectable load latency is 4 core clock cycles).
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register. This means that both the PEBS_EN_CTRX and LL_EN_CTRX bits must be set for the counter(s) of interest. For example, to enable load latency on counter IA32_PMC0, the IA32_PEBS_ENABLE register must be programmed with the 64-bit value 00000001_00000001H.

When the load-latency facility is enabled, load operations are randomly selected by hardware and tagged to carry information related to data source locality and latency. Latency and data source information of tagged loads are updated internally.

When a PEBS assist occurs, the last update of latency and data source information are captured by the assist and written as part of the PEBS record. The PEBS sample after value (SAV), specified in PEBS CounterX Reset, operates orthogonally to the tagging mechanism. Loads are randomly tagged to collect latency data. The SAV controls the number of tagged loads with latency information that will be written into the PEBS record field by the PEBS assists. The load latency data written to the PEBS record will be for the last tagged load operation which retired just before the PEBS assist was invoked.

The load-latency information written into a PEBS record (see Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family", bytes AFH:98H) consists of:

- **Data Linear Address:** This is the linear address of the target of the load operation.

- **Latency Value:** This is the elapsed cycles of the tagged load operation between dispatch to GO, measured in processor core clock domain.
- **Data Source:** The encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 22-5. "Data Source Encoding for Load Latency Record". In the descriptions, local memory refers to system memory physically attached to a processor package, and remote memory refers to system memory physically attached to another processor package.

Table 22-5. Data Source Encoding for Load Latency Record

Encoding	Description
00H	Unknown L3 cache miss.
01H	Minimal latency core cache hit. This request was satisfied by the L1 data cache.
02H	Pending core cache HIT. Outstanding core cache miss to same cache-line address was already underway.
03H	This data request was satisfied by the L2.
04H	L3 HIT. Local or Remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
05H	L3 HIT. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found. (clean).
06H	L3 HIT. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found.
07H ¹	Reserved/LLC Snoop HitM. Local or Remote home requests that hit the last level cache and were serviced by another core with a cross core snoop where modified copies were found.
08H	Reserved/L3 MISS. Local homed requests that missed the L3 cache and were serviced by forwarded data following a cross package snoop where no modified copies were found. (Remote home requests are not counted).
09H	Reserved
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	L3 MISS. Remote home requests that missed the L3 cache and were serviced by remote DRAM (go to shared state).
0CH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to exclusive state).
0DH	L3 MISS. Remote home requests that missed the L3 cache and were serviced by remote DRAM (go to exclusive state).
0EH	I/O, Request of input/output operation.
0FH	The request was to uncacheable memory.

NOTES:

1. Bit 7 is supported only for processors with a CPUID DisplayFamily_DisplayModel signature of 06_2A, and 06_2E; otherwise it is reserved.

The layout of MSR_PEBS_LD_LAT_THRESHOLD is shown in Figure 22-19.

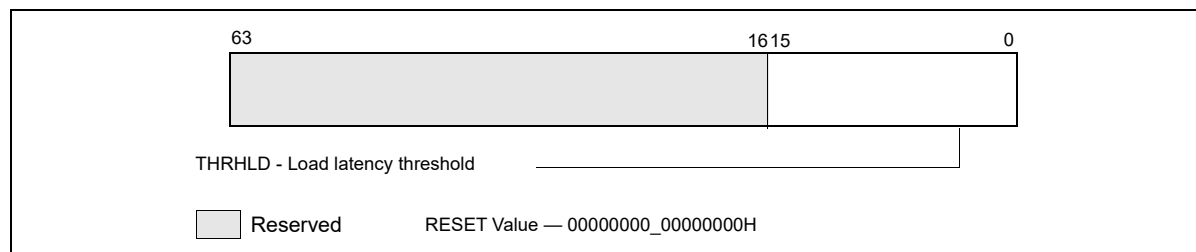


Figure 22-19. Layout of MSR_PEBS_LD_LAT MSR

Bits 15:0 specifies the threshold load latency in core clock cycles. Performance events with latencies greater than this value are counted in IA32_PMCx and their latency information is reported in the PEBS record. Otherwise, they are ignored. The minimum value that may be programmed in this field is 3.

22.3.1.1.3 Off-core Response Performance Monitoring in the Processor Core

Programming a performance event using the off-core response facility can choose any of the four IA32_PERFEVTSELx MSR with specific event codes and predefine mask bit value. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_0. There is only one off-core response configuration MSR. Table Table 22-6. "Off-Core Response Event Encoding" lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 22-6. Off-Core Response Event Encoding

Event code in IA32_PERFEVTSELx	Mask Value in IA32_PERFEVTSELx	Required Off-core Response MSR
B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)

The layout of MSR_OFFCORE_RSP_0 is shown in Figure 22-20. Bits 7:0 specifies the request type of a transaction request to the uncore. Bits 15:8 specifies the response of the uncore subsystem.

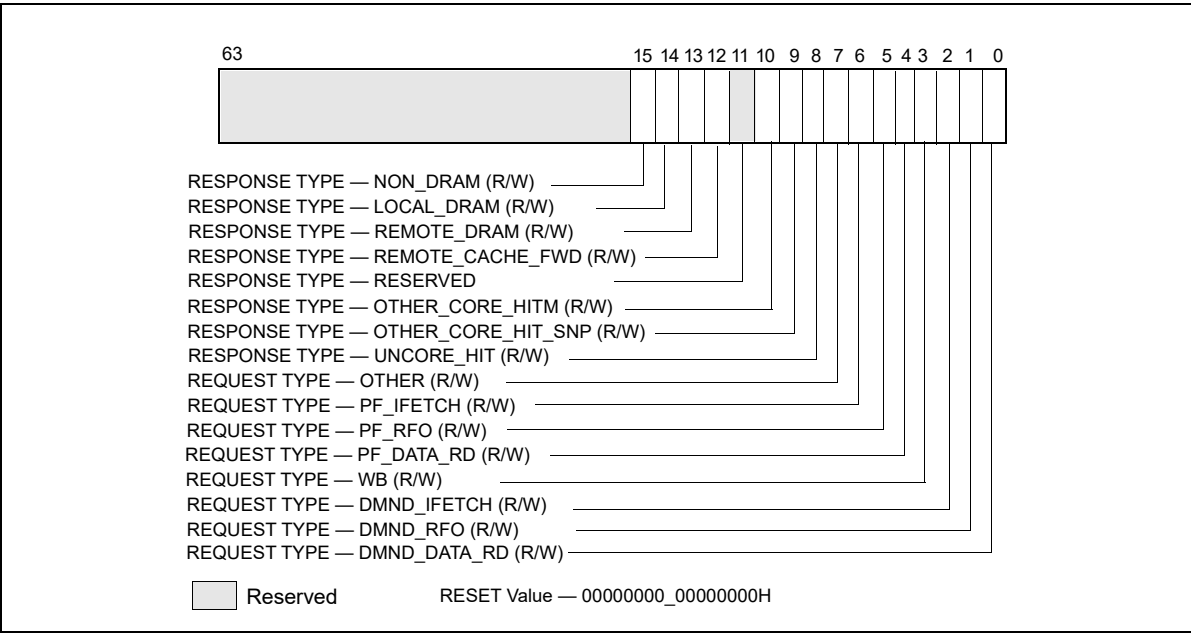


Figure 22-20. Layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 to Configure Off-core Response Events

Table 22-7. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.

Table 22-7. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition (Contd.)

Bit Name	Offset	Description
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
OTHER	7	Counts one of the following transaction types, including L3 invalidate, I/O, full or partial writes, WC or non-temporal stores, CLFLUSH, Fences, lock, unlock, split lock.
UNCORE_HIT	8	L3 Hit: local or remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
OTHER_CORE_HIT_SNP	9	L3 Hit: local or remote home requests that hit L3 cache in the uncore and was serviced by another core with a cross core snoop where no modified copies were found (clean).
OTHER_CORE_HIT_TM	10	L3 Hit: local or remote home requests that hit L3 cache in the uncore and was serviced by another core with a cross core snoop where modified copies were found (HITM).
Reserved	11	Reserved
REMOTE_CACHE_FWD	12	L3 Miss: local homed requests that missed the L3 cache and was serviced by forwarded data following a cross package snoop where no modified copies found. (Remote home requests are not counted)
REMOTE_DRAM	13	L3 Miss: remote home requests that missed the L3 cache and were serviced by remote DRAM.
LOCAL_DRAM	14	L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM.
NON_DRAM	15	Non-DRAM requests that were serviced by IOH.

22.3.1.2 Performance Monitoring Facility in the Uncore

The “uncore” in Nehalem microarchitecture refers to subsystems in the physical processor package that are shared by multiple processor cores. Some of the sub-systems in the uncore include the L3 cache, Intel QuickPath Interconnect link logic, and integrated memory controller. The performance monitoring facilities inside the uncore operates in the same clock domain as the uncore (U-clock domain), which is usually different from the processor core clock domain. The uncore performance monitoring facilities described in this section apply to Intel Xeon processor 5500 series and processors with the following CUID signatures: 06_1AH, 06_1EH, 06_1FH (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4). An overview of the uncore performance monitoring facilities is described separately.

The performance monitoring facilities available in the U-clock domain consist of:

- Eight General-purpose counters (MSR_UNCORE_PerfCntr0 through MSR_UNCORE_PerfCntr7). The counters are 48 bits wide. Each counter is associated with a configuration MSR, MSR_UNCORE_PerfEvtSelx, to specify event code, event mask and other event qualification fields. A set of global uncore performance counter enabling/overflow/status control MSRs are also provided for software.
- Performance monitoring in the uncore provides an address/opcode match MSR that provides event qualification control based on address value or QPI command opcode.
- One fixed-function counter, MSR_UNCORE_FixedCntr0. The fixed-function uncore counter increments at the rate of the U-clock when enabled.

The frequency of the uncore clock domain can be determined from the uncore clock ratio which is available in the PCI configuration space register at offset C0H under device number 0 and Function 0.

22.3.1.2.1 Uncore Performance Monitoring Management Facility

MSR_UNCORE_PERF_GLOBAL_CTRL provides bit fields to enable/disable general-purpose and fixed-function counters in the uncore. Figure 22-21 shows the layout of MSR_UNCORE_PERF_GLOBAL_CTRL for an uncore that is shared by four processor cores in a physical package.

- EN_PCn (bit n, n = 0, 7): When set, enables counting for the general-purpose uncore counter MSR_UNCORE_PerfCntr n.

- EN_FC0 (bit 32): When set, enables counting for the fixed-function uncore counter MSR_UNCORE_FixedCnt0.
- EN_PMI_COREn (bit n, n = 0, 3 if four cores are present): When set, processor core n is programmed to receive an interrupt signal from any interrupt enabled uncore counter. PMI delivery due to an uncore counter overflow is enabled by setting IA32_DEBUGCTL.Offcore_PMI_EN to 1.
- PMI_FRZ (bit 63): When set, all U-clock uncore counters are disabled when any one of them signals a performance interrupt. Software must explicitly re-enable the counter by setting the enable bits in MSR_UNCORE_PERF_GLOBAL_CTRL upon exit from the ISR.

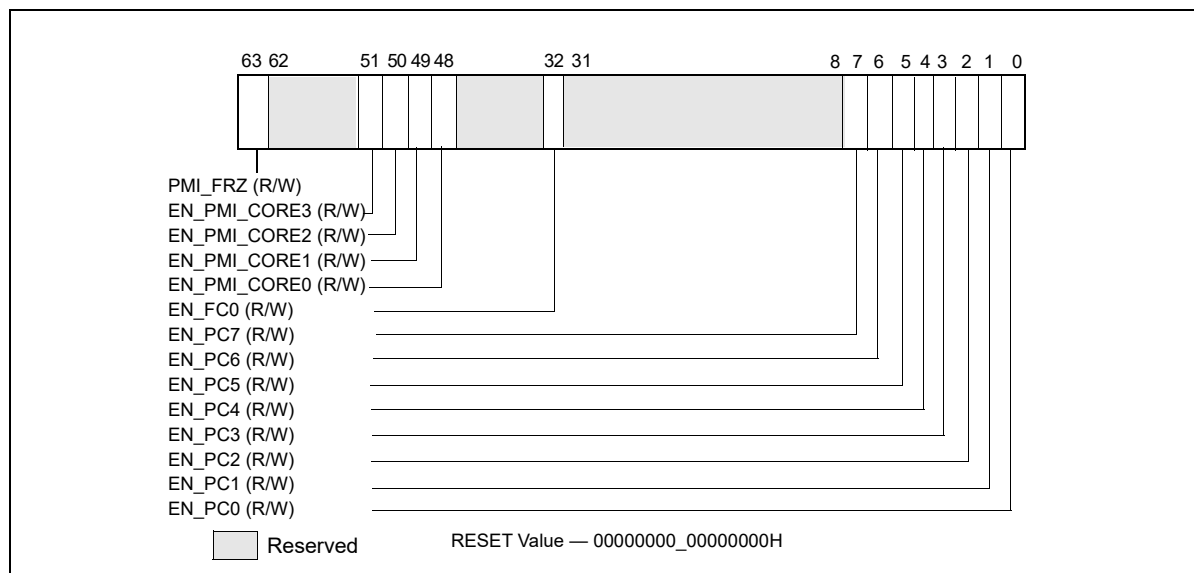


Figure 22-21. Layout of MSR_UNCORE_PERF_GLOBAL_CTRL MSR

MSR_UNCORE_PERF_GLOBAL_STATUS provides overflow status of the U-clock performance counters in the uncore. This is a read-only register. If an overflow status bit is set the corresponding counter has overflowed. The register provides a condition change bit (bit 63) which can be quickly checked by software to determine if a significant change has occurred since the last time the condition change status was cleared. Figure 22-22 shows the layout of MSR_UNCORE_PERF_GLOBAL_STATUS.

- OVF_PCN (bit n, n = 0, 7): When set, indicates general-purpose uncore counter MSR_UNCORE_PerfCntr n has overflowed.
- OVF_FC0 (bit 32): When set, indicates the fixed-function uncore counter MSR_UNCORE_FixedCntr0 has overflowed.
- OVF_PMI (bit 61): When set indicates that an uncore counter overflowed and generated an interrupt request.
- CHG (bit 63): When set indicates that at least one status bit in MSR_UNCORE_PERF_GLOBAL_STATUS register has changed state.

MSR_UNCORE_PERF_GLOBAL_OVF_CTRL allows software to clear the status bits in the UNCORE_PERF_GLOBAL_STATUS register. This is a write-only register, and individual status bits in the global status register are cleared by writing a binary one to the corresponding bit in this register. Writing zero to any bit position in this register has no effect on the uncore PMU hardware.

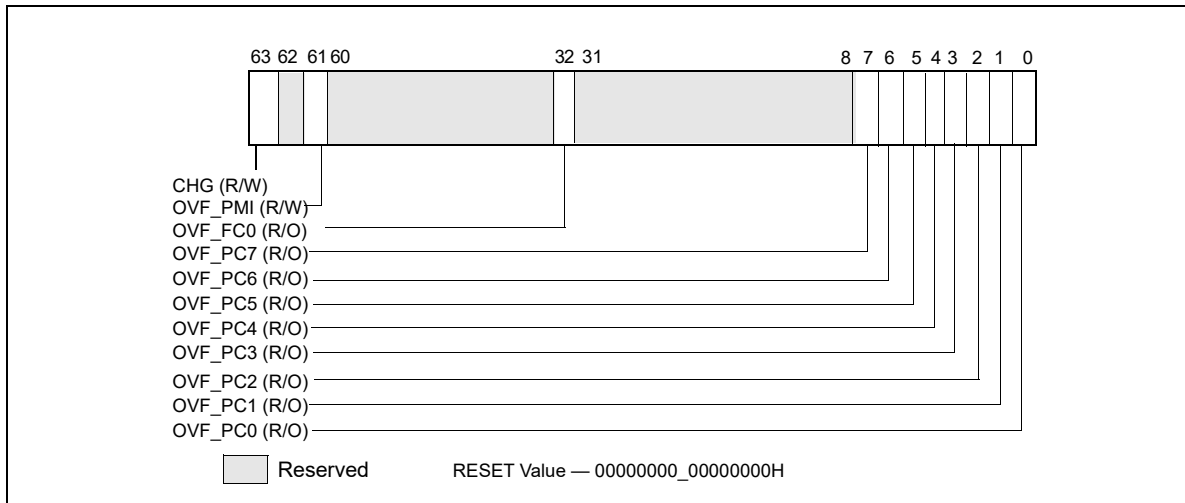


Figure 22-22. Layout of MSR_UNCORE_PERF_GLOBAL_STATUS MSR

Figure 22-23 shows the layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL.

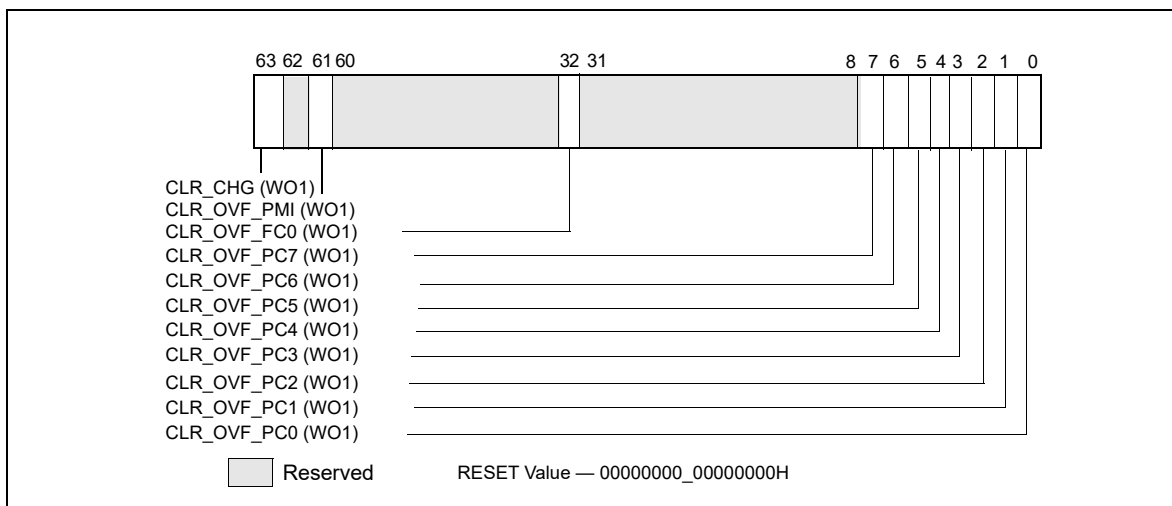


Figure 22-23. Layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL MSR

- CLR_OVF_PCn (bit n, n = 0, 7): Set this bit to clear the overflow status for general-purpose uncore counter MSR_UNCORE_PerfCntr n. Writing a value other than 1 is ignored.
- CLR_OVF_FC0 (bit 32): Set this bit to clear the overflow status for the fixed-function uncore counter MSR_UNCORE_FixedCntr0. Writing a value other than 1 is ignored.
- CLR_OVF_PMI (bit 61): Set this bit to clear the OVF_PMI flag in MSR_UNCORE_PERF_GLOBAL_STATUS. Writing a value other than 1 is ignored.
- CLR_CHG (bit 63): Set this bit to clear the CHG flag in MSR_UNCORE_PERF_GLOBAL_STATUS register. Writing a value other than 1 is ignored.

22.3.1.2.2 Uncore Performance Event Configuration Facility

MSR_UNCORE_PerfEvtSel0 through MSR_UNCORE_PerfEvtSel7 are used to select performance event and configure the counting behavior of the respective uncore performance counter. Each uncore PerfEvtSel MSR is paired with an uncore performance counter. Each uncore counter must be locally configured using the corre-

sponding MSR_UNCORE_PerfEvtSelx and counting must be enabled using the respective EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL. Figure 22-24 shows the layout of MSR_UNCORE_PERFEVTSELx.

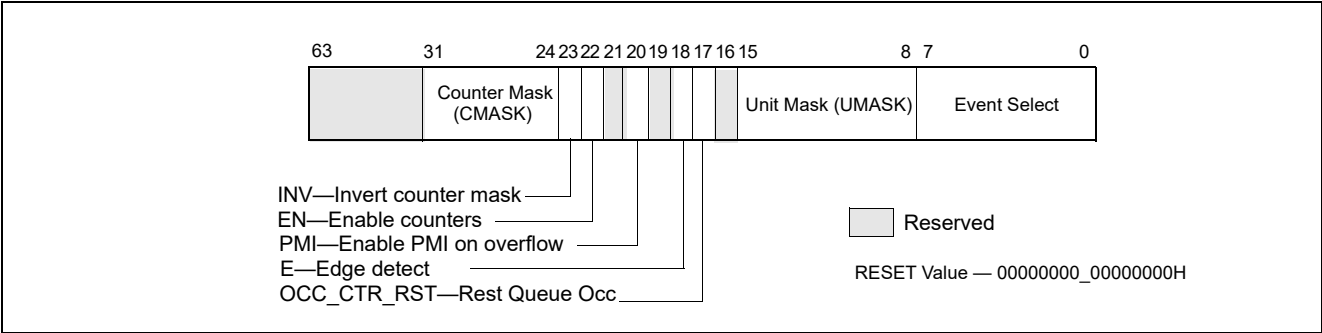


Figure 22-24. Layout of MSR_UNCORE_PERFEVTSELx MSRs

- Event Select (bits 7:0): Selects the event logic unit used to detect uncore events.
- Unit Mask (bits 15:8) : Condition qualifiers for the event selection logic specified in the Event Select field.
- OCC_CTR_RST (bit17): When set causes the queue occupancy counter associated with this event to be cleared (zeroed). Writing a zero to this bit will be ignored. It will always read as a zero.
- Edge Detect (bit 18): When set causes the counter to increment when a deasserted to asserted transition occurs for the conditions that can be expressed by any of the fields in this register.
- PMI (bit 20): When set, the uncore will generate an interrupt request when this counter overflowed. This request will be routed to the logical processors as enabled in the PMI enable bits (EN_PMI_COREx) in the register MSR_UNCORE_PERF_GLOBAL_CTRL.
- EN (bit 22): When clear, this counter is locally disabled. When set, this counter is locally enabled and counting starts when the corresponding EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL is set.
- INV (bit 23): When clear, the Counter Mask field is interpreted as greater than or equal to. When set, the Counter Mask field is interpreted as less than.
- Counter Mask (bits 31:24): When this field is clear, it has no effect on counting. When set to a value other than zero, the logical processor compares this field to the event counts on each core clock cycle. If INV is clear and the event counts are greater than or equal to this field, the counter is incremented by one. If INV is set and the event counts are less than this field, the counter is incremented by one. Otherwise the counter is not incremented.

Figure 22-25 shows the layout of MSR_UNCORE_FIXED_CTR_CTRL.

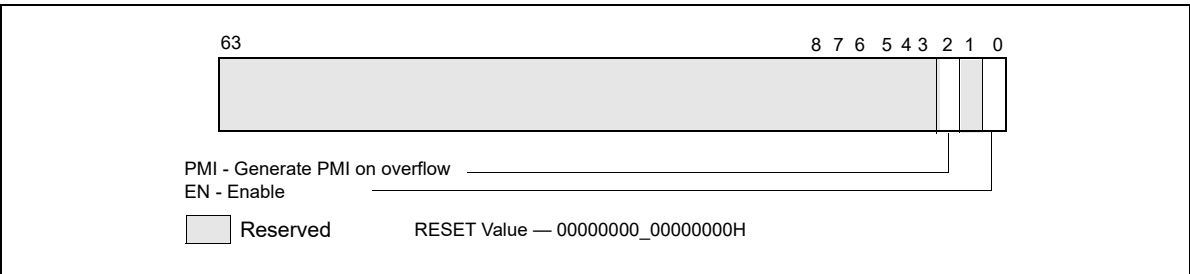


Figure 22-25. Layout of MSR_UNCORE_FIXED_CTR_CTRL MSR

- EN (bit 0): When clear, the uncore fixed-function counter is locally disabled. When set, it is locally enabled and counting starts when the EN_FC0 bit in MSR_UNCORE_PERF_GLOBAL_CTRL is set.
- PMI (bit 2): When set, the uncore will generate an interrupt request when the uncore fixed-function counter overflowed. This request will be routed to the logical processors as enabled in the PMI enable bits (EN_PMI_COREx) in the register MSR_UNCORE_PERF_GLOBAL_CTRL.

Both the general-purpose counters (MSR_UNCORE_PerfCntr) and the fixed-function counter (MSR_UNCORE_FixedCntr0) are 48 bits wide. They support both counting and interrupt based sampling usages. The event logic unit can filter event counts to specific regions of code or transaction types incoming to the home node logic.

22.3.1.2.3 Uncore Address/Opcode Match MSR

The Event Select field [7:0] of MSR_UNCORE_PERFEVTSELx is used to select different uncore event logic unit. When the event "ADDR_OPCODE_MATCH" is selected in the Event Select field, software can filter uncore performance events according to transaction address and certain transaction responses. The address filter and transaction response filtering requires the use of MSR_UNCORE_ADDR_OPCODE_MATCH register. The layout is shown in Figure 22-26.

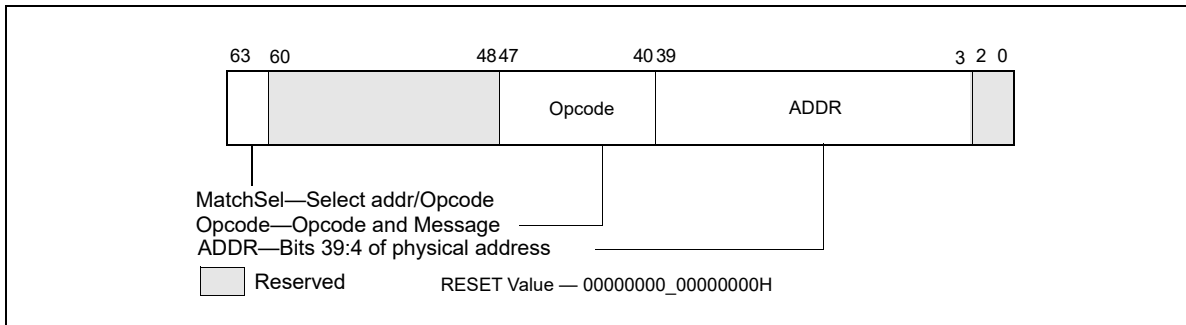


Figure 22-26. Layout of MSR_UNCORE_ADDR_OPCODE_MATCH MSR

- Addr (bits 39:3): The physical address to match if "MatchSel" field is set to select address match. The uncore performance counter will increment if the lowest 40-bit incoming physical address (excluding bits 2:0) for a transaction request matches bits 39:3.
- Opcode (bits 47:40) : Bits 47:40 allow software to filter uncore transactions based on QPI link message class/packed header opcode. These bits are consists two sub-fields:
 - Bits 43:40 specify the QPI packet header opcode.
 - Bits 47:44 specify the QPI message classes.

Table Table 22-8. "Opcode Field Encoding for MSR_UNCORE_ADDR_OPCODE_MATCH" lists the encodings supported in the opcode field.

Table 22-8. Opcode Field Encoding for MSR_UNCORE_ADDR_OPCODE_MATCH

Opcode [43:40]	QPI Message Class		
	Home Request [47:44] = 0000B	Snoop Response [47:44] = 0001B	Data Response [47:44] = 1110B
		1	
DMND_IFETCH	2	2	
WB	3	3	
PF_DATA_RD	4	4	
PF_RFO	5	5	
PF_IFETCH	6	6	
OTHER	7	7	
NON_DRAM	15	15	

- MatchSel (bits 63:61): Software specifies the match criteria according to the following encoding:
 - 000B: Disable addr_opcode match hardware.
 - 100B: Count if only the address field matches.
 - 010B: Count if only the opcode field matches.
 - 110B: Count if either opcode field matches or the address field matches.
 - 001B: Count only if both opcode and address field match.
 - Other encoding are reserved.

22.3.1.3 Intel® Xeon® Processor 7500 Series Performance Monitoring Facility

The performance monitoring facility in the processor core of Intel® Xeon® processor 7500 series are the same as those supported in Intel Xeon processor 5500 series. The uncore subsystem in Intel Xeon processor 7500 series are significantly different The uncore performance monitoring facility consist of many distributed units associated with individual logic control units (referred to as boxes) within the uncore subsystem. A high level block diagram of the various box units of the uncore is shown in Figure 22-27.

Uncore PMUs are programmed via MSR interfaces. Each of the distributed uncore PMU units have several general-purpose counters. Each counter requires an associated event select MSR, and may require additional MSRs to configure sub-event conditions. The uncore PMU MSRs associated with each box can be categorized based on its functional scope: per-counter, per-box, or global across the uncore. The number counters available in each box type are different. Each box generally provides a set of MSRs to enable/disable, check status/overflow of multiple counters within each box.

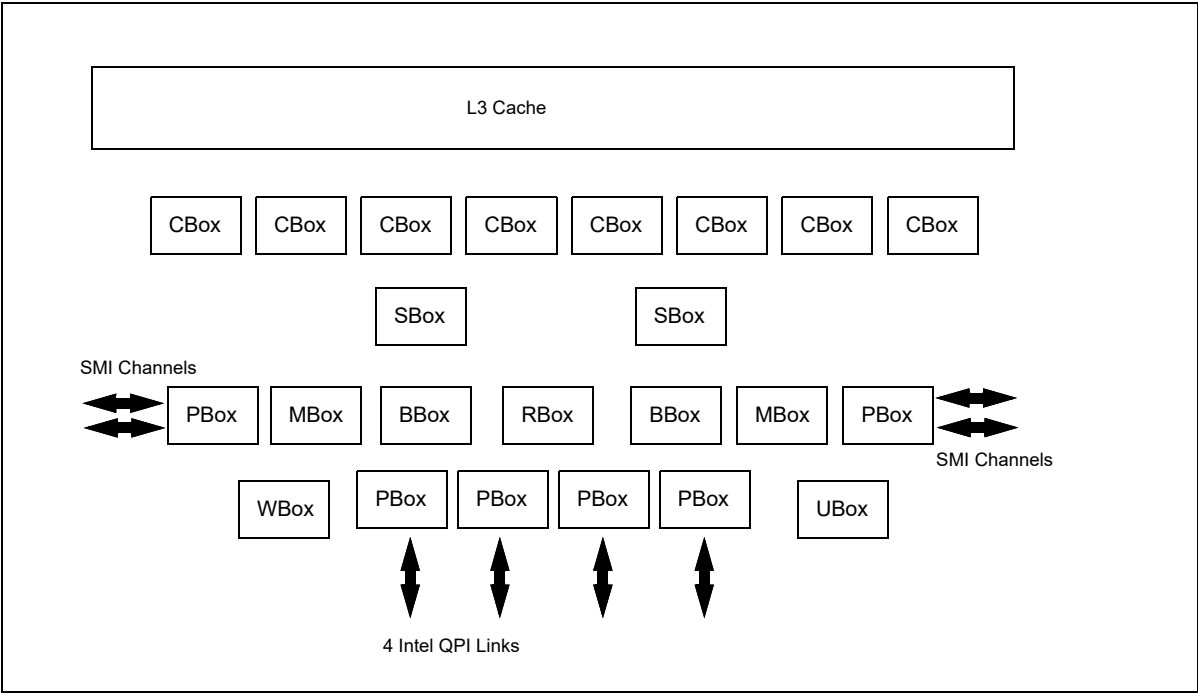


Figure 22-27. Distributed Units of the Uncore of Intel® Xeon® Processor 7500 Series

Table Table 22-9. "Uncore PMU MSR Summary" summarizes the number MSRs for uncore PMU for each box.

Table 22-9. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

The W-Box provides 4 general-purpose counters, each requiring an event select configuration MSR, similar to the general-purpose counters in other boxes. There is also a fixed-function counter that increments clockticks in the uncore clock domain.

For C,S,B,M,R, and W boxes, each box provides an MSR to enable/disable counting, configuring PMI of multiple counters within the same box, this is somewhat similar to the “global control” programming interface, IA32_PERF_GLOBAL_CTRL, offered in the core PMU. Similarly status information and counter overflow control for multiple counters within the same box are also provided in C,S,B,M,R, and W boxes.

In the U-Box, MSR_U_PMON_GLOBAL_CTL provides overall uncore PMU enable/disable and PMI configuration control. The scope of status information in the U-box is at per-box granularity, in contrast to the per-box status information MSR (in the C,S,B,M,R, and W boxes) providing status information of individual counter overflow. The difference in scope also apply to the overflow control MSR in the U-Box versus those in the other Boxes.

The individual MSRs that provide uncore PMU interfaces are listed in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, Table Table 2-17. “Additional MSRs in the Intel® Xeon® Processor 7500 Series” under the general naming style of MSR_%box#%_PMON_%scope_function%, where %box#% designates the type of box and zero-based index if there are more 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”.

22.3.2 Performance Monitoring for Processors Based on Westmere Microarchitecture

All of the performance monitoring programming interfaces (architectural and non-architectural core PMU facilities, and uncore PMU) described in Section 22.6.3 also apply to processors based on Westmere microarchitecture.

Table Table 22-6. "Off-Core Response Event Encoding" describes a non-architectural performance monitoring event (event code 0B7H) and associated MSR_OFFCORE_RSP_0 (address 1A6H) in the core PMU. This event and a second functionally equivalent offcore response event using event code 0BBH and MSR_OFFCORE_RSP_1 (address 1A7H) are supported in processors based on Westmere microarchitecture. The event code and event mask definitions of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

The load latency facility is the same as described in Section 22.3.1.1.2, but added enhancement to provide more information in the data source encoding field of each load latency record. The additional information relates to STLB_MISS and LOCK, see Table Table 22-14. "Layout of Data Source Field of Load Latency Record".

22.3.3 Intel® Xeon® Processor E7 Family Performance Monitoring Facility

The performance monitoring facility in the processor core of the Intel® Xeon® processor E7 family is the same as those supported in the Intel Xeon processor 5600 series¹. The uncore subsystem in the Intel Xeon processor E7 family is similar to those of the Intel Xeon processor 7500 series. The high level construction of the uncore subsystem is similar to that shown in Figure 22-27, with the additional capability that up to 10 C-Box units are supported.

Table Table 22-10. "Uncore PMU MSR Summary for Intel® Xeon® Processor E7 Family" summarizes the number MSRs for uncore PMU for each box.

Table 22-10. Uncore PMU MSR Summary for Intel® Xeon® Processor E7 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	10	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E7 family is available in the "Intel® Xeon® Processor E7 Uncore Performance Monitoring Programming Reference Manual".

22.3.4 Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture

Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel® Xeon® processor E3-1200 family are based on Sandy Bridge microarchitecture; this section describes the performance monitoring facilities provided in the processor core. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 22.2.3.

The core PMU's capability is similar to those described in Section 22.3.1.1 and Section 22.6.3, with some differences and enhancements relative to Westmere microarchitecture summarized in Table Table 22-11. "Core PMU Comparison".

1. Exceptions are indicated for event code 0FH in the event list for this processor (<https://perfmon-events.intel.com/>); and valid bits of data source encoding field of each load latency record is limited to bits 5:4 of Table Table 22-14. "Layout of Data Source Field of Load Latency Record".

Table 22-11. Core PMU Comparison

Box	Sandy Bridge Microarchitecture	Westmere Microarchitecture	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W:32	See Section 22.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4	Use CPUID to determine # of counters. See Section 22.2.1.
PMI Overhead Mitigation	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	See Section 20.4.7.
Processor Event Based Sampling (PEBS) Events	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	See Table Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 22.3.4.4.2; - Data source encoding - STLB miss encoding - Lock transaction encoding	Data source encoding	
PEBS-Precise Store	Section 22.3.4.4.3	No	
PEBS-PDIR	Yes (using precise INST_RETIRED.ALL).	No	
Off-core Response Event	MSR 1A6H and 1A7H, extended request and response types.	MSR 1A6H and 1A7H, limited response types.	Nehalem supports 1A6H only.

22.3.4.1 Global Counter Control Facilities in Sandy Bridge Microarchitecture

The number of general-purpose performance counters visible to a logical processor can vary across Processors based on Sandy Bridge microarchitecture. Software must use CPUID to determine the number performance counters/event select registers (See Section).

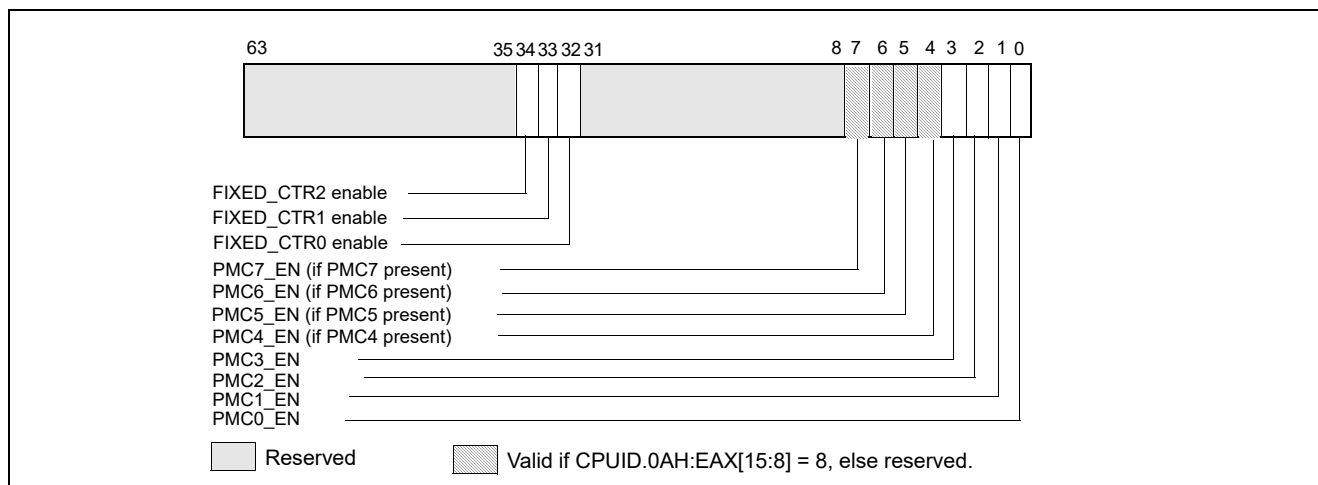
**Figure 22-28. IA32_PERF_GLOBAL_CTRL MSR in Sandy Bridge Microarchitecture**

Figure 22-46 depicts the layout of IA32_PERF_GLOBAL_CTRL MSR. The enable bits (PMC4_EN, PMC5_EN, PMC6_EN, PMC7_EN) corresponding to IA32_PMC4-IA32_PMC7 are valid only if CPUID.0AH:EAX[15:8] reports a value of '8'. If CPUID.0AH:EAX[15:8] = 4, attempts to set the invalid bits will cause #GP.

Each enable bit in IA32_PERF_GLOBAL_CTRL is ANDed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the ANDed results is true; counting is disabled when the result is false. IA32_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. IA32_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer (see Figure 22-29). A value of 1 in each bit of the PMCx_OVF field indicates an overflow condition has occurred in the associated counter.

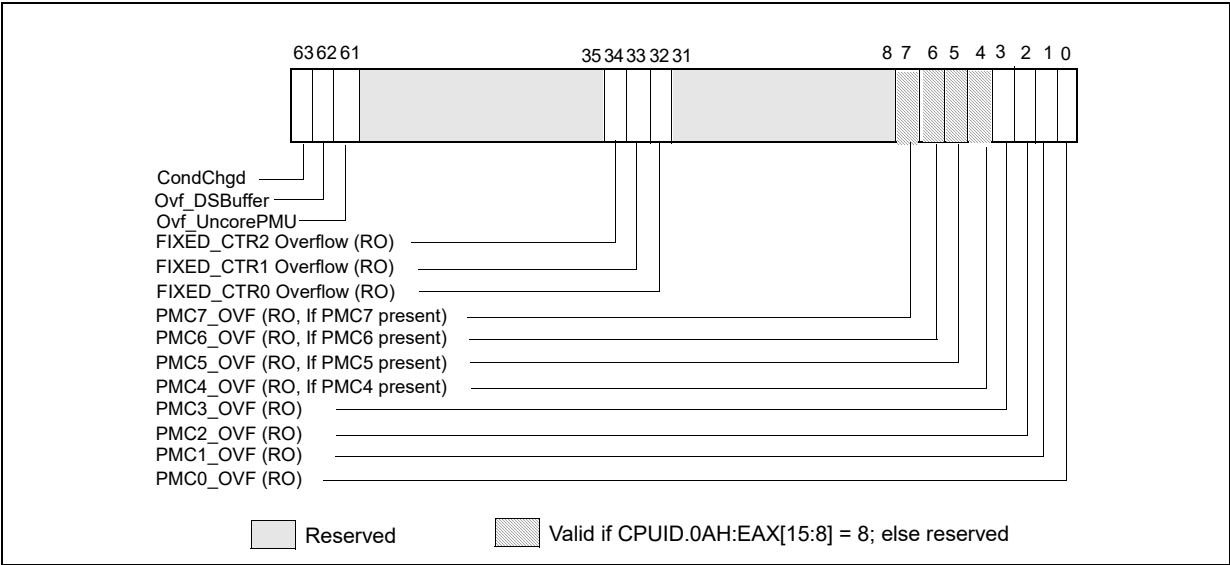


Figure 22-29. IA32_PERF_GLOBAL_STATUS MSR in Sandy Bridge Microarchitecture

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 20.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 22-30). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or interrupt based sampling.
- Reloading counter values to continue sampling.
- Disabling event counting or interrupt based sampling.

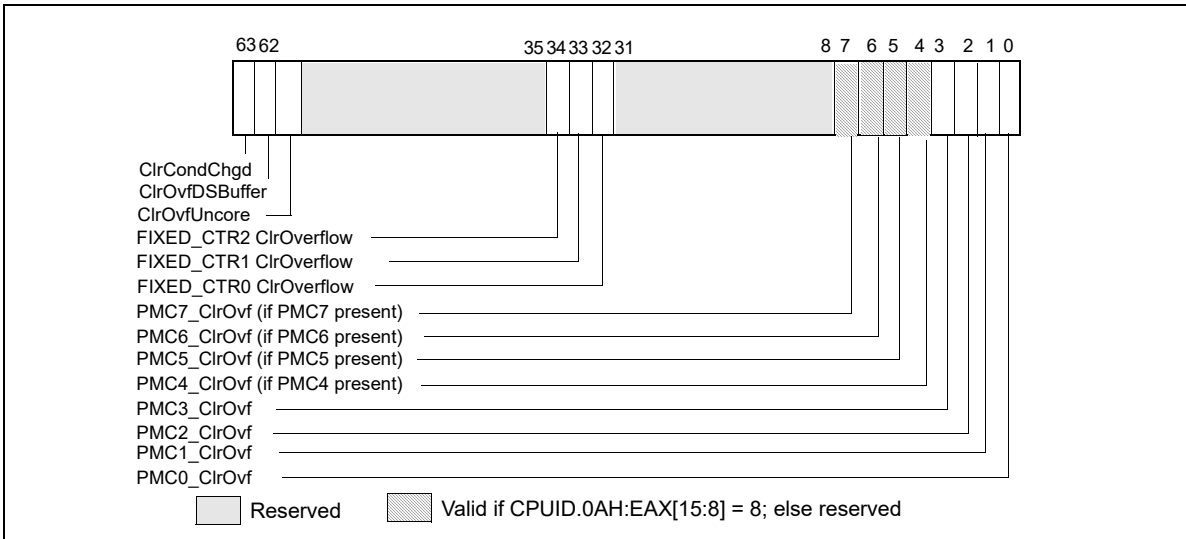


Figure 22-30. IA32_PERF_GLOBAL_OVF_CTRL MSR in Sandy Bridge Microarchitecture

22.3.4.2 Counter Coalescence

In processors based on Sandy Bridge microarchitecture, each processor core implements eight general-purpose counters. CPUID.0AH:EAX[15:8] will report the number of counters visible to software.

If a processor core is shared by two logical processors, each logical processors can access up to four counters (IA32_PMC0-IA32_PMC3). This is the same as in the prior generation for processors based on Nehalem microarchitecture.

If a processor core is not shared by two logical processors, up to eight general-purpose counters are visible. If CPUID.0AH:EAX[15:8] reports 8 counters, then IA32_PMC4-IA32_PMC7 would occupy MSR addresses 0C5H through 0C8H. Each counter is accompanied by an event select MSR (IA32_PERFEVTSEL4-IA32_PERFEVTSEL7).

If CPUID.0AH:EAX[15:8] report 4, access to IA32_PMC4-IA32_PMC7, IA32_PMC4-IA32_PMC7 will cause #GP. Writing 1's to bit position 7:4 of IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, or IA32_PERF_GLOBAL_OVF_CTL will also cause #GP.

22.3.4.3 Full Width Writes to Performance Counters

Processors based on Sandy Bridge microarchitecture support full-width writes to the general-purpose counters, IA32_PMCx. Support of full-width writes are enumerated by IA32_PERF_CAPABILITIES.FW_WRITES[13] (see Section 22.2.4).

The default behavior of IA32_PMCx is unchanged, i.e., WRMSR to IA32_PMCx results in a sign-extended 32-bit value of the input EAX written into IA32_PMCx. Full-width writes must issue WRMSR to a dedicated alias MSR address for each IA32_PMCx.

Software must check the presence of full-width write capability and the presence of the alias address IA32_A_PMCx by testing IA32_PERF_CAPABILITIES[13].

22.3.4.4 PEBS Support in Sandy Bridge Microarchitecture

Processors based on Sandy Bridge microarchitecture support PEBS, similar to those offered in prior generation, with several enhanced features. The key components and differences of PEBS facility relative to Westmere microarchitecture is summarized in Table Table 22-12. "PEBS Facility Comparison".

Table 22-12. PEBS Facility Comparison

Box	Sandy Bridge Microarchitecture	Westmere Microarchitecture	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 22.3.1.1.1	Section 22.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 22-31	Figure 22-17	
PEBS record layout	Physical Layout same as Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family".	Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family"	Enhanced fields at offsets 98H, A0H, A8H.
PEBS Events	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	See Table Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table Table 22-14. "Layout of Data Source Field of Load Latency Record".	Table Table 22-5. "Data Source Encoding for Load Latency Record"	
PEBS-Precise Store	Yes; see Section 22.3.4.4.3.	No	IA32_PMC3 only
PEBS-PDIR	Yes	No	IA32_PMC1 only
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

In IA32_PEBS_ENABLE MSR, bit 63 is defined as PS_ENABLE: When set, this enables IA32_PMC3 to capture precise store information. Only IA32_PMC3 supports the precise store facility. In typical usage of PEBS, the bit fields in IA32_PEBS_ENABLE are written to when the agent software starts PEBS operation; the enabled bit fields should be modified only when re-programming another PEBS event or cleared when the agent uses the performance counters for non-PEBS operations.

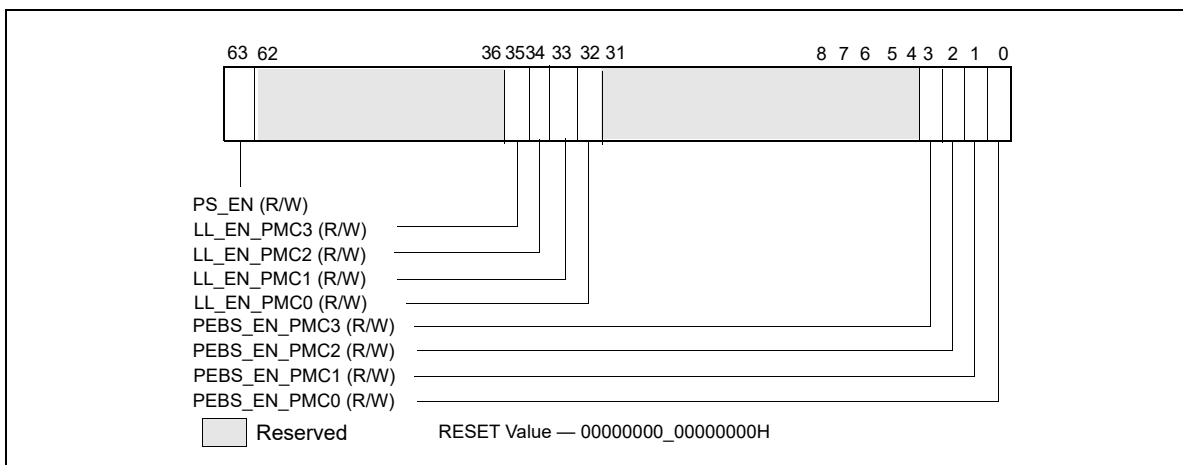


Figure 22-31. Layout of IA32_PEBS_ENABLE MSR

22.3.4.4.1 PEBS Record Format

The layout of PEBS records physically identical to those shown in Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family", but the fields at offsets 98H, A0H, and A8H have been enhanced to support additional PEBS capabilities.

- Load/Store Data Linear Address (Offset 98H): This field will contain the linear address of the source of the load, or linear address of the destination of the store.
- Data Source /Store Status (Offset A0H): When load latency is enabled, this field will contain three piece of information (including an encoded value indicating the source which satisfied the load operation). The source field encodings are detailed in Table Table 22-5. "Data Source Encoding for Load Latency Record". When precise store is enabled, this field will contain information indicating the status of the store, as detailed in Table 19.
- Latency Value/0 (Offset A8H): When load latency is enabled, this field contains the latency in cycles to service the load. This field is not meaningful when precise store is enabled and will be written to zero in that case. Upon writing the PEBS record, microcode clears the overflow status bits in the IA32_PERF_GLOBAL_STATUS corresponding to those counters that both overflowed and were enabled in the IA32_PEBS_ENABLE register. The status bits of other counters remain unaffected.

The number PEBS events has expanded. The list of PEBS events supported in Sandy Bridge microarchitecture is shown in Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".

Table 22-13. PEBS Performance Events for Sandy Bridge Microarchitecture

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST	01H ¹
UOPS_RETIRED	C2H	All	01H
		Retire_Slots	02H
BR_INST_RETIRED	C4H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Near_Return	08H
		Near_Taken	20H

Table 22-13. PEBS Performance Events for Sandy Bridge Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
BR_MISP_RETIRED	C5H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Not_Taken	10H
		Taken	20H
MEM_UOPS_RETIRED	D0H	STLB_MISS_LOADS	11H
		STLB_MISS_STORE	12H
		LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H
MEM_LOAD_UOPS_RETIRED	D1H	L1_Hit	01H
		L2_Hit	02H
		L3_Hit	04H
		Hit_LFB	40H
MEM_LOAD_UOPS_LLC_HIT_RETIRED	D2H	XSNP_Miss	01H
		XSNP_Hit	02H
		XSNP_Hitm	04H
		XSNP_None	08H

NOTES:

1. Only available on IA32_PMC1.

22.3.4.4.2 Load Latency Performance Monitoring Facility

The load latency facility in Sandy Bridge microarchitecture is similar to that in prior microarchitectures. It provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family" and Section 22.3.4.4.1. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).

To use this feature software must assure:

- One of the IA32_PERFEVTSELx MSR is programmed to specify the event unit MEM_TRANS_RETIRED, and the LATENCY_ABOVE_THRESHOLD event mask must be specified (IA32_PerfEvtSelX[15:0] = 1CDH). The corresponding counter IA32_PMCx will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in a MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with latencies greater than this value are eligible for counting and latency data reporting. The minimum value that may be programmed in this register is 3 (the minimum detectable load latency is 4 core clock cycles).
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register. This means that both the PEBS_EN_CTRX and LL_EN_CTRX bits must be set for the counter(s) of interest. For example, to enable load latency on counter IA32_PMC0, the IA32_PEBS_ENABLE register must be programmed with the 64-bit value 00000001.00000001H.

- When Load latency event is enabled, no other PEBS event can be configured with other counters.

When the load-latency facility is enabled, load operations are randomly selected by hardware and tagged to carry information related to data source locality and latency. Latency and data source information of tagged loads are updated internally. The MEM_TRANS_RETIRE event for load latency counts only tagged retired loads. If a load is cancelled it will not be counted and the internal state of the load latency facility will not be updated. In this case the hardware will tag the next available load.

When a PEBS assist occurs, the last update of latency and data source information are captured by the assist and written as part of the PEBS record. The PEBS sample after value (SAV), specified in PEBS CounterX Reset, operates orthogonally to the tagging mechanism. Loads are randomly tagged to collect latency data. The SAV controls the number of tagged loads with latency information that will be written into the PEBS record field by the PEBS assists. The load latency data written to the PEBS record will be for the last tagged load operation which retired just before the PEBS assist was invoked.

The physical layout of the PEBS records is the same as shown in Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family". The specificity of Data Source entry at offset A0H has been enhanced to report three pieces of information.

Table 22-14. Layout of Data Source Field of Load Latency Record

Field	Position	Description
Source	3:0	See Table Table 22-5. "Data Source Encoding for Load Latency Record"
STLB_MISS	4	0: The load did not miss the STLB (hit the DTLB or STLB). 1: The load missed the STLB.
Lock	5	0: The load was not part of a locked transaction. 1: The load was part of a locked transaction.
Reserved	63:6	Reserved

The layout of MSR_PEBS_LD_LAT_THRESHOLD is the same as shown in Figure 22-19.

22.3.4.4.3 Precise Store Facility

Processors based on Sandy Bridge microarchitecture offer a precise store capability that complements the load latency facility. It provides a means to profile store memory references in the system.

Precise stores leverage the PEBS facility and provide additional information about sampled stores. Having precise memory reference events with linear address information for both loads and stores can help programmers improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

Only IA32_PMC3 can be used to capture precise store information. After enabling this facility, counter overflows will initiate the generation of PEBS records as previously described in PEBS. Upon counter overflow hardware captures the linear address and other status information of the next store that retires. This information is then written to the PEBS record.

To enable the precise store facility, software must complete the following steps. Please note that the precise store facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture precise store information.

- Complete the PEBS configuration steps.
- Program the MEM_TRANS_RETIRED.PRECISE_STORE event in IA32_PERFEVTSEL3. Only counter 3 (IA32_PMC3) supports collection of precise store information.
- Set IA32_PEBS_ENABLE[3] and IA32_PEBS_ENABLE[63]. This enables IA32_PMC3 as a PEBS counter and enables the precise store facility, respectively.

The precise store information written into a PEBS record affects entries at offsets 98H, A0H, and A8H of Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family". The specificity of Data Source entry at offset A0H has been enhanced to report three piece of information.

Table 22-15. Layout of Precise Store Information In PEBS Record

Field	Offset	Description
Store Data Linear Address	98H	The linear address of the destination of the store.
Store Status	A0H	L1D Hit (Bit 0): The store hit the data cache closest to the core (lowest latency cache) if this bit is set, otherwise the store missed the data cache. STLB Miss (bit 4): The store missed the STLB if set, otherwise the store hit the STLB Locked Access (bit 5): The store was part of a locked access if set, otherwise the store was not part of a locked access.
Reserved	A8H	Reserved

22.3.4.4.4 Precise Distribution of Instructions Retired (PDIR)

Upon triggering a PEBS assist, there will be a finite delay between the time the counter overflows and when the microcode starts to carry out its data collection obligations. INST_RETIRE is a very common event that is used to sample where performance bottleneck happened and to help identify its location in instruction address space. Even if the delay is constant in core clock space, it invariably manifest as variable “skids” in instruction address space. This creates a challenge for programmers to profile a workload and pinpoint the location of bottlenecks.

The core PMU in processors based on Sandy Bridge microarchitecture include a facility referred to as precise distribution of Instruction Retired (PDIR).

The PDIR facility mitigates the “skid” problem by providing an early indication of when the INST_RETIRE counter is about to overflow, allowing the machine to more precisely trap on the instruction that actually caused the counter overflow. On processors based on Sandy Bridge microarchitecture, skid is significantly reduced and can be as little as one instruction. On future implementations, PDIR may eliminate skid.

PDIR applies only to the INST_RETIRE.ALL precise event, and processors based on Sandy Bridge microarchitecture must use IA32_PMC1 with PerfEvtSel1 property configured and bit 1 in the IA32_PEBS_ENABLE set to 1. INST_RETIRE.ALL is a non-architectural performance event, it is not supported in prior generation microarchitectures. Additionally, on processors with CPUID DisplayFamily_DisplayModel signatures of 06_2A and 06_2D, the tool that programs PDIR should quiesce the rest of the programmable counters in the core when PDIR is active.

22.3.4.5 Off-core Response Performance Monitoring

The core PMU in processors based on Sandy Bridge microarchitecture provides off-core response facility similar to prior generation. Off-core response can be programmed only with a specific pair of event select and counter MSR, and with specific event codes and predefine mask bit value in a dedicated MSR to specify attributes of the off-core transaction. Two event codes are dedicated for off-core response event programming. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Table 22-16 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 22-16. Off-Core Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-3	B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)
PMC0-3	BBH	01H	MSR_OFFCORE_RSP_1 (address 1A7H)

The layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 are shown in Figure 22-32 and Figure 22-33. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

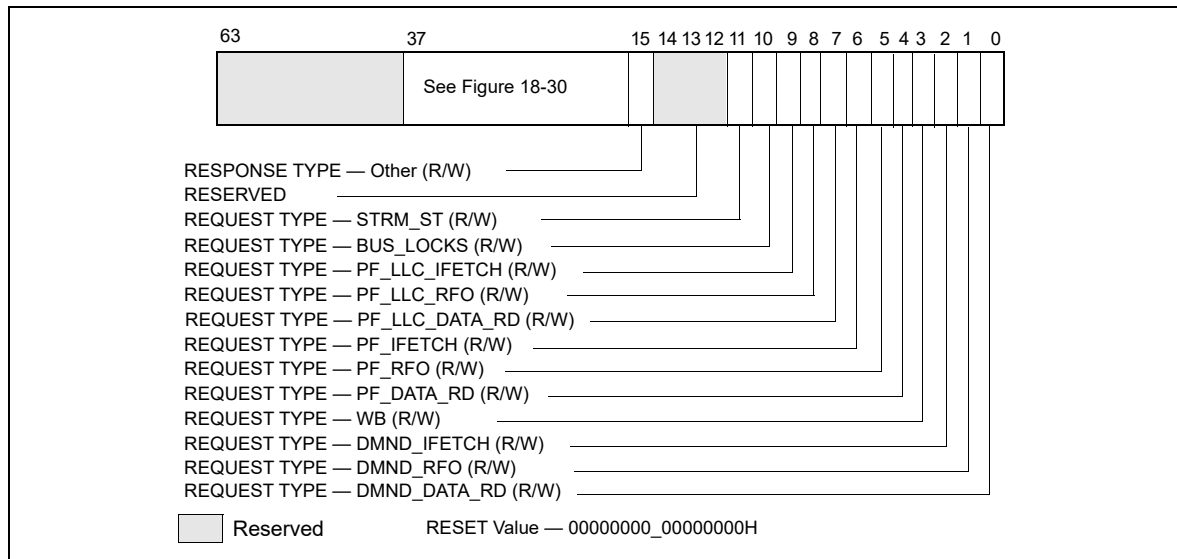


Figure 22-32. Request_Type Fields for MSR_OFFCORE_RSP_x

Table 22-17. MSR_OFFCORE_RSP_x Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PF_LLC_DATA_RD	7	L2 prefetcher to L3 for loads.
PF_LLC_RFO	8	RFO requests generated by L2 prefetcher
PF_LLC_IFETCH	9	L2 prefetcher to L3 for instruction fetches.
BUS_LOCKS	10	Bus lock and split lock requests
STRM_ST	11	Streaming store requests
OTHER	15	Any other request that crosses IDI, including I/O.

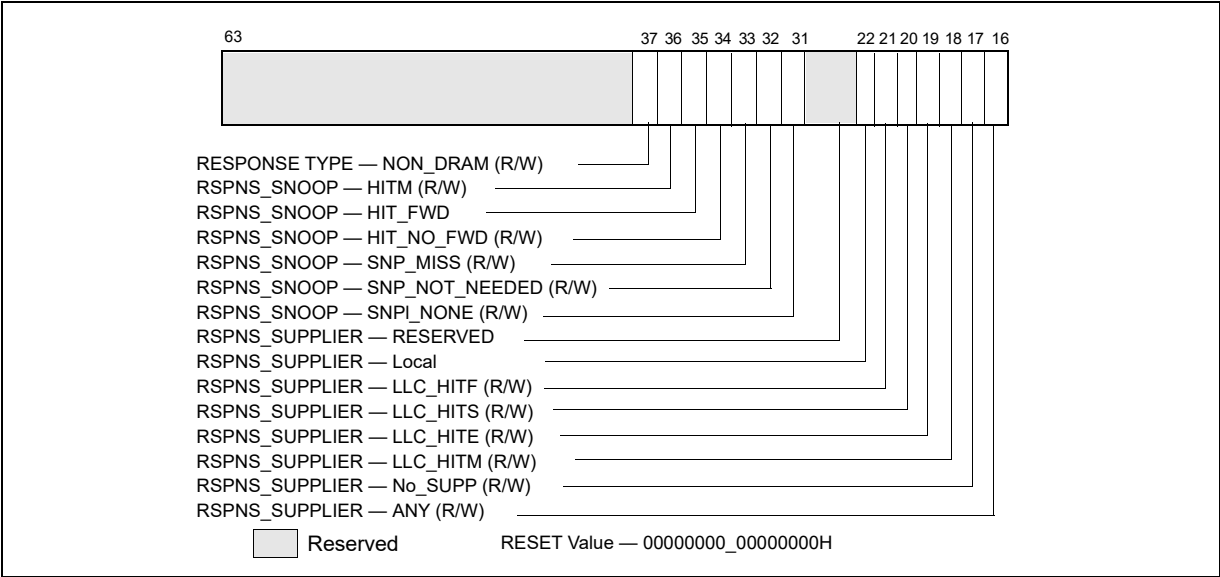


Figure 22-33. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSP_x

To properly program this extra register, software must set at least one request type bit and a valid response type pattern. Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSP_x allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 22-18. MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	LLC_HITM	18	M-state initial lookup stat in L3.
	LLC_HITE	19	E-state
	LLC_HITS	20	S-state
	LLC_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Reserved	30:23	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY” bit is set, the supplier and snoop info bits are ignored.

Table 22-19. MSR_OFFCORE_RSP_x Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	No details on snoop-related information.
	SNP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNP_MISS	33	A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitl was returned by all cores -For LLC Miss, Rspl was returned by all sockets and data was returned from DRAM.
	SNP_NO_FWD	34	A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO) -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD) -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S) In the LLC Miss case, data is returned from DRAM.
	SNP_FWD	35	A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	HITM	36	A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD) -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO) -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.

22.3.4.6 Uncore Performance Monitoring Facilities in the Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, and Intel® Core™ i3-2xxx Processor Series

The uncore sub-system in Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series provides a unified L3 that can support up to four processor cores. The L3 cache consists multiple slices, each slice interface with a processor via a coherence engine, referred to as a C-Box. Each C-Box provides dedicated facility of MSRs to select uncore performance monitoring events and each C-Box event select MSR is paired with a counter register, similar in style as those described in Section 22.3.1.2.2. The ARB unit in the uncore also provides its local performance counters and event select MSRs. The layout of the event select MSRs in the C-Boxes and the ARB unit are shown in Figure 22-34.

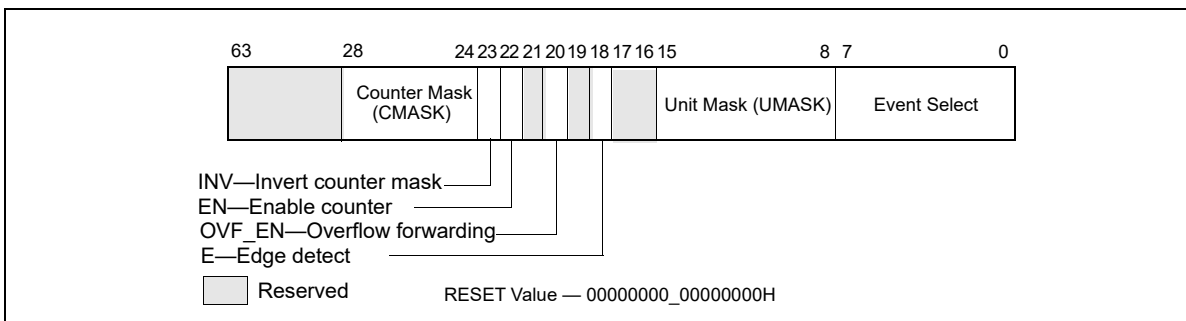


Figure 22-34. Layout of Uncore PERFECTSEL MSR for a C-Box Unit or the ARB Unit

The bit fields of the uncore event select MSRs for a C-box unit or the ARB unit are summarized below:

- Event_Select (bits 7:0) and UMASK (bits 15:8): Specifies the microarchitectural condition to count in a local uncore PMU counter, see the event list at: <https://perfmon-events.intel.com/>.
- E (bit 18): Enables edge detection filtering, if 1.
- OVF_EN (bit 20): Enables the overflow indicator from the uncore counter forwarded to MSR_UNC_PERF_-GLOBAL_CTRL, if 1.
- EN (bit 22): Enables the local counter associated with this event select MSR.
- INV (bit 23): Event count increments with non-negative value if 0, with negated value if 1.
- CMASK (bits 28:24): Specifies a positive threshold value to filter raw event count input.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 22-35 shows the layout of the uncore domain global control.

When an uncore counter overflows, a PMI can be routed to a processor core. Bits 3:0 of MSR_UNC_PERF_-GLOBAL_CTRL can be used to select which processor core to handle the uncore PMI. Software must then write to bit 13 of IA32_DEBUGCTL (at address 1D9H) to enable this capability.

- PMI_SEL_Core#: Enables the forwarding of an uncore PMI request to a processor core, if 1. If bit 30 (WakePMI) is '1', a wake request is sent to the respective processor core prior to sending the PMI.
- EN: Enables the fixed uncore counter, the ARB counters, and the CBO counters in the uncore PMU, if 1. This bit is cleared if bit 31 (FREEZE) is set and any enabled uncore counters overflow.
- WakePMI: Controls sending a wake request to any halted processor core before issuing the uncore PMI request. If a processor core was halted and not sent a wake request, the uncore PMI will not be serviced by the processor core.
- FREEZE: Provides the capability to freeze all uncore counters when an overflow condition occurs in a unit counter. When this bit is set, and a counter overflow occurs, the uncore PMU logic will clear the global enable bit (bit 29).

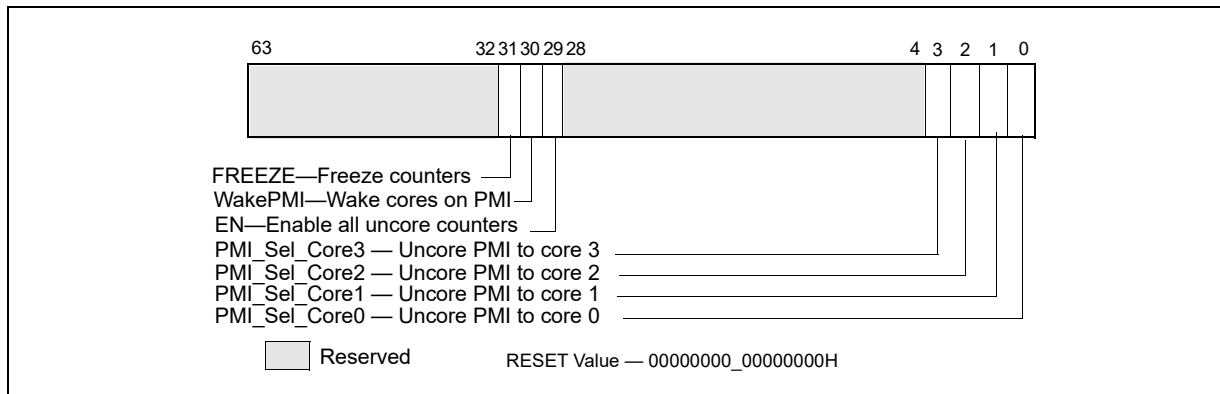


Figure 22-35. Layout of MSR_UNC_PERF_GLOBAL_CTRL MSR for Uncore

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table Table 22-20. "Uncore PMU MSR Summary" summarizes the number MSRs for uncore PMU for each box.

Table 22-20. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
C-Box	SKU specific	2	44	Yes	Per-box	Up to 4, seeTable Table 2-21. "MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)" MSR_UNC_CBO_CONFIG

Table 22-20. Uncore PMU MSR Summary (Contd.)

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
ARB	1	2	44	Yes	Uncore	
Fixed Counter	N.A.	N.A.	48	No	Uncore	

22.3.4.6.1 Uncore Performance Monitoring Events

There are certain restrictions on the uncore performance counters in each C-Box. Specifically,

- Occupancy events are supported only with counter 0 but not counter 1.
- Other uncore C-Box events can be programmed with either counter 0 or 1.

The C-Box uncore performance events can collect performance characteristics of transactions initiated by processor core. In that respect, they are similar to various sub-events in the OFFCORE_RESPONSE family of performance events in the core PMU. Information such as data supplier locality (LLC HIT/MISS) and snoop responses can be collected via OFFCORE_RESPONSE and qualified on a per-thread basis.

On the other hand, uncore performance event logic cannot associate its counts with the same level of per-thread qualification attributes as the core PMU events can. Therefore, whenever similar event programming capabilities are available from both core PMU and uncore PMU, the recommendation is that utilizing the core PMU events may be less affected by artifacts, complex interactions and other factors.

22.3.4.7 Intel® Xeon® Processor E5 Family Performance Monitoring Facility

The Intel® Xeon® Processor E5 Family (and Intel® Core™ i7-3930K Processor) are based on Sandy Bridge-E microarchitecture. While the processor cores share the same microarchitecture as those of the Intel® Xeon® Processor E3 Family and 2nd generation Intel Core i7-2xxx, Intel Core i5-2xxx, Intel Core i3-2xxx processor series, the uncore subsystems are different. An overview of the uncore performance monitoring facilities of the Intel Xeon processor E5 family (and Intel Core i7-3930K processor) is described in Section 22.3.4.8.

Thus, the performance monitoring facilities in the processor core generally are the same as those described in Section 22.6.3 through Section 22.3.4.5. However, the MSR_OFFCORE_RSP_0/MSR_OFFCORE_RSP_1 Response Supplier Info field shown in Table 22-18 applies to Intel Core Processors with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2AH; Intel Xeon processor with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2DH supports an additional field for remote DRAM controller shown in Table 22-21. Additionally, there are some small differences in the non-architectural performance monitoring events (see event list available at: <https://perfmon-events.intel.com/>).

Table 22-21. MSR_OFFCORE_RSP_x Supplier Info Field Definitions

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	LLC_HITM	18	M-state initial lookup stat in L3.
	LLC_HITE	19	E-state
	LLC_HITS	20	S-state
	LLC_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Remote	30:23	Remote DRAM Controller (either all 0s or all 1s).

22.3.4.8 Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5-2600 product family has some similarities with those of the Intel Xeon processor E7 family. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope. For example, each Cbox caching agent has a set of local performance counters, and the power controller unit (PCU) has its own local performance counters. Up to 8 C-Box units are supported in the uncore sub-system.

Table Table 22-22. "Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family" summarizes the uncore PMU facilities providing MSR interfaces.

Table 22-22. Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	4	44	Yes	per-box	None
PCU	1	4	48	Yes	per-box	Match/Mask
U-Box	1	2	44	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 family is available in "Intel® Xeon® Processor E5 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 2-24.

22.3.5 3rd Generation Intel® Core™ Processor Performance Monitoring Facility

The 3rd generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v2 product family are based on the Ivy Bridge microarchitecture. The performance monitoring facilities in the processor core generally are the same as those described in Section 22.6.3 through Section 22.3.4.5. The non-architectural performance monitoring events supported by the processor core can be found at: <https://perfmon-events.intel.com/>.

22.3.5.1 Intel® Xeon® Processor E5 v2 and E7 v2 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5 v2 and Intel Xeon Processor E7 v2 product families are based on the Ivy Bridge-E microarchitecture. There are some similarities with those of the Intel Xeon processor E5 family based on the Sandy Bridge microarchitecture. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope.

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v2 and Intel Xeon Processor E7 v2 families are available in the "Intel® Xeon® Processor E5 v2 and E7 v2 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 2-28.

22.3.6 4th Generation Intel® Core™ Processor Performance Monitoring Facility

The 4th generation Intel® Core™ processor and Intel® Xeon® processor E3-1200 v3 product family are based on the Haswell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 22.2.3.

The core PMU's capability is similar to those described in Section 22.6.3 through Section 22.3.4.5, with some differences and enhancements summarized in Table Table 22-23. "Core PMU Comparison". Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 22.3.6.5. For details of Intel TSX, see Chapter 16, "Programming with Intel® AVX10," of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

Table 22-23. Core PMU Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 22.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to determine # of counters. See Section 22.2.1.
PMI Overhead Mitigation	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	See Section 20.4.7.
Processor Event Based Sampling (PEBS) Events	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture" and Section 22.3.6.5.1.	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 22.3.4.4.2.	See Section 22.3.4.4.2.	
PEBS-Precise Store	No, replaced by Data Address profiling.	Section 22.3.4.4.3	
PEBS-PDIR	Yes (using precise INST_RETIRED.ALL)	Yes (using precise INST_RETIRED.ALL)	
PEBS-EventingIP	Yes	No	
Data Address Profiling	Yes	No	
LBR Profiling	Yes	Yes	
Call Stack Profiling	Yes, see Section 20.11.	No	Use LBR facility.
Off-core Response Event	MSR 1A6H and 1A7H; extended request and response types.	MSR 1A6H and 1A7H; extended request and response types.	
Intel TSX support for PerfMon	See Section 22.3.6.5.	No	

22.3.6.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 4th Generation Intel Core processor is similar to those in processors based on Sandy Bridge microarchitecture, with several enhanced features. The key components and differences of PEBS facility relative to Sandy Bridge microarchitecture is summarized in Table Table 22-24. "PEBS Facility Comparison".

Table 22-24. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7
PEBS Buffer Programming	Section 22.3.1.1.1	Section 22.3.1.1.1	Unchanged
IA32_PEBE_ENABLE Layout	Figure 22-17	Figure 22-31	

Table 22-24. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
PEBS record layout	Table Table 22-25. "PEBS Record Format for 4th Generation Intel Core Processor Family"; enhanced fields at offsets 98H, A0H, A8H, B0H.	Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family"; enhanced fields at offsets 98H, A0H, A8H.	
Precise Events	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table Table 22-14. "Layout of Data Source Field of Load Latency Record".	Table Table 22-14. "Layout of Data Source Field of Load Latency Record"	
PEBS-Precise Store	No, replaced by data address profiling.	Yes; see Section 22.3.4.4.3.	
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

22.3.6.2 PEBS Data Format

The PEBS record format for the 4th Generation Intel Core processor is shown in Table Table 22-25. "PEBS Record Format for 4th Generation Intel Core Processor Family". The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 22-25. PEBS Record Format for 4th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS

Table 22-25. PEBS Record Format for 4th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
38H	R/EDI	98H	Data Linear Address
40H	R/EBP	A0H	Data Source Encoding
48H	R/ESP	A8H	Latency value (core cycles)
50H	R8	B0H	EventingIP
58H	R9	B8H	TX Abort Information (Section 22.3.6.5.1)

The layout of PEBS records are almost identical to those shown in Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family". Offset B0H is a new field that records the eventing IP address of the retired instruction that triggered the PEBS assist.

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 22.3.4.4.2), PDIR (Section 22.3.4.4.4), and the equivalent capability of precise store in prior generation (see Section 22.3.6.3).

In the core PMU of the 4th generation Intel Core processor, load latency facility and PDIR capabilities are unchanged. However, precise store is replaced by an enhanced capability, data address profiling, that is not restricted to store address. Data address profiling also records information in PEBS records at offsets 98H, A0H, and ABH.

22.3.6.3 PEBS Data Address Profiling

The Data Linear Address facility is also abbreviated as DataLA. The facility is a replacement or extension of the precise store facility in previous processor generations. The DataLA facility complements the load latency facility by providing a means to profile load and store memory references in the system, leverages the PEBS facility, and provides additional information about sampled loads and stores. Having precise memory reference events with linear address information for both loads and stores provides information to improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

The DataLA facility in the 4th generation processor supports the following events configured to use PEBS:

Table 22-26. Precise Events That Supports Data Linear Address Profiling

Event Name	Event Name
MEM_UOPS_RETIRE.STLB_MISS_LOADS	MEM_UOPS_RETIRE.STLB_MISS_STORES
MEM_UOPS_RETIRE.LOCK_LOADS	MEM_UOPS_RETIRE.SPLIT_STORES
MEM_UOPS_RETIRE.SPLIT_LOADS	MEM_UOPS_RETIRE.ALL_STORES
MEM_UOPS_RETIRE.ALL_LOADS	MEM_LOAD_UOPS_LLC_MISS_RETIRE.LOCAL_DRAM
MEM_LOAD_UOPS_RETIRE.L1_HIT	MEM_LOAD_UOPS_RETIRE.L2_HIT
MEM_LOAD_UOPS_RETIRE.L3_HIT	MEM_LOAD_UOPS_RETIRE.L1_MISS
MEM_LOAD_UOPS_RETIRE.L2_MISS	MEM_LOAD_UOPS_RETIRE.L3_MISS
MEM_LOAD_UOPS_RETIRE.HIT_LFB	MEM_LOAD_UOPS_L3_HIT_RETIRE.XSNP_MISS
MEM_LOAD_UOPS_L3_HIT_RETIRE.XSNP_HIT	MEM_LOAD_UOPS_L3_HIT_RETIRE.XSNP_HITM
UOPS_RETIRE.ALL (if load or store is tagged)	MEM_LOAD_UOPS_LLC_HIT_RETIRE.XSNP_NONE

DataLA can use any one of the IA32_PMC0-IA32_PMC3 counters. Counter overflows will initiate the generation of PEBS records. Upon counter overflow, hardware captures the linear address and possible other status information of the retiring memory uop. This information is then written to the PEBS record that is subsequently generated.

To enable the DataLA facility, software must complete the following steps. Please note that the DataLA facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture DataLA information.

- Complete the PEBS configuration steps.
- Program an event listed in Table 22-26 using any one of IA32_PERFVTSEL0-IA32_PERFVTSEL3.
- Set the corresponding IA32_PEBS_ENABLE.PEBS_EN_CTRx bit. This enables the corresponding IA32_PMCx as a PEBS counter and enables the DataLA facility.

When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H, and A8H, as shown in Table 22-27. "Layout of Data Linear Address Information In PEBS Record".

Table 22-27. Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	▪ DCU Hit (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_UOPS_RETIRED.STLB_MISS_STORES, MEM_UOPS_RETIRED.SPLIT_STORES, MEM_UOPS_RETIRED.ALL_STORES ▪ Other bits are zero, The STLB_MISS, LOCK bit information can be obtained by programming the corresponding store event in Table 22-26. "Precise Events That Supports Data Linear Address Profiling".
Reserved	A8H	Always zero.

22.3.6.3.1 EventingIP Record

The PEBS record layout for processors based on Haswell microarchitecture adds a new field at offset 0B0H. This is the eventingIP field that records the IP address of the retired instruction that triggered the PEBS assist. The EIP/RIP field at offset 08H records the IP address of the next instruction to be executed following the PEBS assist.

22.3.6.4 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 22.3.4.5. The event codes are listed in Table 22-16. "Off-Core Response Event Encoding". Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 22-28. "MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture)".
- Supplier information (bits 30:16): see Table 22-29. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures: 06_3CH, 06_46H)".
- Snoop response information (bits 37:31): see Table 22-19. "MSR_OFFCORE_RSP_x Snoop Info Field Definition".

Table 22-28. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture)

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts demand read (RFO) and software prefetches (PREFETCHW) for exclusive ownership in anticipation of a write.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
COREWB	3	Counts the number of modified cachelines written back.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.

Table 22-28. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture) (Contd.)

Bit Name	Offset	Description
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PF_L3_DATA_RD	7	Counts the number of data cacheline reads generated by L3 prefetchers.
PF_L3_RFO	8	Counts the number of RFO requests generated by L3 prefetchers.
PF_L3_CODE_RD	9	Counts the number of code reads generated by L3 prefetchers.
SPLIT_LOCK_UC_LOCK	10	Counts the number of lock requests that split across two cachelines or are to UC memory.
STRM_ST	11	Counts the number of streaming store requests electronically.
Reserved	14:12	Reserved
OTHER	15	Any other request that crosses IDI, including I/O.

The supplier information field listed in Table Table 22-29. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures: 06_3CH, 06_46H)". The fields vary across products (according to CPUID signatures) and is noted in the description.

Table 22-29. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures: 06_3CH, 06_46H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	LOCAL	22	Local DRAM Controller.
	Reserved	30:23	Reserved

Table 22-30. MSR_OFFCORE_RSP_x Supplier Info Field Definition (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

22.3.6.4.1 Off-core Response Performance Monitoring in Intel Xeon Processors E5 v3 Series

Table 22-29. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures: 06_3CH, 06_46H)" lists the supplier information field that apply to Intel Xeon processor E5 v3 series (CPUID signature 06_3FH).

Table 22-31. MSR_OFFCORE_RSP_x Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	L3_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Reserved	26:23	Reserved
	L3_MISS_REMOTE_HOP0	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P	29	Hop 2 or more Remote supplier.
	Reserved	30	Reserved

22.3.6.5 Performance Monitoring and Intel® TSX

Chapter 16 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, describes the details of Intel® Transactional Synchronization Extensions (Intel® TSX). This section describes performance monitoring support for Intel TSX.

If a processor supports Intel TSX, the core PMU enhances its IA32_PERFEVTSELx MSR with two additional bit fields for event filtering. Support for Intel TSX is indicated by either (a) CPUID.07H.00H:EBX.RTM[11]=1, or (b) if CPUID.07H.00H:EBX.HLE[4] = 1. The TSX-enhanced layout of IA32_PERFEVTSELx is shown in Figure 22-36. The two additional bit fields are:

- **IN_TX** (bit 32): When set, the counter will only include counts that occurred inside a transactional region, regardless of whether that region was aborted or committed. This bit may only be set if the processor supports HLE or RTM.
- **IN_TXCP** (bit 33): When set, the counter will not include counts that occurred inside of an aborted transactional region. This bit may only be set if the processor supports HLE or RTM. This bit may only be set for IA32_PERFEVTSEL2.

When the IA32_PERFEVTSELx MSR is programmed with both IN_TX=0 and IN_TXCP=0 on a processor that supports Intel TSX, the result in a counter may include detectable conditions associated with a transaction code region for its aborted execution (if any) and completed execution.

In the initial implementation, software may need to take pre-caution when using the IN_TXCP bit. See Table 2-29.

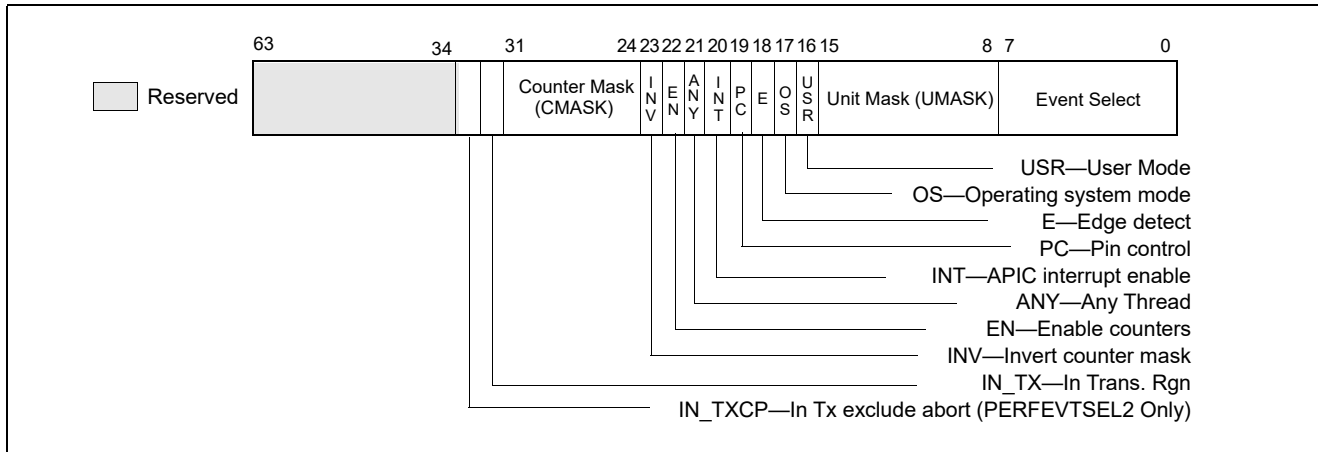


Figure 22-36. Layout of IA32_PERFEVTSELx MSRs Supporting Intel TSX

A common usage of setting `IN_TXCP=1` is to capture the number of events that were discarded due to a transactional abort. With `IA32_PMC2` configured to count in such a manner, then when a transactional region aborts, the value for that counter is restored to the value it had prior to the aborted transactional region. As a result, any updates performed to the counter during the aborted transactional region are discarded.

On the other hand, setting `IN_TX=1` can be used to drill down on the performance characteristics of transactional code regions. When a `PMCx` is configured with the corresponding `IA32_PERFEVTSELx.IN_TX=1`, only eventing conditions that occur inside transactional code regions are propagated to the event logic and reflected in the counter result. Eventing conditions specified by `IA32_PERFEVTSELx` but occurring outside a transactional region are discarded.

Additionally, a number of performance events are solely focused on characterizing the execution of Intel TSX transactional code, they can be found at: <https://perfmon-events.intel.com/>.

22.3.6.5.1 Intel® TSX and PEBS Support

If a PEBS event would have occurred inside a transactional region, then the transactional region first aborts, and then the PEBS event is processed.

Two of the TSX performance monitoring events also support using the PEBS facility to capture additional information. They are:

- `HLE_RETIREDA.BORTED` (encoding `C8H` mask `04H`),
- `RTM_RETIREDA.BORTED` (encoding `C9H` mask `04H`).

A transactional abort (`HLE_RETIREDA.BORTED`, `RTM_RETIREDA.BORTED`) can also be programmed to cause PEBS events. In this scenario, a PEBS event is processed following the abort.

Pending a PEBS record inside of a transactional region will cause a transactional abort. If a PEBS record was pended at the time of the abort or on an overflow of the TSX PEBS events listed above, only the following PEBS entries will be valid (enumerated by PEBS entry offset `B8H` bits[33:32] to indicate an HLE abort or an RTM abort):

- Offset `B0H`: EventingIP,
- Offset `B8H`: TX Abort Information

These fields are set for all PEBS events.

- Offset `08H` (RIP/EIP) corresponds to the instruction following the outermost `XACQUIRE` in HLE or the first instruction of the fallback handler of the outermost `XBEGIN` instruction in RTM. This is useful to identify the aborted transactional region.

In the case of HLE, an aborted transaction will restart execution deterministically at the start of the HLE region. In the case of RTM, an aborted transaction will transfer execution to the RTM fallback handler.

The layout of the TX Abort Information field is given in Table Table 22-32. "TX Abort Information Field Definition".

Table 22-32. TX Abort Information Field Definition

Bit Name	Offset	Description
Cycles_Last_TX	31:0	The number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
HLE_Abort	32	If set, the abort information corresponds to an aborted HLE execution
RTM_Abort	33	If set, the abort information corresponds to an aborted RTM execution
Instruction_Abort	34	If set, the abort was associated with the instruction corresponding to the eventing IP (offset 0B0H) within the transactional region.
Non_Instruction_Abort	35	If set, the instruction corresponding to the eventing IP may not necessarily be related to the transactional abort.
Retry	36	If set, retrying the transactional execution may have succeeded.
Data_Conflict	37	If set, another logical processor conflicted with a memory address that was part of the transactional region that aborted.
Capacity Writes	38	If set, the transactional region aborted due to exceeding resources for transactional writes.
Capacity Reads	39	If set, the transactional region aborted due to exceeding resources for transactional reads.
In_Suspend	40	Transaction was aborted while in a suspend region. This is an Intel Xeon processor only feature, available beginning with 4th generation Intel Xeon Scalable Processor Family; otherwise reserved.
Reserved	63:41	Reserved

22.3.6.6 Uncore Performance Monitoring Facilities in the 4th Generation Intel® Core™ Processors

The uncore sub-system in the 4th Generation Intel® Core™ processors provides its own performance monitoring facility. The uncore PMU facility provides dedicated MSRs to select uncore performance monitoring events in a similar manner as those described in Section 22.3.4.6.

The ARB unit and each C-Box provide local pairs of event select MSR and counter register. The layout of the event select MSRs in the C-Boxes are identical as shown in Figure 22-34.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 22-35 shows the layout of the uncore domain global control.

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 22-20. "Uncore PMU MSR Summary" summarizes the number MSRs for uncore PMU for each box.

Table 22-33. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
C-Box	SKU specific	2	44	Yes	Per-box	Up to 4, see Table 2-21 MSR_UNC_CBO_CONFIG
ARB	1	2	44	Yes	Uncore	
Fixed Counter	N.A.	N.A.	48	No	Uncore	

The uncore performance events for the C-Box and ARB units can be found at: <https://perfmon-events.intel.com/>.

22.3.6.7 Intel® Xeon® Processor E5 v3 Family Uncore Performance Monitoring Facility

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v3 families are available in "Intel® Xeon® Processor E5 v3 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 2-33.

22.3.7 5th Generation Intel® Core™ Processor and Intel® Core™ M Processor Performance Monitoring Facility

The 5th Generation Intel® Core™ processor and the Intel® Core™ M processor families are based on the Broadwell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 22.2.3.

The core PMU has the same capability as those described in Section 22.3.6. IA32_PERF_GLOBAL_STATUS provide a bit indicator (bit 55) for PMI handler to distinguish PMI due to output buffer overflow condition due to accumulating packet data from Intel Processor Trace.

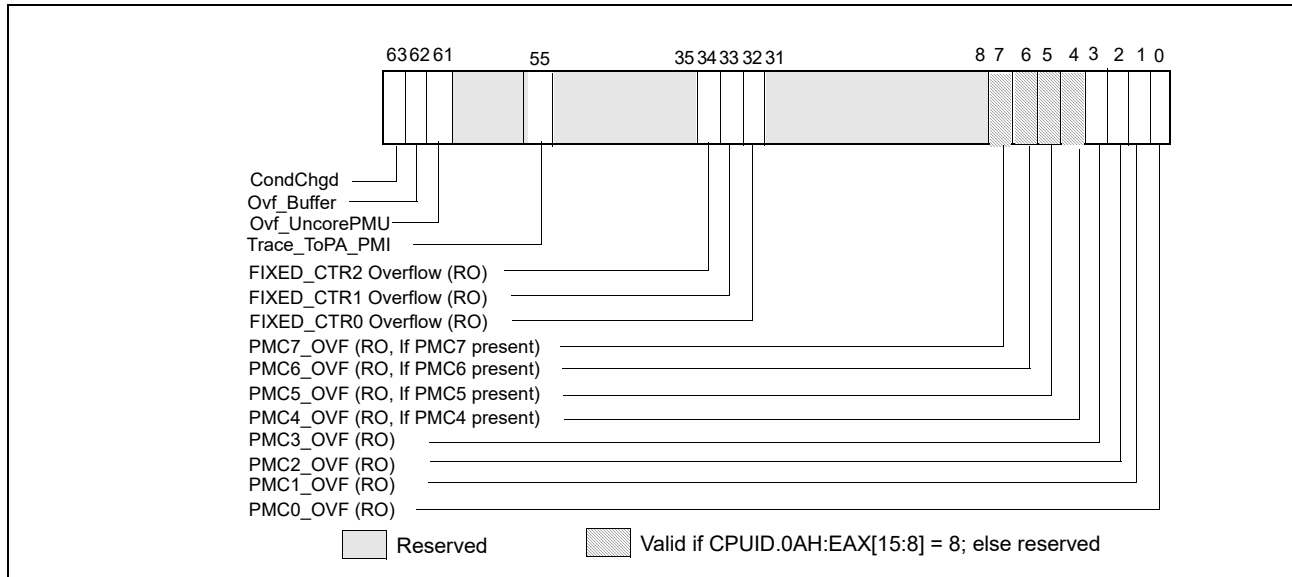


Figure 22-37. IA32_PERF_GLOBAL_STATUS MSR in Broadwell Microarchitecture

Details of Intel Processor Trace is described in Chapter 36, “Intel® Processor Trace.” The IA32_PERF_GLOBAL_OVF_CTRL MSR provides a corresponding reset control bit.

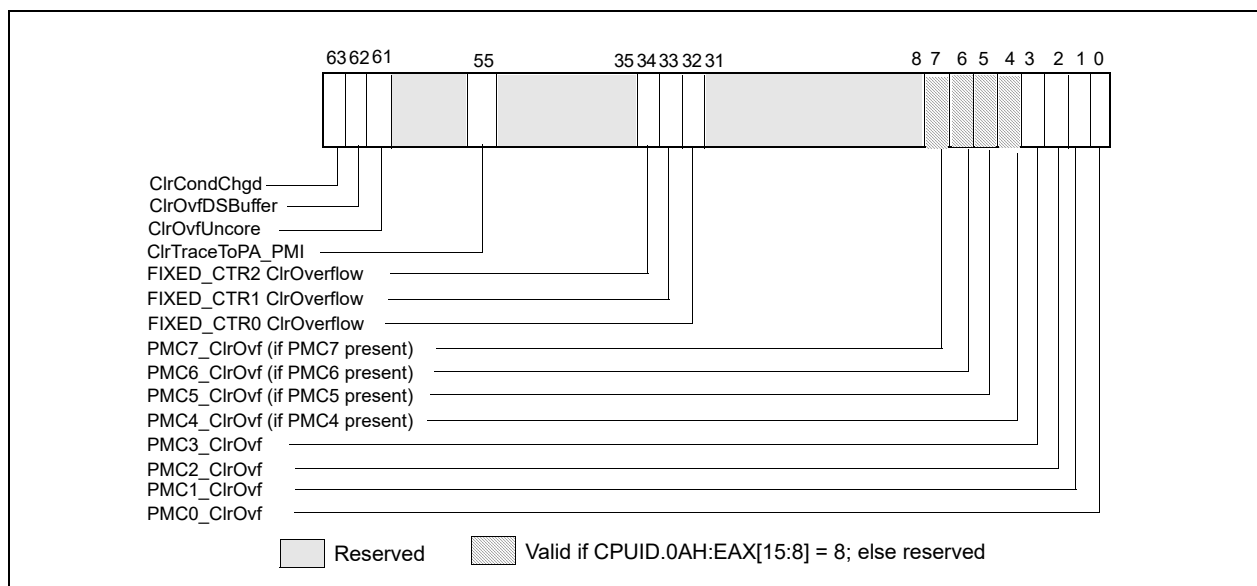


Figure 22-38. IA32_PERF_GLOBAL_OVF_CTRL MSR in Broadwell microarchitecture

The specifics of non-architectural performance events can be found at: <https://perfmon-events.intel.com/>.

22.3.8 6th Generation, 7th Generation and 8th Generation Intel® Core™ Processor Performance Monitoring Facility

The 6th generation Intel® Core™ processor is based on the Skylake microarchitecture. The 7th generation Intel® Core™ processor is based on the Kaby Lake microarchitecture. The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture. For these microarchitectures, the core PMU supports architectural performance monitoring capability with version ID 4 (see Section 22.2.4) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 4 capabilities are described in Section 22.2.4.

The core PMU's capability is similar to those described in Section 22.6.3 through Section 22.3.4.5, with some differences and enhancements summarized in Table 22-34. "Core PMU Comparison". Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 22.3.6.5. For details of Intel TSX, see Chapter 16, "Programming with Intel® AVX10," of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

Performance monitoring result may be affected by side-band activity on processors that support Intel SGX, details are described in Chapter 43, "Enclave Code Debug and Profiling."

Table 22-34. Core PMU Comparison

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 22.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to determine # of counters. See Section 22.2.1.
Architectural PerfMon version	4	3	See Section 22.2.4
PMI Overhead Mitigation	- Freeze_PerfMon_on_PMI with streamlined semantics. - Freeze_LBR_on_PMI with streamlined semantics. - Freeze_while_SMM.	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	See Section 20.4.7. Legacy semantics not supported with version 4 or higher.
Counter and Buffer Overflow Status Management	- Query via IA32_PERF_GLOBAL_STATUS - Reset via IA32_PERF_GLOBAL_STATUS_RESET - Set via IA32_PERF_GLOBAL_STATUS_SET	- Query via IA32_PERF_GLOBAL_STATUS - Reset via IA32_PERF_GLOBAL_OVF_CTRL	See Section 22.2.4.

Table 22-34. Core PMU Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/Overhead/Interference	- Individual counter overflow - PEBS buffer overflow - ToPA buffer overflow - CTR_Frz, LBR_Frz, ASCI	- Individual counter overflow - PEBS buffer overflow - ToPA buffer overflow (applicable to Broadwell microarchitecture)	See Section 22.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	- CTR_Frz - LBR_Frz	NA	See Section 22.2.4.1.
PerfMon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	NA	See Section 22.2.4.3.
Precise Events	See Table Table 22-37. "Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures".	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	IA32_PMC4-PMC7 do not support PEBS.
PEBS for front end events	See Section 22.3.8.2.	No	
LBR Record Format Encoding	000101b	000100b	Section 20.4.8.1
LBR Size	32 entries	16 entries	
LBR Entry	From_IP/To_IP/LBR_Info triplet	From_IP/To_IP pair	Section 20.12
LBR Timing	Yes	No	Section 20.12.1
Call Stack Profiling	Yes, see Section 20.11	Yes, see Section 20.11	Use LBR facility.
Off-core Response Event	MSR 1A6H and 1A7H; Extended request and response types.	MSR 1A6H and 1A7H; Extended request and response types.	
Intel TSX support for PerfMon	See Section 22.3.6.5.	See Section 22.3.6.5.	

22.3.8.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 6th generation, 7th generation and 8th generation Intel Core processors provides a number enhancement relative to PEBS in processors based on Haswell/Broadwell microarchitectures. The key components and differences of PEBS facility relative to Haswell/Broadwell microarchitecture is summarized in Table Table 22-35. "PEBS Facility Comparison".

Table 22-35. PEBS Facility Comparison

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 22.3.1.1.1	Section 22.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 22-17	Figure 22-17	
PEBS-EventingIP	Yes	Yes	
PEBS record format encoding	0011b	0010b	
PEBS record layout	Table Table 22-36. "PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families"; enhanced fields at offsets 98H- B8H; and TSC record field at C0H.	Table Table 22-25. "PEBS Record Format for 4th Generation Intel Core Processor Family"; enhanced fields at offsets 98H, A0H, A8H, B0H.	

Table 22-35. PEBS Facility Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
Multi-counter PEBS resolution	PEBS record 90H resolves the eventing counter overflow.	PEBS record 90H reflects IA32_PERF_GLOBAL_STATUS.	
Precise Events	See Table Table 22-37. "Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures".	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS-Load Latency	See Section 22.3.4.4.2.	See Section 22.3.4.4.2.	
Data Address Profiling	Yes	Yes	
FrontEnd event support	FrontEnd_Retried event and MSR_PEBS_FRONTEND.	No	IA32_PMC0-PMC3 only.

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTES

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

22.3.8.1.1 PEBS Data Format

The PEBS record format for the 6th generation, 7th generation and 8th generation Intel Core processors is reporting with encoding 0011b in IA32_PERF_CAPABILITIES[11:8]. The lay out is shown in Table Table 22-36. "PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families". The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 22-36. PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counter
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Data Source Encoding
40H	R/EBP	A8H	Latency value (core cycles)

Table 22-36. PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families

Byte Offset	Field	Byte Offset	Field
48H	R/ESP	B0H	EventingIP
50H	R8	B8H	TX Abort Information (Section 22.3.6.5.1)
58H	R9	C0H	TSC
60H	R10		

The layout of PEBS records are largely identical to those shown in Table Table 22-25. "PEBS Record Format for 4th Generation Intel Core Processor Family".

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 22.3.4.4.2), PDIR (Section 22.3.4.4.4), and data address profiling (Section 22.3.6.3).

In the core PMU of the 6th generation, 7th generation and 8th generation Intel Core processors, load latency facility and PDIR capabilities and data address profiling are unchanged relative to the 4th generation and 5th generation Intel Core processors. Similarly, precise store is replaced by data address profiling.

With format 0010b, a snapshot of the IA32_PERF_GLOBAL_STATUS may be useful to resolve the situations when more than one of IA32_PMICx have been configured to collect PEBS data and two consecutive overflows of the PEBS-enabled counters are sufficiently far apart in time. It is also possible for the image at 90H to indicate multiple PEBS-enabled counters have overflowed. In the latter scenario, software cannot to correlate the PEBS record entry to the multiple overflowed bits.

With PEBS record format encoding 0011b, offset 90H reports the "applicable counter" field, which is a multi-counter PEBS resolution index allowing software to correlate the PEBS record entry with the eventing PEBS overflow when multiple counters are configured to record PEBS records. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

22.3.8.1.2 PEBS Events

The list of precise events supported for PEBS in the Skylake, Kaby Lake and Coffee Lake microarchitectures is shown in Table Table 22-37. "Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures".

Table 22-37. Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST ¹	01H
		ALL_CYCLES ²	01H
OTHER_ASSISTS	C1H	ANY	3FH
BR_INST_RETIRED	C4H	CONDITIONAL	01H
		NEAR_CALL	02H
		ALL_BRANCHES	04H
		NEAR_RETURN	08H
		NEAR_TAKEN	20H
		FAR_BRACHES	40H
BR_MISP_RETIRED	C5H	CONDITIONAL	01H
		ALL_BRANCHES	04H
		NEAR_TAKEN	20H
FRONTEND_RETIRED	C6H	<Programmable ³ >	01H
HLE_RETIRED	C8H	ABORTED	04H
RTM_RETIRED	C9H	ABORTED	04H

Table 22-37. Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_INST_RETIRED ²	D0H	LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H
MEM_LOAD_RETIRED ⁴	D1H	L1_HIT	01H
		L2_HIT	02H
		L3_HIT	04H
		L1_MISS	08H
		L2_MISS	10H
		L3_MISS	20H
		HIT_LFB	40H
MEM_LOAD_L3_HIT_RETIRED ²	D2H	XSNP_MISS	01H
		XSNP_HIT	02H
		XSNP_HITM	04H
		XSNP_NONE	08H

NOTES:

1. Only available on IA32_PMC1.
2. INST_RETIRED.ALL_CYCLES is configured with additional parameters of cmask = 10 and INV = 1
3. Subevents are specified using MSR_PEBBS_FRONTEND, see Section 22.3.8.3
4. Instruction with at least one load uop experiencing the condition specified in the UMask.

22.3.8.1.3 Data Address Profiling

The PEBS Data address profiling on the 6th generation, 7th generation and 8th generation Intel Core processors is largely unchanged from the prior generation. When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H, and A8H, as shown in Table Table 22-27. "Layout of Data Linear Address Information In PEBS Record".

Table 22-38. Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	▪ DCU Hit (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_INST_RETIRED.STLB_MISS_STORES, MEM_INST_RETIRED.ALL_STORES, MEM_INST_RETIRED.SPLIT_STORES. ▪ Other bits are zero.
Reserved	A8H	Always zero.

22.3.8.2 Frontend Retired Facility

The Skylake Core PMU has been extended to cover common microarchitectural conditions related to the front end pipeline in addition to providing a generic latency mechanism that can locate fetch bubbles without necessarily attributing them to a particular condition. The facility counts the events if the associated instruction reaches retire-

ment (architecturally committed). Additionally, the user may opt to enable the PEBS facility to obtain precise information on the context of the event, e.g., EventingIP.

The supported frontend microarchitectural conditions require the following interfaces:

- The IA32_PERFEVTSELx MSR must select the FRONTEND_RETIRED event, EventSelect = C6H and UMASK = 01H.
- This event employs a new MSR, MSR_PEBS_FRONTEND, to specify the supported frontend event details, see Table Table 22-39. "FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL".
- If precise information is desired, program the PEBS_EN_PMCx field of IA32_PEBS_ENABLE MSR as required.

Note the AnyThread field of IA32_PERFEVTSELx is ignored by the processor for the "FRONTEND_RETIRED" event.

The sub-event encodings supported by MSR_PEBS_FRONTEND.EVTSEL is given in Table Table 22-39. "FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL".

Table 22-39. FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL

Sub-Event Name	EVTSEL	Description
ANY_DSB_MISS	1H	Retired Instructions which experienced any decode stream buffer (DSB) miss.
DSB_MISS	11H	Retired Instructions which experienced a DSB miss that caused a fetch starvation cycle.
L1I_MISS	12H	The fetch of retired Instructions which experienced Instruction L1 Cache true miss ¹ . Additional requests to the same cache line as an in-flight L1I cache miss will not be counted.
L2_MISS	13H	The fetch of retired Instructions which experienced L2 Cache true miss. Additional requests to the same cache line as an in-flight MLC cache miss will not be counted.
ITLB_MISS	14H	The fetch of retired Instructions which experienced ITLB true miss. Additional requests to the same cache line as an in-flight ITLB miss will not be counted.
STLB_MISS	15H	The fetch of retired Instructions which experienced STLB true miss. Additional requests to the same cache line as an in-flight STLB miss will not be counted.
IDQ_READ_BUBBLES	6H	An IDQ read bubble is defined as any one of the 4 allocation slots of IDQ that is not filled by the front-end on any cycle where there is no back end stall. Using the threshold and latency fields in MSR_PEBS_FRONTEND allows counting of IDQ read bubbles of various magnitude and duration. Latency controls the number of cycles and Threshold controls the number of allocation slots that contain bubbles. The event counts if and only if a sequence of at least FE_LATENCY consecutive cycles contain at least FE_TRESHOLD number of bubbles each.

NOTES:

1. A true miss is the first miss for a cacheline/page (excluding secondary misses that fall into same cacheline/page).

The layout of MSR_PEBS_FRONTEND is given in Table Table 22-40. "MSR_PEBS_FRONTEND Layout".

Table 22-40. MSR_PEBS_FRONTEND Layout

Bit Name	Offset	Description
EVTSEL	7:0	Encodes the sub-event within FrontEnd_Retired that can use PEBS facility, see Table Table 22-39. "FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL".
IDQ_Bubble_Length	19:8	Specifies the threshold of continuously elapsed cycles for the specified width of bubbles when counting IDQ_READ_BUBBLES event.
IDQ_Bubble_Width	22:20	Specifies the threshold of simultaneous bubbles when counting IDQ_READ_BUBBLES event.
Reserved	63:23	Reserved

The FRONTEND_RETIRED event is designed to help software developers identify exact instructions that caused front-end issues. There are some instances in which the event will, by design, the under-counting scenarios include the following:

- The event counts only retired (non-speculative) front-end events, i.e., events from just true program execution path are counted.
- The event will count once per cacheline (at most). If a cacheline contains multiple instructions which caused front-end misses, the count will be only 1 for that line.
- If the multibyte sequence of an instruction spans across two cachelines and causes a miss it will be recorded once. If there were additional misses in the second cacheline, they will not be counted separately.
- If a multi-uop instruction exceeds the allocation width of one cycle, the bubbles associated with these uops will be counted once per that instruction.
- If 2 instructions are fused (macro-fusion), and either of them or both cause front-end misses, it will be counted once for the fused instruction.
- If a front-end (miss) event occurs outside instruction boundary (e.g., due to processor handling of architectural event), it may be reported for the next instruction to retire.

22.3.8.3 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 22.3.4.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table Table 22-41. "MSR_OFFCORE_RSP_x Request_Type Definition (Skylake, Kaby Lake, and Coffee Lake Microarchitectures)".
- Supplier information (bits 29:16): see Table Table 22-42. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)".
- Snoop response information (bits 37:30): see Table Table 22-43. "MSR_OFFCORE_RSP_x Snoop Info Field Definition (CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9E, 06_55H)".

**Table 22-41. MSR_OFFCORE_RSP_x Request_Type Definition
(Skylake, Kaby Lake, and Coffee Lake Microarchitectures)**

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count hw or sw prefetches.
DMND_RFO	1	Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
Reserved	14:3	Reserved
OTHER	15	Counts miscellaneous requests, such as I/O and uncacheable accesses.

Table Table 22-42. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)" lists the supplier information field that applies to 6th generation, 7th generation and 8th generation Intel Core processors. (6th generation Intel Core processor CUID signatures: 06_4EH and 06_5EH; 7th generation and 8th generation Intel Core processor CUID signatures: 06_8EH and 06_9EH).

**Table 22-42. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)**

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.

Table 22-42. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)

Subtype	Bit Name	Offset	Description
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	L4_HIT	22	L4 Cache (if L4 is present in the processor).
	Reserved	25:23	Reserved
	DRAM	26	Local Node
	Reserved	29:27	Reserved
	SPL_HIT	30	L4 cache super line hit (if L4 is present in the processor).

Table Table 22-43. "MSR_OFFCORE_RSP_x Snoop Info Field Definition (CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9E, 06_55H)" lists the snoop information field that apply to processors with CUID signatures 06_4EH, 06_5EH, 06_8EH, 06_9E, and 06_55H.

Table 22-43. MSR_OFFCORE_RSP_x Snoop Info Field Definition
(CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9E, 06_55H)

Subtype	Bit Name	Offset	Description
Snoop Info	SPL_HIT	30	L4 cache super line hit (if L4 is present in the processor).
	SNOOP_NONE	31	No details on snoop-related information.
	SNOOP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNOOP_MISS	33	A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitl was returned by all cores. -For LLC Miss, Rspl was returned by all sockets and data was returned from DRAM.
	SNOOP_HIT_NO_FWD	34	A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO). -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD). -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S). In the LLC Miss case, data is returned from DRAM.
	SNOOP_HIT_WITH_FWD	35	A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	SNOOP_HITM	36	A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD). -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO). -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	SNOOP_NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.

22.3.8.3.1 Off-core Response Performance Monitoring for the Intel® Xeon® Scalable Processor Family

The following tables list the requestor and supplier information fields that apply to the Intel® Xeon® Scalable Processor Family.

- Transaction request type encoding (bits 15:0): see Table 22-44.
- Supplier information (bits 29:16): see Table 22-45.
- Supplier information (bits 29:16) with support for Intel® Optane™ DC Persistent Memory support: see Table 22-46. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H, Steppings 0x5H - 0xFH)".
- Snoop response information has not been changed and is the same as in (bits 37:30): see Table 22-43.

Table 22-44. MSR_OFFCORE_RSP_x Request_Type Definition (Intel® Xeon® Scalable Processor Family)

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count hw or sw prefetches.
DEMAND_RFO	1	Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DEMAND_CODE_RD	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
Reserved	3	Reserved.
PF_L2_DATA_RD	4	Counts the number of prefetch data reads into L2.
PF_L2_RFO	5	Counts the number of RFO Requests generated by the MLC prefetches to L2.
Reserved	6	Reserved.
PF_L3_DATA_RD	7	Counts the number of MLC data read prefetches into L3.
PF_L3_RFO	8	Counts the number of RFO requests generated by MLC prefetches to L3.
Reserved	9	Reserved.
PF_L1D_AND_SW	10	Counts data cacheline reads generated by hardware L1 data cache prefetcher or software prefetch requests.
Reserved	14:11	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

Table 22-45. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H)" lists the supplier information field that applies to the Intel Xeon Scalable Processor Family (CPUID signature: 06_55H).

Table 22-45. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	SUPPLIER_NONE	17	No Supplier Information available.
	L3_HIT_M	18	M-state initial lookup stat in L3.
	L3_HIT_E	19	E-state
	L3_HIT_S	20	S-state
	L3_HIT_F	21	F-state
	Reserved	25:22	Reserved
	L3_MISS_LOCAL_DRAM	26	L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM.
	L3_MISS_REMOTE_HOP0_DRAM	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1_DRAM	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P_DRAM	29	Hop 2 or more Remote supplier.
	Reserved	30	Reserved

Table 22-46 lists the supplier information field that applies to the Intel Xeon Scalable Processor Family (CPUID signature: 06_55H, Steppings 0x5H - 0xFH).

Table 22-46. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H, Steppings 0x5H - 0xFH)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	SUPPLIER_NONE	17	No Supplier Information available.
	L3_HIT_M	18	M-state initial lookup stat in L3.
	L3_HIT_E	19	E-state
	L3_HIT_S	20	S-state
	L3_HIT_F	21	F-state
	LOCAL_PMM	22	Local home requests that were serviced by local PMM.
	REMOTE_HOP0_PMM	23	Hop 0 Remote supplier.
	REMOTE_HOP1_PMM	24	Hop 1 Remote supplier.
	REMOTE_HOP2P_PMM	25	Hop 2 or more Remote supplier.
	L3_MISS_LOCAL_DRAM	26	L3 Miss: Local home requests that missed the L3 cache and were serviced by local DRAM.
	L3_MISS_REMOTE_HOP0_DRAM	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1_DRAM	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P_DRAM	29	Hop 2 or more Remote supplier.
	Reserved	30	Reserved

22.3.8.4 Uncore Performance Monitoring Facilities on Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Cannon Lake microarchitecture introduces LLC support of up to six processor cores. To support six processor cores and eight LLC slices, existing MSRs have been rearranged and new CBo MSRs have been added. Uncore perfor-

mance monitoring software drivers from prior generations of Intel Core processors will need to update the MSR addresses. The new MSRs and updated MSR addresses have been added to the Uncore PMU listing in Section 2.17.2, “MSRs Specific to 8th Generation Intel® Core™ i3 Processors,” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

22.3.9 10th Generation Intel® Core™ Processor Performance Monitoring Facility

Some 10th generation Intel® Core™ processors and some 3rd generation Intel® Xeon® Scalable Processor Family are based on Ice Lake microarchitecture. Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture. For these processors, the core PMU supports architectural performance monitoring capability with version Id 5 (see Section 22.2.5) and a host of non-architectural monitoring capabilities.

The core PMU's capability is similar to those described in Section 22.3.1 through Section 22.3.8, with some differences and enhancements summarized in Table 22-47.

Table 22-47. Core PMU Summary of the Ice Lake Microarchitecture

Box	Ice Lake Microarchitecture	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Comment
Architectural PerfMon version	5	4	See Section 22.2.5.
Number of programmable counters per thread	8	4	Use CPUID to determine number of counters. See Section 22.2.1.
PEBS: Basic functionality	Yes	Yes	See Section 22.3.9.1.
PEBS record format encoding	0100b	0011b	See Section 22.6.2.4.2.
Extended PEBS	PEBS is extended to all Fixed and General Purpose counters and to all performance monitoring events.	No	See Section 22.9.1.
Adaptive PEBS	Yes	No	See Section 22.9.2.
Performance Metrics	Yes (4)	No	See Section 22.3.9.3.
PEBS-PDIR	IA32_FIXED0 only (Corresponding counter control MSRs must be enabled.)	IA32_PMC1 only.	

22.3.9.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 10th generation Intel Core processors provides a number of enhancements relative to PEBS in processors based on the Skylake, Kaby Lake, and Coffee Lake microarchitectures. Enhancement of the PEBS facility with Extended PEBS and Adaptive PEBS features is described in detail in Section 22.9.

The 3rd generation Intel Xeon Scalable Family of processors based on the Ice Lake microarchitecture introduce EPT-friendly PEBS. This allows EPT violations and other VM Exits to be taken on PEBS accesses to the DS Area. See Section 22.9.5 for details.

22.3.9.2 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 22.3.4.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 18-[N1].
- Response type encoding (bits 16-37) of

- Supplier information: see Table [18-N2].
- Snoop response information: see Table [18-N3].
- All transactions are tracked at cacheline granularity except some in request type OTHER.

**Table 22-48. MSR_OFFCORE_RSP_x Request_Type Definition
(Processors Based on Ice Lake Microarchitecture)**

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data and page table entry reads.
DEMAND_RFO	1	Counts demand read (RFO) and software prefetches (PREFETCHW) for exclusive ownership in anticipation of a write.
DEMAND_CODE_RD	2	Counts demand instruction fetches and instruction prefetches targeting the L1 instruction cache.
Reserved	3	Reserved
HWPF_L2_DATA_RD	4	Counts hardware generated data read prefetches targeting the L2 cache.
HWPF_L2_RFO	5	Counts hardware generated prefetches for exclusive ownership (RFO) targeting the L2 cache.
Reserved	6	Reserved
HWPF_L3	9:7 and 13 ¹	Counts hardware generated prefetches of any type targeting the L3 cache.
HWPF_L1D_AND_SWPF	10	Counts hardware generated data read prefetches targeting the L1 data cache and the following software prefetches (PREFETCHNTA, PREFETCHT0/1/2).
STREAMING_WR	11	Counts streaming stores.
Reserved	12	Reserved
Reserved	14	Reserved
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

NOTES:

1. All bits need to be set to 1 to count this type.

Ice Lake microarchitecture has added a new category of Response subtype, called a Combined Response Info. To count a feature in this type, all the bits specified must be set to 1.

A valid response type must be a non-zero value of the following expression:

Any | ['OR' of Combined Response Info Bits | [('OR' of Supplier Info Bits) & ('OR' of Snoop Info Bits)]]

If "ANY" bit[16] is set, other response type bits [17-39] are ignored.

Table 22-49 lists the supplier information field that applies to processors based on Ice Lake microarchitecture.

**Table 22-49. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(Processors Based on Ice Lake Microarchitecture)**

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Combined Response Info	DRAM	26, 31, 32 ¹	Requests that are satisfied by DRAM.
	NON_DRAM	26, 37 ¹	Requests that are satisfied by a NON_DRAM system component. This includes MMIO transactions.
	L3_MISS	22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37 ¹	Requests that were not supplied by the L3 Cache. The event includes some currently reserved bits in anticipation of future memory designs.

Table 22-49. MSR_OFFCORE_RSP_x Supplier Info Field Definition (Contd.)
(Processors Based on Ice Lake Microarchitecture)

Subtype	Bit Name	Offset	Description
Supplier Info	L3_HIT	18,19, 20 ¹	Requests that hit in L3 cache. Depending on the snoop response the L3 cache may have retrieved the cacheline from another core's cache.
Reserved		17, 21:25, 27:29	Reserved.

NOTES:

1. All bits need to be set to 1 to count this type.

Table 22-50 lists the snoop information field that applies to processors based on Ice Lake microarchitecture.

Table 22-50. MSR_OFFCORE_RSP_x Snoop Info Field Definition
(Processors Based on Ice Lake Microarchitecture)

Subtype	Bit Name	Offset	Description
Snoop Info	Reserved	30	Reserved.
	SNOOP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNOOP_MISS	33	A snoop was sent and none of the snooped caches contained the cacheline.
	SNOOP_HIT_NO_FWD	34	A snoop was sent and hit in at least one snooped cache. The unmodified cacheline was not forwarded back, because the L3 already has a valid copy.
	Reserved	35	Reserved.
	SNOOP_HITM	36	A snoop was sent and the cacheline was found modified in another core's caches. The modified cacheline was forwarded to the requesting core.

22.3.9.3 Performance Metrics

The Ice Lake core PMU provides built-in support for Top-down Microarchitecture Analysis (TMA) method level 1 metrics. These metrics are always available to cross-validate performance observations, freeing general purpose counters to count other events in high counter utilization scenarios. For more details about the method, refer to Top-Down Analysis Method chapter (Appendix B.1) of the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

A new MSR called MSR_PERF_METRICS reports the metrics directly. Software can check (and/or expose to its guests) the availability of the PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15). For additional details on this MSR, refer to Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

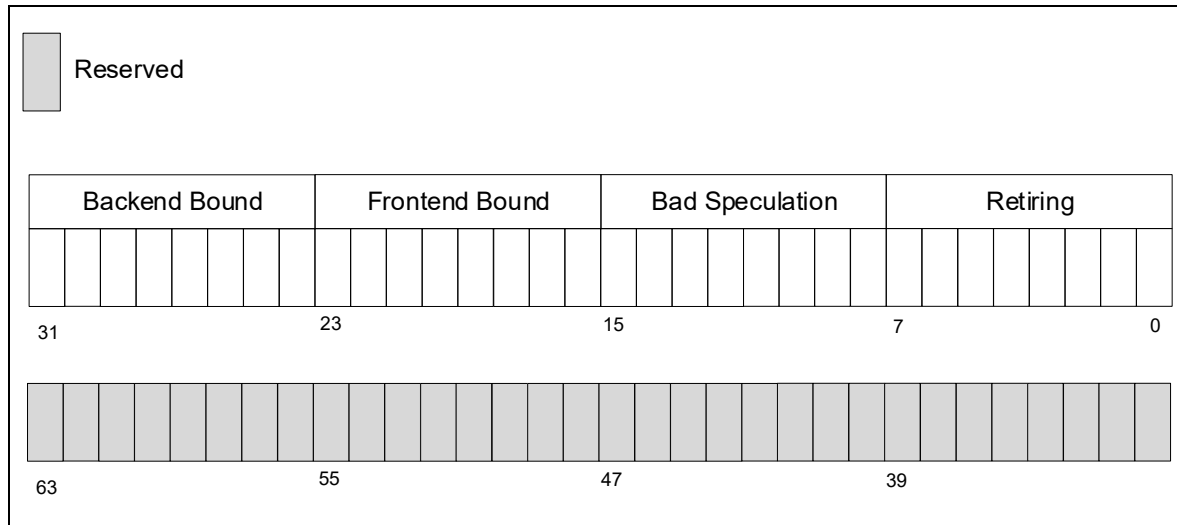


Figure 22-39. MSR_PERF_METRICS Definition

This register exposes the four TMA Level 1 metrics. The lower 32 bits are divided into four 8-bit fields, as shown by the above figure, each of which is an integer fraction of 255.

To support built-in performance metrics, new bits have been added to the following MSRs:

- IA32_PERF_GLOBAL_CTRL. EN_PERF_METRICS[48]: If this bit is set and fixed-function performance-monitoring counter 3 is enabled, built-in performance metrics are enabled.
- IA32_PERF_GLOBAL_STATUS_SET. SET_OVF_PERF_METRICS[48]: If this bit is set, it will set the status bit in the IA32_PERF_GLOBAL_STATUS register for PERF_METRICS.
- IA32_PERF_GLOBAL_STATUS_RESET. RESET_OVF_PERF_METRICS[48]: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for PERF_METRICS.
- IA32_PERF_GLOBAL_STATUS. OVF_PERF_METRICS[48]: If this bit is set, it indicates that a PERF_METRICS-related resource has overflowed and a PMI is triggered¹. If this bit is clear, no such overflow has occurred.

NOTE

Software has to synchronize, e.g., re-start, fixed-function performance-monitoring counter 3 as well as PERF_METRICS when either bit 35 or 48 in IA32_PERF_GLOBAL_STATUS is set. Otherwise, PERF_METRICS may return undefined values.

The values in MSR_PERF_METRICS are derived from fixed-function performance-monitoring counter 3. Software should start both registers, PERF_METRICS and fixed-function performance-monitoring counter 3, from zero. Additionally, software is recommended to periodically clear both registers in order to maintain accurate measurements for certain scenarios that involve sampling metrics at high rates.

In order to save/restore PERF_METRICS, software should follow these guidelines:

- PERF_METRICS and fixed-function performance-monitoring counter 3 should be saved and restored together.
- To ensure that PERF_METRICS and fixed-function performance-monitoring counter 3 remain synchronized, both should be disabled during both save and restore. Software should enable/disable them atomically, with a single write to IA32_PERF_GLOBAL_CTRL to set/clear both EN_PERF_METRICS[bit 48] and EN_FIXED_CTR3[bit 35].

1. An overflow of fixed-function performance-monitoring counter 3 should normally happen first if software follows Intel's recommendations.

- On state restore, fixed-function performance-monitoring counter 3 must be restored **before** PERF_METRICS, otherwise undefined results may be observed.

22.3.10 12th and 13th Generation Intel® Core™ Processors, and 4th and 5th Generation Intel® Xeon® Scalable Processor Family Performance Monitoring Facility

The 12th generation Intel® Core™ processor supports Alder Lake performance hybrid architecture. These processors offer a unique combination of Performance and Efficient-cores (P-core and E-core). The P-core is based on Golden Cove microarchitecture and the E-core is based on Gracemont microarchitecture. The 13th generation Intel® Core™ processor supports Raptor Lake performance hybrid architecture, utilizing both Raptor Cove cores and enhanced Gracemont cores. The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture utilizing Golden Cove cores. The 5th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture utilizing Raptor Cove cores. These processors all report architectural performance monitoring version ID = 5 and support non-architectural monitoring capabilities described in this section.

22.3.10.1 P-core Performance Monitoring Unit

The P-core PMU's capability is similar to those described in Section 22.3.1 through Section 22.3.9, with some differences and enhancements summarized in Table 22-51.

Table 22-51. Core PMU Summary of the Golden Cove Microarchitecture

Box	Golden Cove Microarchitecture	Ice Lake Microarchitecture	Comment
Architectural PerfMon version	5	5	See Section 22.2.5.
Event-Counter Restrictions	Simplified identification		Counters 4-7 support a subset of events. See Section 22.3.10.1.2.
Performance Metrics	Yes (12)	Yes (4)	See Section 22.3.9.3.
PEBS: Baseline, record format	Yes 0100b	Yes 0100b	See Section 22.3.9.
PEBS: EPT-friendly	Yes	No; debuts in Ice Lake server microarchitecture	See Section 22.6.2.4.2.
PEBS: Precise Distribution	IA32_FIXED0 instruction-granularity PDist on IA32_PMC0	IA32_FIXED0 cycle-granularity No PDist	See Section 22.9.6.
PEBS: Load Latency	Instruction latency Cache latency Access info fields (5)	Instruction latency Access info fields (3)	See Section 22.9.7.
PEBS: Store Latency	Cache latency Access info fields (3)	None	See Section 22.9.8.
PEBS: Intel TSX support	Abort info fields (9)	Abort info fields (8)	See Section 22.3.6.5.1. (Intel Xeon processor only feature.)

22.3.10.1.1 P-core Perf Metrics Extensions

For 12th generation Intel Core processor P-cores, the core PMU supports the built-in metrics that were introduced in the Ice Lake microarchitecture PMU. This core PMU extends the PERF_METRICS MSR to feature TMA method level 2 metrics, as shown in Figure 22-40.

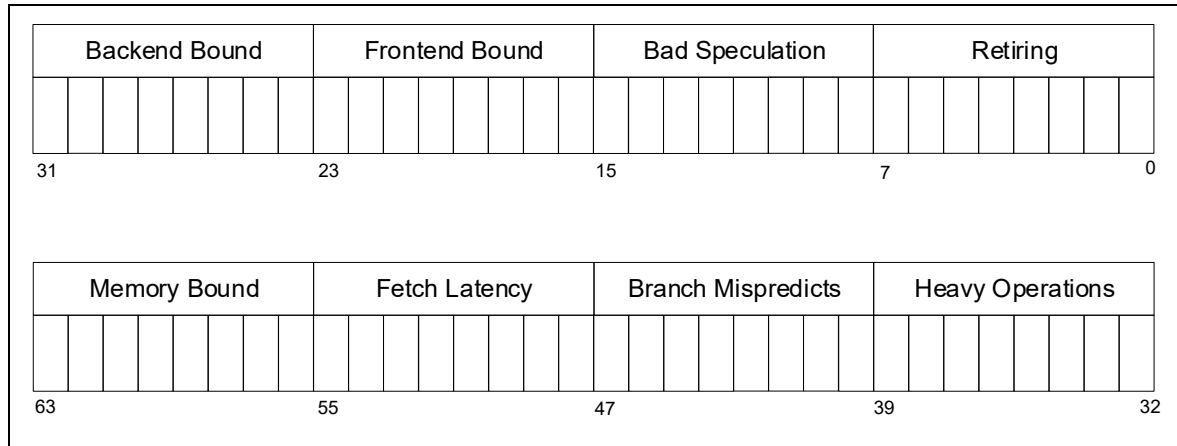


Figure 22-40. PERF_METRICS MSR Definition for 12th Generation Intel® Core™ Processor P-core

The lower half of the register is the TMA level 1 metrics (legacy). The upper half is also divided into four 8-bit fields, each of which is an integer fraction of 255. Additionally, each of the new level 2 metrics in the upper half is a subset of the corresponding level 1 metric in the lower half (that is, its parent node per the TMA hierarchy). This enables software to deduce the other four level 2 metrics by subtracting corresponding metrics as shown in Figure 22-41.

Light_Operations = Retiring - Heavy_Operations
 Machine_Clears = Bad_Speculation - Branch_Mispredicts
 Fetch_Bandwidth = Frontend_Bound - Fetch_Latency
 Core_Bound = Backend_Bound - Memory_Bound

Figure 22-41. Deducing Implied Level 2 Metrics in the Core PMU for 12th Generation Intel® Core™ Processor P-core

The PERF_METRICS MSR and fixed-function performance-monitoring counter 3 of the core PMU feature 12 metrics in total that cover all level 1 and level 2 nodes of the TMA hierarchy.

22.3.10.1.2 P-core Counter Restrictions Simplification

The 12th generation Intel Core processor P-core allows identification of performance monitoring events with counter restrictions based on event encodings. The general rule is: Event Codes < 0x90 are restricted to general-purpose performance-monitoring counters 0-3. Event Codes ≥ 0x90 are likely to have no restrictions. Table 22-52 lists the exceptions to this rule.

Table 22-52. Special Performance Monitoring Events with Counter Restrictions

Event Encoding ¹	Event Name	Counter Restriction
xx3C	CPU_CLK_UNHALTED.*	0-7 (No restriction for all architectural events.)
xx2E	LONGEST_LAT_CACHE.*	
xxDx	MEM_*_RETIRED.*	0-3
01A3, 02A3, 08A3	Some CYCLE_ACTIVITY sub-events	0-3
02CD	MEM_TRANS_RETIRED.STORE_SAMPLE	0

Table 22-52. Special Performance Monitoring Events with Counter Restrictions

Event Encoding ¹	Event Name	Counter Restriction
04A4	TOPDOWN.BAD_SPEC_SLOTS	0
08A4	TOPDOWN.BR_MISPREDICT_SLOTS	
xxCE	AMX_OPS_RETIRED	0

NOTES:

1. Linux perf rUUEE syntax, where UU is the Unit Mask field and EE is the Event Select (also known as Event Code) field in the IA32_PERFECTSELx MSRs.

22.3.10.1.3 P-core Off-core Response Facility

For the 12th generation Intel Core processor P-core, the Off-core Response (OCR) Facility is similar to that described in Section 22.3.9.2.

The following enhancements are introduced for the Request_Type of MSR_OFFCORE_RSP_x:

- WB (bits 3 and 12): Count writeback (modified or non-modified) transactions by core caches.
- HWPFL1D (bit 10): Counts hardware generated data read prefetches targeting the L1 data cache (only).
- SWPFLREAD (bit 14): Counts software generated data read prefetches by the PREFETCHNTA and PREFETCHT0/1/2 instructions.

22.3.10.2 E-core Performance Monitoring Unit

The core PMU capabilities on the 12th generation Intel Core processor E-core are summarized in Table 22-53 below.

Table 22-53. Core PMU Summary of the Gracemont Microarchitecture

Box	Gracemont Microarchitecture	Tremont Microarchitecture	Comment
Number of fixed-function performance-monitoring counters per core	3	3	Use CPUID to enumerate number of counters. See Section 22.2.1.
Number of general-purpose counters per core	6	4	Use CPUID to enumerate number of counters. See Section 22.2.1.
Architectural Performance Monitoring version ID	5	5	See Section 22.2.5.
PEBS record format encoding	0100b	0100b	See Section 22.5.5.
EPT-friendly PEBS support	Yes	No	See Section 22.9.5.
Extended PEBS	Yes	Yes	See Section 22.9.1.
Adaptive PEBS	Yes	Yes	See Section 22.9.2.
Precise distribution (PDist) PEBS	IA32_PMC0 and IA32_FIXED_CTR0	IA32_PMC0 and IA32_FIXED_CTR0	PDist eliminates skid, see Section 22.9.3, Section 22.9.4, and Section 22.9.6.
PEBS Latency	Load and Store Latency	No	See Section 22.3.10.2.1, Section 22.3.10.2.2, Section 22.9.7, and Section 22.9.8.
PEBS Output	DS Save Area or Intel® Processor Trace	DS Save Area or Intel® Processor Trace	See Section 22.5.5.2.1.

Table 22-53. Core PMU Summary of the Gracemont Microarchitecture (Contd.)

Box	Gracemont Microarchitecture	Tremont Microarchitecture	Comment
Offcore Response	MSR 01A6H and 01A7H, each core has its own register, extended request and response types.	MSR 1A6H and 1A7H, each core has its own register, extended request and response types.	See Section 22.5.5.4.

22.3.10.2.1 E-core PEBS Load Latency

The 12th generation Intel Core processor E-core includes PEBS Load Latency support similar to that described in Section 22.9.7.

When a programmable counter is configured to count MEM_UOPS_RETIRED.LOAD_LATENCY_ABOVE_THRESHOLD (IA32_PERFVTSELx[15:0] = 0xD005, with CMASK=0 and INV=0), selected load operations whose latency exceeds the threshold provided in MSR_PEBS_LD_LAT_THRESHOLD (MSR 03F6H) will be counted. If a PEBS record is generated on overflow of this counter, the Memory Access Latency and Memory Auxiliary Info data is reported in the Memory Access Info group (Section 22.9.2.2.2). The formats of these fields are shown in Table 22-54 and Table 22-98.

Table 22-54. E-core PEBS Memory Access Info Encoding

Bit(s)	Field	Description
3:0	Data Source	The source of the data; see Table 22-55.
4	Lock	0: The operation was not part of a locked transaction. 1: The operation was part of a locked transaction.
5	STLB_MISS	0: The load did not miss the STLB (hit the DTLB or STLB). 1: The load missed the STLB.
6	ST_FWD_BLK	0: Load did not get a store forward block. 1: Load got a store forward block.
63:7	Reserved	Reserved

For details on E-core PEBS memory access latency encoding, see the Access Latency Field in Table 22-98.

Table 22-55. E-core PEBS Data Source Encodings

Encoding	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request) and MMIO. Memory mapped I/O hit.
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. Outstanding core cache miss to same cache-line address was already underway. (Pending core cache hit.)
03H	L2 HIT. This request was satisfied by the L2 cache.
04H	L3 HIT. Local or Remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
05H	L3 HITE. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found (clean).
06H	L3 HITM. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where a modified copy was found.
07H	Reserved.
08H	L3 HITF. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where a shared or forwarding copy was found.
09H	Reserved.

Table 22-55. E-core PEBS Data Source Encodings (Contd.)

Encoding	Description
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	Reserved.
0CH	Reserved.
0DH	Reserved.
0EH	I/O. Request of input/output operation.
0FH	The request was to uncacheable memory.

22.3.10.2.2 E-core PEBS Store Latency

The 12th generation Intel Core processor E-core includes PEBS Store Latency support. When a programmable counter is configured to count MEM_UOPS_RETIRED.STORE_LATENCY (IA32_PERFVTSELx[15:0] = 0xD006, with CMASK=0 and INV=0), all store operations will be counted. If a PEBS record is generated on overflow of this counter, the Memory Access Latency and Memory Auxiliary Info data is reported in the Memory Access Info group (Section 22.9.2.2.2). The formats of these fields are shown in Table 22-54 and Table 22-98.

22.3.10.2.3 E-core Precise Distribution (PDist) Support

The 12th generation Intel Core processor E-core supports PEBS with Precise Distribution (PDist) on IA32_PMC0 and IA32_FIXED_CTR0. All precise events support PDist save for UOPS_RETIRED. See Section 22.9.6 for additional details on PDist.

22.3.10.2.4 E-core Enhanced Off-core Response

Event number 0B7H support off-core response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. There are unique pairs of MSR_OFFCORE_RSPx registers per core. The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as follows:

- Bits 15:0 and bits 49:44 specify the request type of a transaction request to the uncore. This is described in Table 22-56.
- Bits 30:16 specify Response Type information or an L2 Hit, and is described in Table 22-79.
- If L2 misses, then bits 37:31 can be used to specify snoop response information and is described in Table 22-80.
- For outstanding requests, bit 38 can enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously; see Section 22.5.2.3 for details.

Table 22-56. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data reads.
DEMAND_RFO	1	Counts all demand reads for ownership (RFO) requests and software based prefetches for exclusive ownership (prefetchw).
DEMAND_CODE_RD	2	Counts demand instruction fetches and L1 instruction cache prefetches.
COREWB_M	3	Counts modified write backs from L1 and L2.
HWPf_L2_DATA_RD	4	Counts prefetch (that bring data to L2) data reads.
HWPf_L2_RFO	5	Counts all prefetch (that bring data to L2) RFOs.
HWPf_L2_CODE_RD	6	Counts all prefetch (that bring data to MLC only) code reads.
HWPf_L3_DATA_RD	7	Counts L3 cache hardware prefetch data reads (written to the L3 cache only).

Table 22-56. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
HWPf_L3_RFO	8	Counts L3 cache hardware prefetch RFOs (written to the L3 cache only) .
HWPf_L3_CODE_RD	9	Counts L3 cache hardware prefetch code reads (written to the L3 cache only).
HWPf_L1D_AND_SWPF	10	Counts L1 data cache hardware prefetch requests, read for ownership prefetch requests and software prefetch requests (except prefetchw).
STREAMING_WR	11	Counts all streaming stores.
COREWB_NONM	12	Counts non-modified write backs from L2.
RSVD	14:13	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O accesses that have any response type.
UC_RD	44	Counts uncached memory reads (PRd, UCRdF).
UC_WR	45	Counts uncached memory writes (WiL).
PARTIAL_STREAMING_WR	46	Counts partial (less than 64 byte) streaming stores (WCiL).
FULL_STREAMING_WR	47	Counts full, 64 byte streaming stores (WCiLF).
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

22.3.10.3 Unhalted Reference Cycles

The Unhalted Reference Cycles architectural performance monitoring event is enhanced to count at TSC-rate in the 12th generation Intel Core processor P-core when used on a general-purpose PMC. This enhancement makes it consistent with the fixed-function counter 2 and the E-core. As a result, this event is kept enumerated in CPUID.0AH.EBX (unlike prior hybrid parts).

22.3.11 Intel® Series 2 Core™ Ultra Processor Performance Monitoring Facility

The Intel® Series 2 Core™ Ultra processor supports Lunar Lake performance hybrid architecture. This processor offers a combination of Performance and Efficient-cores (P-core and E-core). The P-core is based on Lion Cove microarchitecture and the E-core is based on Skymont microarchitecture. This processor reports architectural performance monitoring version ID = 6 and supports non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 6 capabilities are described in Section 22.2.6.

22.3.11.1 P-core Performance Monitoring Unit

The core PMU capabilities on the Intel Series 2 Core Ultra processor P-core are similar to those described in Section 22.3.1 through Section 22.3.10, with some differences and enhancements summarized in Table 22-57.

Table 22-57. Core PMU Summary of the Lion Cove Microarchitecture

Box	Lion Cove Microarchitecture	Golden Cove and Redwood Cove Microarchitectures	Comment
Architectural Performance Monitoring version ID	6	5	See Section 22.2.6.
Number of general-purpose counters per core	10	8	Use CPUID to enumerate number of counters. See Section 22.2.1 and Section 22.2.9.

Table 22-57. Core PMU Summary of the Lion Cove Microarchitecture (Contd.)

Box	Lion Cove Microarchitecture	Golden Cove and Redwood Cove Microarchitectures	Comment
Number of architectural performance-monitoring events	12	8 (Golden Cove) 11 (Redwood Cove)	See Section 22.2.7.
Event-Counter Restrictions	Mostly homogeneous general counters.	Simplified identification of events supported on counters 4-7.	Few counter restrictions may apply; see Section 22.3.1.1.1.
Performance Metrics	12 metrics Metrics clear mode or read-only.	12 metrics Read-only.	See the RDPNC instruction in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B.
OCR: MSR_OFFCORE_RSP_0/1	0FFF FFFF FFFFH	003F FFFF FFFFH	See Section 2.17.9 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.
PEBS: Baseline	Yes	Yes	See Section 22.8.
PEBS record format encoding	0110b	0101b	See Section 22.9.2.2.
PEBS: Precise Distribution	IA32_FIXED0 instruction-granularity. PDist on IA32_PMC0 and IA32_PMC1.	IA32_FIXED0 instruction-granularity. PDist on IA32_PMC0.	See Section 22.9.6.
PEBS: Data Source field	5-bits	4-bits	See Section 22.9.7.
LBR: Event Logging	Yes	No	See Section 21.1.3.6.
Intel PT: TNT Disable	Yes	No	See Chapter 36 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

22.3.11.1.1 P-core Homogeneous General Counters

The Lion Cove PMU enhances general-counters to support most of the performance monitoring events. The remaining events that do have counter restrictions are summarized in next Table 22-58.

Table 22-58. Performance Monitoring Events with Counter Restrictions in Lion Cove PMU

Event Encoding ¹	Event Name	Counter Restriction
xx20	OFFCORE_REQUESTS_OUTSTANDING.*	0-3
0148	L1D_PENDING.*	2
0175	INST_DECODED.DECODERS	2
08A3, 0CA3	CYCLE_ACTIVITY.*_L1D_MISS	2
04A4, 08A4, 10A4	TOPDOWN.BAD_SPEC_SLOTS, TOPDOWN.BR_MISPREDICT_SLOTS, TOPDOWN.MEMORY_BOUND_SLOTS	0
01B1	UOPS_EXECUTED.*	3
xxDx	MEM_INST_RETIRED.*, MEM_LOAD*_RETIRED.*	0-3
02CD	MEM_TRANS_RETIRED.STORE_SAMPLE	0-1

NOTES:

1. Linux perf rUUEE syntax, where UU is the Unit Mask field and EE is the Event Select (also known as Event Code) field in the IA32_PERFVTSELx MSRs.

22.3.11.2 E-core Performance Monitoring Unit

Skymont microarchitecture performance monitoring capabilities are similar to Crestmont microarchitecture capabilities, with the following extensions:

- Support for fixed counters 4, 5, and 6 (see Section 22.2.9.4)
- Architecturally defined events: TMA L1 and LBR Inserts (see Section 22.2.9.6 and Section 21.1.3.6)
- PEBS Counter Snapshotting (see Section 22.9.10)
- Auto Counter Reload (see Section 22.11)

The core PMU capabilities on the Intel Series 2 Core Ultra processor E-core are summarized in Table 22-59.

Table 22-59. Core PMU Summary of the Skymont Microarchitecture

Box	Skymont Microarchitecture	Crestmont Microarchitecture	Comment
Architectural Performance Monitoring version ID	6	5	See Section 22.2.6.
Number of fixed-function performance-monitoring counters per core	6 (0, 1, 2, 4, 5, 6)	3 (0, 1, 2)	Use CPUID to enumerate number of counters. See Section 22.2.1 and Section 22.2.9.4.
Number of general-purpose counters per core	8	8	Use CPUID to enumerate number of counters. See Section 22.2.1.
PEBS record format encoding	0110b	0101b	See Section 22.3.10.
Auto Counter Reload (ACR)	Yes	No	See Section 22.11.

22.4 PERFORMANCE MONITORING (INTEL® XEON™ PHI PROCESSORS)

NOTE

This section also applies to the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture.

22.4.1 Intel® Xeon Phi™ Processor 7200/5200/3200 Performance Monitoring

The Intel® Xeon Phi™ processor 7200/5200/3200 series are based on the Knights Landing microarchitecture. The performance monitoring capabilities are distributed between its tiles (pair of processor cores) and untile (connecting many tiles in a physical processor package). Functional details of the tiles and untile of the Knights Landing microarchitecture can be found in Chapter 16 of Intel® 64 and IA-32 Architectures Optimization Reference Manual.

A complete description of the tile and untile PMU programming interfaces for Intel Xeon Phi processors based on the Knights Landing microarchitecture can be found in the Technical Document section at <http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html>.

A tile contains a pair of cores attached to a shared L2 cache and is similar to those found in Intel Atom® processors based on the Silvermont microarchitecture. The processor provides several new capabilities on top of the Silvermont performance monitoring facilities.

The processor supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural performance monitoring capabilities. The processor provides two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

Non-architectural performance monitoring in the processor also uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3. The processor supports AnyThread counting in three architectural performance monitoring events.

22.4.1.1 Enhancements of Performance Monitoring in the Intel® Xeon Phi™ Processor Tile

The Intel® Xeon Phi™ processor tile includes the following enhancements to the Silvermont microarchitecture.

- AnyThread support. This facility is limited to following three architectural events: Instructions Retired, Unhalted Core Cycles, Unhalted Reference Cycles using IA32_FIXED_CTR0-2 and Unhalted Core Cycles, Unhalted Reference Cycles using IA32_PERFEVTSELx.
- PEBS-DLA (Processor Event-Based Sampling-Data Linear Address) fields. The processor provides memory address in addition to the Silvermont PEBS record support on select events. The PEBS recording format as reported by IA32_PERF_CAPABILITIES [11:8] is 2.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor tile to subsystems outside the tile (untile). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx. Two cores do not share the off-core response MSRs. Knights Landing expands off-core response capability to match the processor untile changes.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests. This facility is updated to match the processor untile changes.

22.4.1.1.1 Processor Event-Based Sampling

The processor supports processor event based sampling (PEBS). PEBS is supported using IA32_PMC0 (see also Section 20.4.9, “BTS and DS Save Area”).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 22.6.2.4).

The list of PEBS events supported in the processor is shown in the following table.

Table 22-60. PEBS Performance Events for Knights Landing Microarchitecture

Event Name	Event Select	Sub-event	UMask	Data Linear Address Support
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H	No
		JCC	7EH	No
		TAKEN_JCC	FEH	No
		CALL	F9H	No
		REL_CALL	FDH	No
		IND_CALL	FBH	No
		NON_RETURN_IND	EBH	No
		FAR_BRANCH	BFH	No
		RETURN	F7H	No

Table 22-60. PEBS Performance Events for Knights Landing Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask	Data Linear Address Support
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H	No
		JCC	7EH	No
		TAKEN_JCC	FEH	No
		IND_CALL	FBH	No
		NON_RETURN_IND	EBH	No
		RETURN	F7H	No
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H	Yes
		L2_MISS_LOADS	04H	Yes
		DLTB_MISS_LOADS	08H	Yes
RECYCLEQ	03H	LD_BLOCK_ST_FORWARD	01H	Yes
		LD_SPLITS	08H	Yes

The PEBS record format 2 supported by processors based on the Knights Landing microarchitecture is shown in Table Table 22-61. "PEBS Record Format for Knights Landing Microarchitecture", and each field in the PEBS record is 64 bits long.

Table 22-61. PEBS Record Format for Knights Landing Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	PSDLA
40H	R/EBP	A0H	Reserved
48H	R/ESP	A8H	Reserved
50H	R8	B0H	EventingRIP
58H	R9	B8H	Reserved

22.4.1.1.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table Table 22-62. "OffCore Response Event Encoding" lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 22-62. OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMCO-1	B7H	01H	MSR_OFFCORE_RSP0 (address 1A6H)
PMCO-1	B7H	02H	MSR_OFFCORE_RSP1 (address 1A7H)

Some of the MSR_OFFCORE_RESP [0,1] register bits are not valid in this processor and their use is reserved. The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 registers are defined in Table Table 22-63. "Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers". Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 22.5.2.3 for details.

Table 22-63. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers

Main	Sub-field	Bit	Name	Description
Request Type		0	DEMAND_DATA_RD	Demand cacheable data and L1 prefetch data reads.
		1	DEMAND_RFO	Demand cacheable data writes.
		2	DEMAND_CODE_RD	Demand code reads and prefetch code reads.
		3	Reserved	Reserved.
		4	Reserved	Reserved.
		5	PF_L2_RFO	L2 data RFO prefetches (includes PREFETCHW instruction).
		6	PF_L2_CODE_RD	L2 code HW prefetches.
		7	PARTIAL_READS	Partial reads (UC or WC).
		8	PARTIAL_WRITES	Partial writes (UC or WT or WP). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		9	UC_CODE_READS	UC code reads.
		10	BUS_LOCKS	Bus locks and split lock requests.
		11	FULL_STREAMING_STORES	Full streaming stores (WC). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		12	SW_PREFETCH	Software prefetches.
		13	PF_L1_DATA_RD	L1 data HW prefetches.
		14	PARTIAL_STREAMING_STORES	Partial streaming stores (WC). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		15	ANY_REQUEST	Account for any requests.
Response Type	Any	16	ANY_RESPONSE	Account for any response.
	Data Supply from Untile	17	NO_SUPP	No Supplier Details.
		18	Reserved	Reserved.
		19	L2_HIT_OTHER_TILE_NEAR	Other tile L2 hit E Near.
		20	Reserved	Reserved.
		21	MCDRAM_NEAR	MCDRAM Local.
		22	MCDRAM_FAR_OR_L2_HIT_OTHER_TILE_FAR	MCDRAM Far or Other tile L2 hit far.
		23	DRAM_NEAR	DRAM Local.
		24	DRAM_FAR	DRAM Far.

Table 22-63. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers (Contd.)

Main	Sub-field	Bit	Name	Description
	Data Supply from within same tile	25	L2_HITM_THIS_TILE	M-state.
		26	L2_HITE_THIS_TILE	E-state.
		27	L2_HITS_THIS_TILE	S-state.
		28	L2_HITF_THIS_TILE	F-state.
		29	Reserved	Reserved.
		30	Reserved	Reserved.
	Snoop Info; Only Valid in case of Data Supply from Untile	31	SNOOP_NONE	None of the cores were snooped.
		32	NO_SNOOP_NEEDED	No snoop was needed to satisfy the request.
		33	Reserved	Reserved.
		34	Reserved	Reserved.
		35	HIT_OTHER_TILE_FWD	Snoop request hit in the other tile with data forwarded.
		36	HITM_OTHER_TILE	A snoop was needed and it HitM-ed in other core's L1 cache. HitM denotes a cache-line was in modified state before effect as a result of snoop.
		37	NON_DRAM	Target was non-DRAM system address. This includes MMIO transactions.
Outstanding requests	Weighted cycles	38	OUTSTANDING (Valid only for MSR_OFFCORE_RESP0. Should only be used on PMCO. This bit is reserved for MSR_OFFCORE_RESP1).	If set, counts total number of weighted cycles of any outstanding offcore requests with data response. Valid only for OFFCORE_RESP_0 event. Should only be used on PMCO. This bit is reserved for OFFCORE_RESP_1 event.

22.4.1.1.3 Average Offcore Request Latency Measurement

Measurement of average latency of offcore transaction requests can be enabled using MSR_OFFCORE_RSP0.[bit 38] with the choice of request type specified in MSR_OFFCORE_RSP0.[bit 15:0].

Refer to Section 22.5.2.3, "Average Offcore Request Latency Measurement," for typical usage. Note that MSR_OFFCORE_RESPx registers are not shared between cores in Knights Landing. This allows one core to measure average latency while other core is measuring different offcore response events.

22.5 PERFORMANCE MONITORING (INTEL ATOM® PROCESSORS)

22.5.1 Performance Monitoring (45 nm and 32 nm Intel Atom® Processors)

45 nm and 32 nm Intel Atom processors report architectural performance monitoring versionID = 3 (supporting the aggregate capabilities of versionID 1, 2, and 3; see Section 22.2.3) and a host of non-architectural monitoring capabilities. These 45 nm and 32 nm Intel Atom processors provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

NOTE

The number of counters available to software may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may

not expose all counters. CPUID.0AH:EAX[15:8] reports the MSRs available to software; see Section 22.2.1.

Non-architectural performance monitoring in Intel Atom processor family uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

Architectural and non-architectural performance monitoring events in 45 nm and 32 nm Intel Atom processors support thread qualification using bit 21 (AnyThread) of IA32_PERFEVTSELx MSR, i.e., if IA32_PERFEVTSELx.AnyThread = 1, event counts include monitored conditions due to either logical processors in the same processor core.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3.

Valid event mask (Umask) bits can be found at: <https://perfmon-events.intel.com/>. The UMASK field may contain sub-fields that provide the same qualifying actions like those listed in Table Table 22-81. "Core Specificity Encoding within a Non-Architectural Umask", Table Table 22-82. "Agent Specificity Encoding within a Non-Architectural Umask", Table Table 22-83. "HW Prefetch Qualification Encoding within a Non-Architectural Umask", and Table Table 22-84. "MESI Qualification Definitions within a Non-Architectural Umask". One or more of these sub-fields may apply to specific events on an event-by-event basis. Precise Event Based Monitoring is supported using IA32_PMC0 (see also Section 20.4.9, "BTS and DS Save Area").

22.5.2 Performance Monitoring for Silvermont Microarchitecture

Intel processors based on the Silvermont microarchitecture report architectural performance monitoring versionID = 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities. Intel processors based on the Silvermont microarchitecture provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2). Intel Atom processors based on the Airmont microarchitecture support the same performance monitoring capabilities as those based on the Silvermont microarchitecture.

Non-architectural performance monitoring in the Silvermont microarchitecture uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3. Architectural and non-architectural performance monitoring events in the Silvermont microarchitecture ignore the AnyThread qualification regardless of its setting in IA32_PERFEVTSELx MSR.

22.5.2.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- The width of counter reported by CPUID.0AH:EAX[23:16] is 40 bits.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests.

22.5.2.1.1 Processor Event Based Sampling (PEBS)

In the Silvermont microarchitecture, the PEBS facility can be used with precise events. PEBS is supported using IA32_PMC0 (see also Section 20.4.9).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 22.6.2.4).

The list of precise events supported in the Silvermont microarchitecture is shown in Table Table 22-64. "PEBS Performance Events for the Silvermont Microarchitecture".

Table 22-64. PEBS Performance Events for the Silvermont Microarchitecture

Event Name	Event Select	Sub-event	UMask
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		FAR_BRANCH	BFH
		RETURN	F7H
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H
		L2_MISS_LOADS	04H
		DLTB_MISS_LOADS	08H
		HITM	20H
REHABQ	03H	LD_BLOCK_ST_FORWARD	01H
		LD_SPLITS	08H

PEBS Record Format The PEBS record format supported by processors based on the Intel Silvermont microarchitecture is shown in Table Table 22-65. "PEBS Record Format for the Silvermont Microarchitecture", and each field in the PEBS record is 64 bits long.

Table 22-65. PEBS Record Format for the Silvermont Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	Reserved
40H	R/EBP	A0H	Reserved
48H	R/ESP	A8H	Reserved
50H	R8	B0H	EventingRIP
58H	R9	B8H	Reserved

22.5.2.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table 22-66. "OffCore Response Event Encoding" lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

In the Silvermont microarchitecture, each MSR_OFFCORE_RSPx is shared by two processor cores.

Table 22-66. OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-1	B7H	01H	MSR_OFFCORE_RSP0 (address 1A6H)
PMC0-1	B7H	02H	MSR_OFFCORE_RSP1 (address 1A7H)

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are shown in Figure 22-42 and Figure 22-43. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 22.5.2.3 for details.

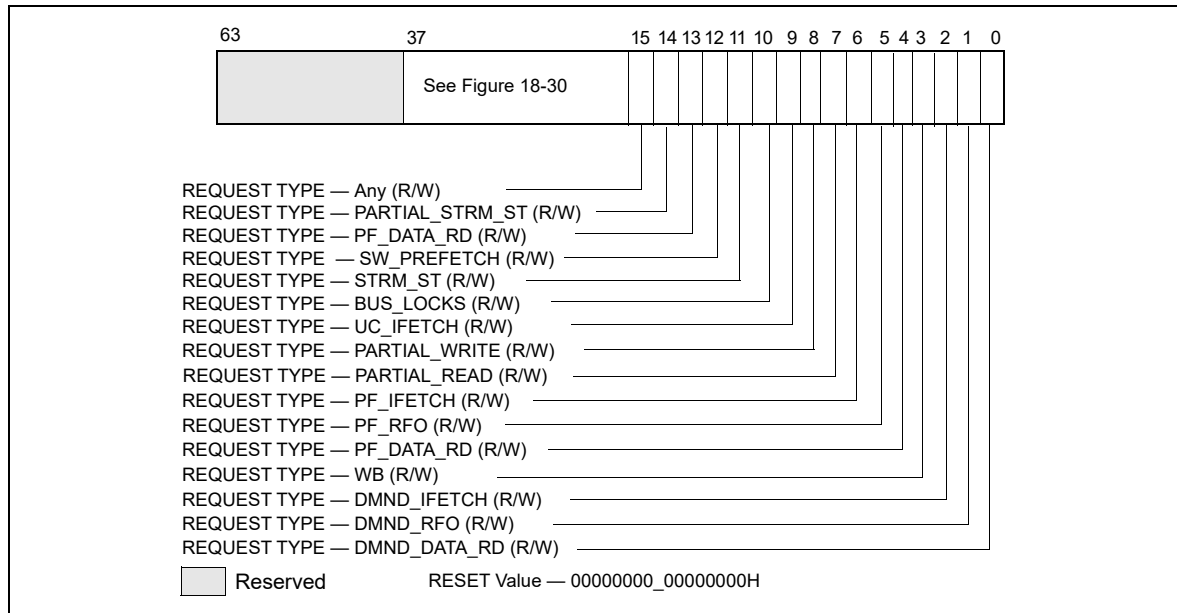


Figure 22-42. Request_Type Fields for MSR_OFFCORE_RSPx

Table 22-67. MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PARTIAL_READ	7	Counts the number of demand reads of partial cache lines (including UC and WC).
PARTIAL_WRITE	8	Counts the number of demand RFO requests to write to partial cache lines (includes UC, WT, and WP).
UC_IFETCH	9	Counts the number of UC instruction fetches.
BUS_LOCKS	10	Bus lock and split lock requests.
STRM_ST	11	Streaming store requests.
SW_PREFETCH	12	Counts software prefetch requests.
PF_DATA_RD	13	Counts DCU hardware prefetcher data read requests.
PARTIAL_STRM_ST	14	Streaming store requests.
ANY	15	Any request that crosses IDI, including I/O.

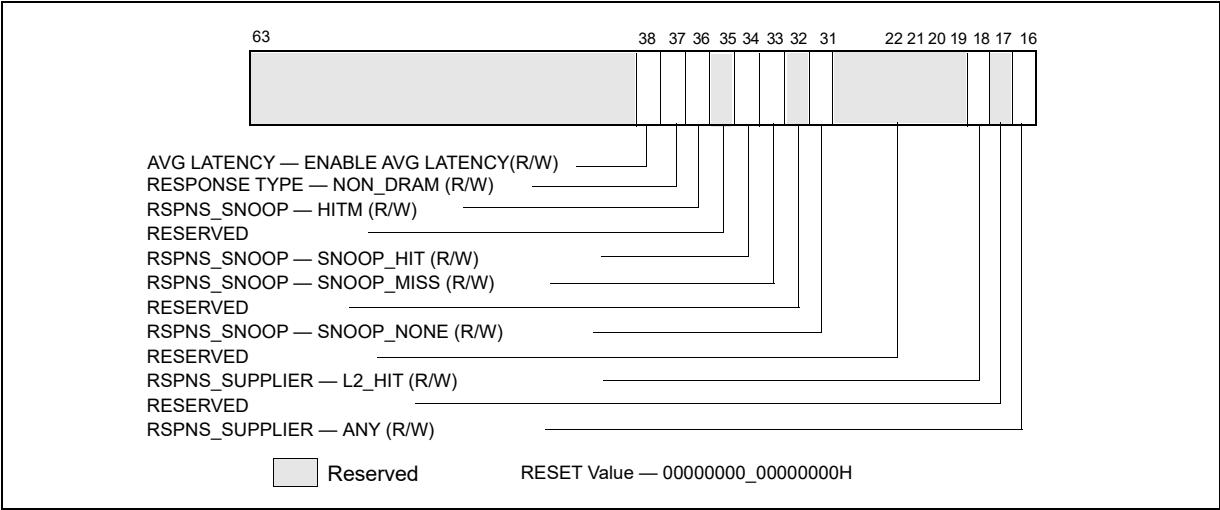


Figure 22-43. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSPx

To properly program this extra register, software must set at least one request type bit (Table Table 22-67. "MSR_OFFCORE_RSPx Request_Type Field Definition") and a valid response type pattern (Table Table 22-68. "MSR_OFFCORE_RSP_x Response Supplier Info Field Definition", Table Table 22-69. "MSR_OFFCORE_RSPx Snoop Info Field Definition"). Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 22-68. MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	ANY_RESPONSE	16	Catch all value for any response types.
Supplier Info	Reserved	17	Reserved
	L2_HIT	18	Cache reference hit L2 in either M/E/S states.
	Reserved	30:19	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY” bit is set, the supplier and snoop info bits are ignored.

Table 22-69. MSR_OFFCORE_RSPx Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	No details on snoop-related information.
	Reserved	32	Reserved
	SNOOP_MISS	33	Counts the number of snoop misses when L2 misses.
	SNOOP_HIT	34	Counts the number of snoops hit in the other module where no modified copies were found.

Table 22-69. MSR_OFFCORE_RSPx Snoop Info Field Definition (Contd.)

Subtype	Bit Name	Offset	Description
	Reserved	35	Reserved
	HITM	36	Counts the number of snoops hit in the other module where modified copies were found in other core's L1 cache.
	NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.
	AVG_LATENCY	38	Enable average latency measurement by counting weighted cycles of outstanding offcore requests of the request type specified in bits 15:0 and any response (bits 37:16 cleared to 0). This bit is available in MSR_OFFCORE_RESP0. The weighted cycles is accumulated in the specified programmable counter IA32_PMCx and the occurrence of specified requests are counted in the other programmable counter.

22.5.2.3 Average Offcore Request Latency Measurement

Average latency for offcore transactions can be determined by using both MSR_OFFCORE_RSP registers. Using two performance monitoring counters, program the two OFFCORE_RESPONSE event encodings into the corresponding IA32_PERFVTSELx MSRs. Count the weighted cycles via MSR_OFFCORE_RSP0 by programming a request type in MSR_OFFCORE_RSP0.[15:0] and setting MSR_OFFCORE_RSP0.OUTSTANDING[38] to 1, while setting the remaining bits to 0. Count the number of requests via MSR_OFFCORE_RSP1 by programming the same request type from MSR_OFFCORE_RSP0 into MSR_OFFCORE_RSP1[bit 15:0], and setting MSR_OFFCORE_RSP1.ANY_RESPONSE[16] = 1, while setting the remaining bits to 0. The average latency can be obtained by dividing the value of the IA32_PMCx register that counted weight cycles by the register that counted requests.

22.5.3 Performance Monitoring for Goldmont Microarchitecture

Intel Atom processors based on the Goldmont microarchitecture report architectural performance monitoring versionID = 4 (see Section 22.2.4) and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 22.2.4.

The bit fields (except bit 21) within each IA32_PERFVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3. The Goldmont microarchitecture does not support Hyper-Threading and thus architectural and non-architectural performance monitoring events ignore the AnyThread qualification regardless of its setting in the IA32_PERFVTSELx MSR. However, Goldmont does not set the AnyThread deprecation bit (CPUID.0AH:EDX[15]).

The core PMU's capability is similar to that of the Silvermont microarchitecture described in Section 22.5.2, with some differences and enhancements summarized in Table 22-70. "Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures".

Table 22-70. Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	Goldmont Microarchitecture	Silvermont Microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	4	2	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:40, W:32	See Section 22.2.2.
Architectural Performance Monitoring version ID	4	3	Use CPUID to determine # of counters. See Section 22.2.1.

Table 22-70. Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	Goldmont Microarchitecture	Silvermont Microarchitecture	Comment
PMI Overhead Mitigation	- Freeze_PerfMon_on_PMI with streamlined semantics. - Freeze_LBR_on_PMI with streamlined semantics for branch profiling.	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling.	See Section 20.4.7. Legacy semantics not supported with version 4 or higher.
Counter and Buffer Overflow Status Management	- Query via IA32_PERF_GLOBAL_STATUS - Reset via IA32_PERF_GLOBAL_STATUS_RESET - Set via IA32_PERF_GLOBAL_STATUS_SET	- Query via IA32_PERF_GLOBAL_STATUS - Reset via IA32_PERF_GLOBAL_OVF_CTRL	See Section 22.2.4.
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/Overhead/Interference	- Individual counter overflow - PEBS buffer overflow - ToPA buffer overflow - CTR_Frz, LBR_Frz	- Individual counter overflow - PEBS buffer overflow	See Section 22.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	- CTR_Frz, - LBR_Frz	No	See Section 22.2.4.1.
PerfMon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	No	See Section 22.2.4.3.
Processor Event Based Sampling (PEBS) Events	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 22-71. "Precise Events Supported by the Goldmont Microarchitecture".	See Section 22.5.2.1.1. General-Purpose Counter 0 only. Only supports precise events (see Table 22-64. "PEBS Performance Events for the Silvermont Microarchitecture").	IA32_PMC0 only.
PEBS record format encoding	0011b	0010b	
Reduce skid PEBS	IA32_PMC0 only	No	
Data Address Profiling	Yes	No	
PEBS record layout	Table 22-72. "PEBS Record Format for the Goldmont Microarchitecture"; enhanced fields at offsets 90H- 98H; and TSC record field at COH.	Table 22-65. "PEBS Record Format for the Silvermont Microarchitecture".	
PEBS EventingIP	Yes	Yes	
Off-core Response Event	MSR 1A6H and 1A7H, each core has its own register.	MSR 1A6H and 1A7H, shared by a pair of cores.	Nehalem supports 1A6H only.

22.5.3.1 Processor Event Based Sampling (PEBS)

Processor event based sampling (PEBS) on the Goldmont microarchitecture is enhanced over prior generations with respect to sampling support of precise events and non-precise events. In the Goldmont microarchitecture, PEBS is supported using IA32_PMC0 for all events (see Section 20.4.9).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor at the time the sample was generated.

Precise events work the same way on Goldmont microarchitecture as on the Silvermont microarchitecture. The record will be generated after an instruction that causes the event when the counter is already overflowed and will

capture the architectural state at this point (see Section 22.6.2.4 and Section 20.4.9). The eventingIP in the record will indicate the instruction that caused the event. The list of precise events supported in the Goldmont microarchitecture is shown in Table 22-71. "Precise Events Supported by the Goldmont Microarchitecture".

In the Goldmont microarchitecture, the PEBS facility also supports the use of non-precise events to record processor state information into PEBS records with the same format as with precise events.

However, a non-precise event may not be attributable to a particular retired instruction or the time of instruction execution. When the counter overflows, a PEBS record will be generated at the next opportunity. Consider the event ICACHE.HIT. When the counter overflows, the processor is fetching future instructions. The PEBS record will be generated at the next opportunity and capture the state at the processor's current retirement point. It is likely that the instruction fetch that caused the event to increment was beyond that current retirement point. Other examples of non-precise events are CPU_CLK_UNHALTED.CORE_P and HARDWARE_INTERRUPTS.RECEIVED. CPU_CLK_UNHALTED.CORE_P will increment each cycle that the processor is awake. When the counter overflows, there may be many instructions in various stages of execution. Additionally, zero, one or multiple instructions may be retired the cycle that the counter overflows. HARDWARE_INTERRUPTS.RECEIVED increments independent of any instructions being executed. For all non-precise events, the PEBS record will be generated at the next opportunity, after the counter has overflowed. The PEBS facility thus allows for identification of the instructions which were executing when the event overflowed.

After generating a record for a non-precise event, the PEBS facility reloads the counter and resumes execution, just as is done for precise events. Unlike interrupt-based sampling, which requires an interrupt service routine to collect the sample and reload the counter, the PEBS facility can collect samples even when interrupts are masked and without using NMI. Since a PEBS record is generated immediately when a counter for a non-precise event is enabled, it may also be generated after an overflow is set by an MSR write to IA32_PERF_GLOBAL_STATUS_SET.

Table 22-71. Precise Events Supported by the Goldmont Microarchitecture

Event Name	Event Select	Sub-event	UMask
LD_BLOCKS	03H	DATA_UNKNOWN	01H
		STORE_FORWARD	02H
		4K_ALIAS	04H
		UTLB_MISS	08H
		ALL_BLOCK	10H
MISALIGN_MEM_REF	13H	LOAD_PAGE_SPLIT	02H
		STORE_PAGE_SPLIT	04H
INST_RETIRED	C0H	ANY	00H
UOPS_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

Table 22-71. Precise Events Supported by the Goldmont Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H
MEM_UOPS_RETIRED	D0H	ALL_LOADS	81H
		ALL_STORES	82H
		ALL	83H
		DLTB_MISS_LOADS	11H
		DLTB_MISS_STORES	12H
		DLTB_MISS	13H
MEM_LOAD_UOPS_RETIRED	D1H	L1_HIT	01H
		L2_HIT	02H
		L1_MISS	08H
		L2_MISS	10H
		HITM	20H
		WCB_HIT	40H
		DRAM_HIT	80H

The PEBS record format supported by processors based on the Goldmont microarchitecture is shown in Table Table 22-72. "PEBS Record Format for the Goldmont Microarchitecture", and each field in the PEBS record is 64 bits long.

Table 22-72. PEBS Record Format for the Goldmont Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counters
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Reserved
40H	R/EBP	A8H	Reserved
48H	R/ESP	B0H	EventingRIP
50H	R8	B8H	Reserved
58H	R9	C0H	TSC
60H	R10		

On Goldmont microarchitecture, all 64 bits of architectural registers are written into the PEBS record regardless of processor mode.

With PEBS record format encoding 0011b, offset 90H reports the “Applicable Counter” field, which indicates which counters actually requested generating a PEBS record. This allows software to correlate the PEBS record entry properly with the instruction that caused the event even when multiple counters are configured to record PEBS records and multiple bits are set in the field. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

22.5.3.1.1 PEBS Data Linear Address Profiling

Goldmont supports the Data Linear Address field introduced in Haswell. It does not support the Data Source Encoding or Latency Value fields that are also part of Data Address Profiling; those fields are present in the record but are reserved.

For Goldmont microarchitecture, the Data Linear Address field will record the linear address of memory accesses in the previous instruction (e.g., the one that triggered a precise event that caused the PEBS record to be generated). Goldmont microarchitecture may record a Data Linear Address for the instruction that caused the event even for events not related to memory accesses. This may differ from other microarchitectures.

22.5.3.1.2 Reduced Skid PEBS

Processors based on Goldmont Plus microarchitecture support the Reduced Skid PEBS feature described in Section 22.9.4 on the IA32_PMC0 counter. Although Extended PEBS adds support for generating PEBS records for precise events on additional general-purpose and fixed-function performance counters, those counters do not support the Reduced Skid PEBS feature.

22.5.3.1.3 Enhancements to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62]

In addition to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62] being set when PEBS_Index reaches the PEBS_Interrupt_Threshold, the bit is also set when PEBS_Index is out of bounds. That is, the bit will be set when PEBS_Index < PEBS_Buffer_Base or PEBS_Index > PEBS_Absolute_Maximum. Note that when an out of bound condition is encountered, the overflow bits in IA32_PERF_GLOBAL_STATUS will be cleared according to Applicable Counters, however the IA32_PMCx values will not be reloaded with the Reset values stored in the DS_AREA.

22.5.3.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table Table 22-66. "OffCore Response Event Encoding" lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

The Goldmont microarchitecture provides unique pairs of MSR_OFFCORE_RSPx registers per core.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as follows:

- Bits 15:0 specifies the request type of a transaction request to the uncore. This is described in Table Table 22-73. "MSR_OFFCORE_RSPx Request_Type Field Definition".
- Bits 30:16 specifies common supplier information or an L2 Hit, and is described in Table Table 22-68. "MSR_OFFCORE_RSP_x Response Supplier Info Field Definition".
- If L2 misses, then Bits 37:31 can be used to specify snoop response information and is described in Table Table 22-74. "MSR_OFFCORE_RSPx For L2 Miss and Outstanding Requests".
- For outstanding requests, bit 38 can enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously; see Section 22.5.2.3 for details.

Table 22-73. MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts cacheline read requests due to demand reads (excludes prefetches).

Table 22-73. MSR_OFFCORE_RSPx Request_Type Field Definition (Contd.)

Bit Name	Offset	Description
DEMAND_RFO	1	Counts cacheline read for ownership (RFO) requests due to demand writes (excludes prefetches).
DEMAND_CODE_RD	2	Counts demand instruction cacheline and I-side prefetch requests that miss the instruction cache.
COREWB	3	Counts writeback transactions caused by L1 or L2 cache evictions.
PF_L2_DATA_RD	4	Counts data cacheline reads generated by hardware L2 cache prefetcher.
PF_L2_RFO	5	Counts reads for ownership (RFO) requests generated by L2 prefetcher.
Reserved	6	Reserved.
PARTIAL_READS	7	Counts demand data partial reads, including data in uncacheable (UC) or uncacheable (WC) write combining memory types.
PARTIAL_WRITES	8	Counts partial writes, including uncacheable (UC), write through (WT) and write protected (WP) memory type writes.
UC_CODE_READS	9	Counts code reads in uncacheable (UC) memory region.
BUS_LOCKS	10	Counts bus lock and split lock requests.
FULL_STREAMING_STORES	11	Counts full cacheline writes due to streaming stores.
SW_PREFETCH	12	Counts cacheline requests due to software prefetch instructions.
PF_L1_DATA_RD	13	Counts data cacheline reads generated by hardware L1 data cache prefetcher.
PARTIAL_STREAMING_STORES	14	Counts partial cacheline writes due to streaming stores.
ANY_REQUEST	15	Counts requests to the uncore subsystem.

To properly program this extra register, software must set at least one request type bit (Table Table 22-67. "MSR_OFFCORE_RSPx Request_Type Field Definition") and a valid response type pattern (either Table Table 22-68. "MSR_OFFCORE_RSP_x Response Supplier Info Field Definition" or Table Table 22-74. "MSR_OFFCORE_RSPx For L2 Miss and Outstanding Requests"). Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 22-74. MSR_OFFCORE_RSPx For L2 Miss and Outstanding Requests

Subtype	Bit Name	Offset	Description
L2_MISS (Snoop Info)	Reserved	32:31	Reserved
	L2_MISS.SNOOP_MISS_0 R_NO_SNOOP_NEEDED	33	A true miss to this module, for which a snoop request missed the other module or no snoop was performed/needed.
	L2_MISS.HIT_OTHER_CO RE_NO_FWD	34	A snoop hit in the other processor module, but no data forwarding is required.
	Reserved	35	Reserved
	L2_MISS.HITM_OTHER_C ORE	36	Counts the number of snoops hit in the other module or other core's L1 where modified copies were found.
	L2_MISS.NON_DRAM	37	Target was a non-DRAM system address. This includes MMIO transactions.
Outstanding requests ¹	OUTSTANDING	38	Counts weighted cycles of outstanding offcore requests of the request type specified in bits 15:0, from the time the XQ receives the request and any response is received. Bits 37:16 must be set to 0. This bit is only available in MSR_OFFCORE_RESP0.

NOTES:

1. See Section 22.5.2.3, “Average Offcore Request Latency Measurement,” for details on how to use this bit to extract average latency.

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

Any_Response Bit | L2 Hit | ‘OR’ of Snoop Info Bits | Outstanding Bit

22.5.3.3 Average Offcore Request Latency Measurement

In Goldmont microarchitecture, measurement of average latency of offcore transaction requests is the same as described in Section 22.5.2.3.

22.5.4 Performance Monitoring for Goldmont Plus Microarchitecture

Intel Atom processors based on the Goldmont Plus microarchitecture report architectural performance monitoring versionID = 4 and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 22.2.4.

Goldmont Plus performance monitoring capabilities are similar to Goldmont capabilities. The differences are in specific events and in which counters support PEBS. Goldmont Plus introduces the ability for fixed performance monitoring counters to generate PEBS records.

Goldmont Plus will set the AnyThread deprecation CPUID bit (CPUID.0AH:EDX[15]) to indicate that the Any-Thread bits in IA32_PERFVTSELx and IA32_FIXED_CTR_CTRL have no effect.

The core PMU's capability is similar to that of the Goldmont microarchitecture described in Section 22.6.3, with some differences and enhancements summarized in Table 22-75.

Table 22-75. Core PMU Comparison Between the Goldmont Plus and Goldmont Microarchitectures

Box	Goldmont Plus Microarchitecture	Goldmont Microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change.
Architectural Performance Monitoring version ID	4	4	No change.
Processor Event Based Sampling (PEBS) Events	All General-Purpose and Fixed counters. Each General-Purpose counter supports all events (precise and non-precise).	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 22-71. “Precise Events Supported by the Goldmont Microarchitecture”.	Goldmont Plus supports PEBS on all counters.
PEBS record format encoding	0011b	0011b	No change.

22.5.4.1 Extended PEBS

The PEBS facility in Goldmont Plus microarchitecture provides a number of enhancements relative to PEBS in processors from previous generations. Enhancement of PEBS facility with the Extended PEBS feature are described in detail in section 18.9.

22.5.5 Performance Monitoring for Tremont Microarchitecture

Intel Atom processors based on the Tremont microarchitecture report architectural performance monitoring versionID = 5 and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 5 capabilities are described in Section 22.2.5.

Tremont performance monitoring capabilities are similar to Goldmont Plus capabilities, with the following extensions:

- Support for Adaptive PEBS.
- Support for PEBS output to Intel® Processor Trace.
- Precise Distribution support on Fixed Counter0.
- Compatibility enhancements to off-core response MSRs, MSR_OFFCORE_RSPx.

The differences and enhancements between Tremont microarchitecture and Goldmont Plus microarchitecture are summarized in Table 22-76.

Table 22-76. Core PMU Comparison Between the Tremont and Goldmont Plus Microarchitectures

Box	Tremont Microarchitecture	Goldmont Plus Microarchitecture	Comment
# of fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change. See Section 22.2.2.
Architectural Performance Monitoring version ID	5	4	
PEBS record format encoding	0100b	0011b	See Section 22.6.2.4.2.
Reduce skid PEBS	IA32_PMC0 and IA32_FIXED_CTR0	IA32_PMC0 only	
Extended PEBS	Yes	Yes	See Section 22.5.4.1.
Adaptive PEBS	Yes	No	See Section 22.9.2.
PEBS output	DS Save Area or Intel® Processor Trace	DS Save Area only	See Section 22.5.5.2.1.
PEBS record layout	See Section 22.9.2.3 for output to DS, Section 22.5.5.2.2 for output to Intel PT.	Table 22-72; enhanced fields at offsets 90H- 98H; and TSC record field at COH.	
Off-core Response Event	MSR 1A6H and 1A7H, each core has its own register, extended request and response types.	MSR 1A6H and 1A7H, each core has its own register.	

22.5.5.1 Adaptive PEBS

The PEBS record format and configuration interface has changed versus Goldmont Plus, as the Tremont microarchitecture includes support for the configurable Adaptive PEBS records; see Section 22.9.2.

22.5.5.2 PEBS output to Intel® Processor Trace

Intel Atom processors based on the Tremont microarchitecture introduce the following Precise Event-Based Sampling (PEBS) extensions:

- A mechanism to direct PEBS output into the Intel® Processor Trace (Intel® PT) output stream. In this scenario, the PEBS record is written in packetized form, in order to co-exist with other Intel PT trace data.
- New Performance Monitoring counter reload MSRs, which are used by PEBS in place of the counter reload values stored in the DS Management area when PEBS output is directed into the Intel PT output stream.

Processors that indicate support for Intel PT by setting CPUID.07H.00H:EBX[25]=1, and set the new IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] bit, support these extensions.

22.5.5.2.1 PEBS Configuration

PEBS output to Intel Processor Trace includes support for two new fields in IA32_PEBS_ENABLE.

Table 22-77. New Fields in IA32_PEBS_ENABLE

Field	Description
PMI_AFTER_EACH_RECORD[60]	Pend a PerfMon Interrupt (PMI) after each PEBS event.
PEBS_OUTPUT[62:61]	Specifies PEBS output destination. Encodings: 00B: DS Save Area. Matches legacy PEBS behavior, output location defined by IA32_DS_AREA. 01B: Intel PT trace output. 10B: Reserved. 11B: Reserved.

When PEBS_OUTPUT is set to 01B, the DS Management Area is not used and need not be configured. Instead, the output mechanism is configured through IA32_RTIT_CTL and other Intel PT MSRs, while counter reload values are configured in the MSR_RELOAD_PMCx MSRs. Details on configuring Intel PT can be found in Section 36.2.7.

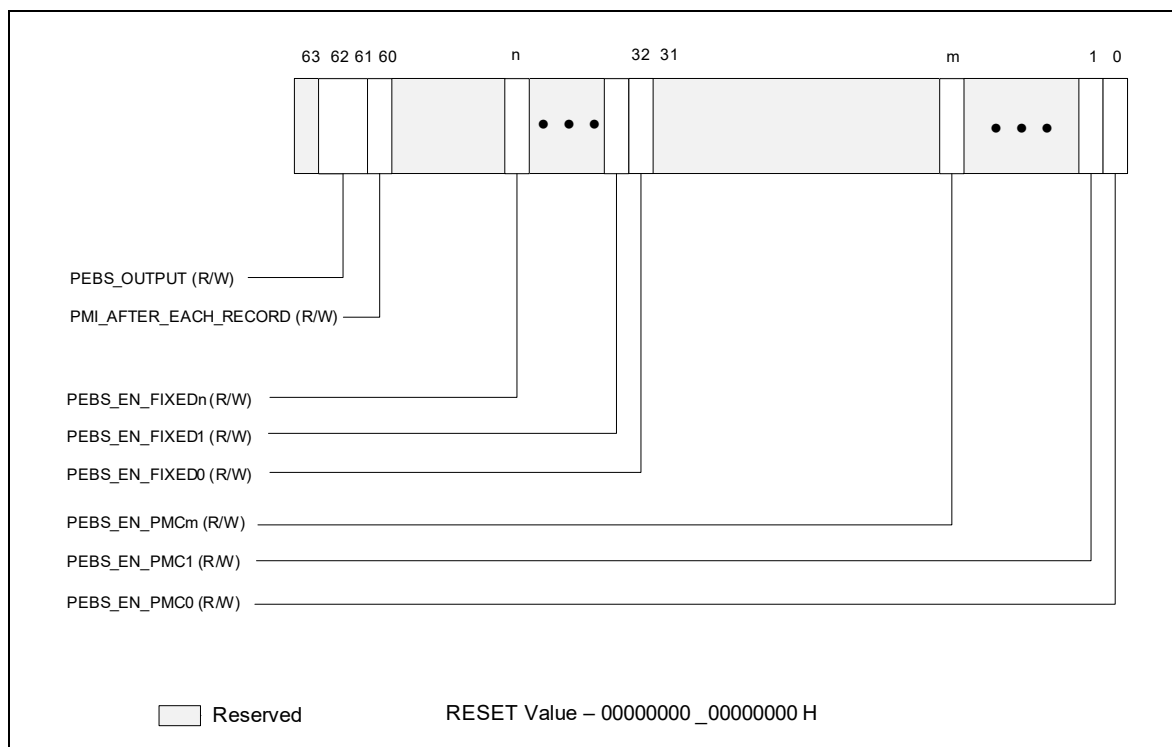


Figure 22-44. IA32_PEBS_ENABLE MSR with PEBS Output to Intel® Processor Trace

22.5.5.2.2 PEBS Record Format in Intel® Processor Trace

The format of the PEBS record changes when output to Intel PT, as the PEBS state is packetized. Each PEBS grouping is emitted as a Block Begin (BBP) and following Block Item (BIP) packets. A PEBS grouping ends when either a new PEBS grouping begins (indicated by a BBP packet) or a Block End (BEP) packet is encountered. See Section 36.4.1.1 for details of these Intel PT packets.

Because the packet headers describe the state held in the packet payload, PEBS state ordering is not fixed. PEBS state groupings may be emitted in any order, and the PEBS state elements within those groupings may be emitted in any order. Further, there is no packet that provides indication of “Record Format” or “Record Size”.

If Intel PT tracing is not enabled (IA32_RTIT_STATUS.TriggerEn=0), any PEBS records triggered will be dropped. PEBS packets do not depend on ContextEn or FilterEn in IA32_RTIT_STATUS, any filtering of PEBS must be enabled from within the PerfMon configuration. Counter reload will occur in all scenarios where PEBS is triggered, regardless of TriggerEn.

The PEBS threshold mechanism for generating PerfMon Interrupts (PMIs) is not available in this mode. However, there exist other means to generate PMIs based on PEBS output. When the Intel PT ToPA output mechanism is chosen, a PMI can optionally be pended when a ToPA region is filled; see Section 36.2.7.2 for details. Further, software can opt to generate a PMI on each PEBS record by setting the new IA32_PEBS_ENABLE.PMI_AFTER_EACH_RECORD[60] bit.

The IA32_PERF_GLOBAL_STATUS.OvfDSBuffer bit will not be set in this mode.

22.5.5.2.3 PEBS Counter Reload

When PEBS output is directed into Intel PT (IA32_PEBS_ENABLE.PEBS_OUTPUT = 01B), new MSR_RELOAD_PMCx MSRs are used by the PEBS routine to reload PerfMon counters. The value from the associated reload MSR will be loaded to the appropriate counter on each PEBS event.

22.5.5.3 Precise Distribution Support on Fixed Counter 0

The Tremont microarchitecture supports the PDIR (Precise Distribution of Retired Instructions) facility, as described in Section 22.3.4.4.4, on Fixed Counter 0. Fixed Counter 0 counts the INST_RETIRED.ALL event. PEBS skid for Fixed Counter 0 will be precisely one instruction.

This is in addition to the reduced skid PEBS behavior on IA32_PMC0; see Section 22.5.3.1.2.

22.5.5.4 Compatibility Enhancements to Offcore Response MSRs

The Off-core Response facility is similar to that described in Section 22.5.3.2.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as shown below. RequestType bits are defined in Table 22-78, ResponseType bits in Table 22-79, and SnoopInfo bits in Table 22-80.

Table 22-78. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data reads.
DEMAND_RFO	1	Counts all demand reads for ownership (RFO) requests and software based prefetches for exclusive ownership (prefetchw).
DEMAND_CODE_RD	2	Counts demand instruction fetches and L1 instruction cache prefetches.
COREWB_M	3	Counts modified write backs from L1 and L2.
HWPf_L2_DATA_RD	4	Counts prefetch (that bring data to L2) data reads.
HWPf_L2_RFO	5	Counts all prefetch (that bring data to L2) RFOs.
HWPf_L2_CODE_RD	6	Counts all prefetch (that bring data to L2 only) code reads.
Reserved	9:7	Reserved.

Table 22-78. MSR_OFFCORE_RSPx Request Type Definition (Contd.)

Bit Name	Offset	Description
HWPf_L1D_AND_SWPF	10	Counts L1 data cache hardware prefetch requests, read for ownership prefetch requests and software prefetch requests (except prefetchw).
STREAMING_WR	11	Counts all streaming stores.
COREWB_NONM	12	Counts non-modified write backs from L2.
Reserved	14:13	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O accesses that have any response type.
UC_RD	44	Counts uncached memory reads (PRd, UCRdF).
UC_WR	45	Counts uncached memory writes (WiL).
PARTIAL_STREAMING_WR	46	Counts partial (less than 64 byte) streaming stores (WCiL).
FULL_STREAMING_WR	47	Counts full, 64 byte streaming stores (WCiLF).
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

Table 22-79. MSR_OFFCORE_RSPx Response Type Definition

Bit Name	Offset	Description
ANY_RESPONSE	16	Catch all value for any response types.
L3_HIT_M	18	LLC/L3 Hit - M-state.
L3_HIT_E	19	LLC/L3 Hit - E-state.
L3_HIT_S	20	LLC/L3 Hit - S-state.
L3_HIT_F	21	LLC/L3 Hit - I-state.
LOCAL_DRAM	26	LLC/L3 Miss, DRAM Hit.
OUTSTANDING	63	Average latency of outstanding requests with the other counter counting number of occurrences; can also can be used to count occupancy.

Table 22-80. MSR_OFFCORE_RSPx Snoop Info Definition

Bit Name	Offset	Description
SNOOP_NONE	31	None of the cores were snooped. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_NOT_NEEDED	32	No snoop needed to satisfy the request. ▪ LLC hit and CV bit(s) (core valid) was not set. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_MISS	33	A snoop was sent but missed. ▪ LLC hit and CV bit(s) was set but snoop missed (silent data drop in core), data returned from LLC. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_HIT_NO_FWD	34	A snoop was sent but no data forward. ▪ LLC hit and CV bit(s) was set but no data forward from the core, data returned from LLC. ▪ LLC miss and Dram data returned directly to the core.

Table 22-80. MSR_OFFCORE_RSPx Snoop Info Definition

Bit Name	Offset	Description
SNOOP_HIT_WITH_FWD	35	A snoop was sent and non-modified data was forward. ▪ LLC hit and CV bit(s) was set, non-modified data was forward from core.
SNOOP_HITM	36	A snoop was sent and modified data was forward. ▪ LLC hit E or M and the CV bit(s) was set, modified data was forward from core.
NON_DRAM_BIT	37	Target was non-DRAM system address, MMIO access. ▪ LLC miss and Non-Dram data returned.

The Off-core Response capability behaves as follows:

- To specify a complete offcore response filter, software must properly program at least one RequestType and one ResponseType. A valid request type must have at least one bit set in the non-reserved bits of 15:0 or 49:44. A valid response type must be a non-zero value of one the following expressions:
 - Read requests:
Any_Response Bit | ('OR' of Supplier Info Bits) 'AND' ('OR' of Snoop Info Bits) | Outstanding Bit
 - Write requests:
Any_Response Bit | ('OR' of Supplier Info Bits) | Outstanding Bit
- When the ANY_RESPONSE bit in the ResponseType is set, all other response type bits will be ignored.
- True Demand Cacheable Loads include neither L1 Prefetches nor Software Prefetches.
- Bits 15:0 and Bits 49:44 specifies the request type of a transaction request to the uncore. This is described in Table 22-78.
- Bits 30:16 specifies common supplier information.
- "Outstanding Requests" (bit 63) is only available on MSR_OFFCORE_RSP0; a #GP fault will occur if software attempts to write a 1 to this bit in MSR_OFFCORE_RSP1. It is mutually exclusive with any ResponseType. Software must guarantee that all other ResponseType bits are set to 0 when the "Outstanding Requests" bit is set.
- "Outstanding Requests" bit 63 can enable measurement of the average latency of a specific type of off-core transaction; two programmable counters must be used simultaneously and the RequestType programming for MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 must be the same when using this Average Latency feature. See Section 22.5.2.3 for further details.

22.6 PERFORMANCE MONITORING (LEGACY INTEL PROCESSORS)

22.6.1 Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)

In Intel Core Solo and Intel Core Duo processors, non-architectural performance monitoring events are programmed using the same facilities (see Figure 22-1) used for architectural performance events.

Non-architectural performance events use event select values that are model-specific. Event mask (Umask) values are also specific to event logic units. Some microarchitectural conditions detectable by a Umask value may have specificity related to processor topology (see Section 11.6, "Detecting Hardware Multi-Threading Support and Topology," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A). As a result, the unit mask field (for example, IA32_PERFEVTSELx[bits 15:8]) may contain sub-fields that specify topology information of processor cores.

The sub-field layout within the Umask field may support two-bit encoding that qualifies the relationship between a microarchitectural condition and the originating core. This data is shown in Table 22-81. The two-bit encoding for core-specificity is only supported for a subset of Umask values (see: <https://perfmon-events.intel.com/>) and for Intel Core Duo processors. Such events are referred to as core-specific events.

Table 22-81. Core Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 15:14 Encoding	Description
11B	All cores
10B	Reserved
01B	This core
00B	Reserved

Some microarchitectural conditions allow detection specificity only at the boundary of physical processors. Some bus events belong to this category, providing specificity between the originating physical processor (a bus agent) versus other agents on the bus. Sub-field encoding for agent specificity is shown in Table Table 22-82. "Agent Specificity Encoding within a Non-Architectural Umask".

Table 22-82. Agent Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13 Encoding	Description
0	This agent
1	Include all agents

Some microarchitectural conditions are detectable only from the originating core. In such cases, unit mask does not support core-specificity or agent-specificity encodings. These are referred to as core-only conditions.

Some microarchitectural conditions allow detection specificity that includes or excludes the action of hardware prefetches. A two-bit encoding may be supported to qualify hardware prefetch actions. Typically, this applies only to some L2 or bus events. The sub-field encoding for hardware prefetch qualification is shown in Table Table 22-83. "HW Prefetch Qualification Encoding within a Non-Architectural Umask".

Table 22-83. HW Prefetch Qualification Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13:12 Encoding	Description
11B	All inclusive
10B	Reserved
01B	Hardware prefetch only
00B	Exclude hardware prefetch

Some performance events may (a) support none of the three event-specific qualification encodings (b) may support core-specificity and agent specificity simultaneously (c) or may support core-specificity and hardware prefetch qualification simultaneously. Agent-specificity and hardware prefetch qualification are mutually exclusive.

In addition, some L2 events permit qualifications that distinguish cache coherent states. The sub-field definition for cache coherency state qualification is shown in Table Table 22-84. "MESI Qualification Definitions within a Non-Architectural Umask". If no bits in the MESI qualification sub-field are set for an event that requires setting MESI qualification bits, the event count will not increment.

Table 22-84. MESI Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	Description
Bit 11	Counts modified state
Bit 10	Counts exclusive state

Table 22-84. MESI Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	Description
Bit 9	Counts shared state
Bit 8	Counts Invalid state

22.6.2 Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)

In addition to architectural performance monitoring, processors based on the Intel Core microarchitecture support non-architectural performance monitoring events.

Architectural performance events can be collected using general-purpose performance counters. Non-architectural performance events can be collected using general-purpose performance counters (coupled with two IA32_PERFEVTSELx MSRs for detailed event configurations), or fixed-function performance counters (see Section 22.6.2.1). IA32_PERFEVTSELx MSRs are architectural; their layout is shown in Figure 22-1. Starting with Intel Core 2 processor T 7700, fixed-function performance counters and associated counter control and status MSR becomes part of architectural performance monitoring version 2 facilities (see also Section 22.2.2).

Non-architectural performance events in processors based on Intel Core microarchitecture use event select values that are model-specific. Valid event mask (Umask) bits can be found at: <https://perfmon-events.intel.com/>. The UMASK field may contain sub-fields identical to those listed in Table Table 22-81. "Core Specificity Encoding within a Non-Architectural Umask", Table Table 22-82. "Agent Specificity Encoding within a Non-Architectural Umask", Table Table 22-83. "HW Prefetch Qualification Encoding within a Non-Architectural Umask", and Table Table 22-84. "MESI Qualification Definitions within a Non-Architectural Umask". One or more of these sub-fields may apply to specific events on an event-by-event basis.

In addition, the UMASK field may also contain a sub-field that allows detection specificity related to snoop responses. Bits of the snoop response qualification sub-field are defined in Table Table 22-85. "Bus Snoop Qualification Definitions within a Non-Architectural Umask".

Table 22-85. Bus Snoop Qualification Definitions within a Non-Architectural Umask

IA32_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 22-86.

Table 22-86. 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.

22.6.2.1 Fixed-function Performance Counters

Processors based on Intel Core microarchitecture provide three fixed-function performance counters. Bits beyond the width of the fixed counter are reserved and must be written as zeros. Model-specific fixed-function performance counters on processors that support Architectural PerfMon version 1 are 40 bits wide.

Each of the fixed-function counter is dedicated to count a pre-defined performance monitoring events. See Table 22-1 for details of the PMC addresses and what these events count.

Programming the fixed-function performance counters does not involve any of the IA32_PERFEVTSELx MSRs, and does not require specifying any event masks. Instead, the MSR IA32_FIXED_CTR_CTRL provides multiple sets of 4-bit fields; each 4-bit field controls the operation of a fixed-function performance counter (PMC). See Figures 22-45. Two sub-fields are defined for each control. See Figure 22-45; bit fields are:

- **Enable field (low 2 bits in each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring 0.

When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring greater than 0.

Writing 0 to both bits stops the performance counter. Writing 11B causes the counter to increment irrespective of privilege levels.

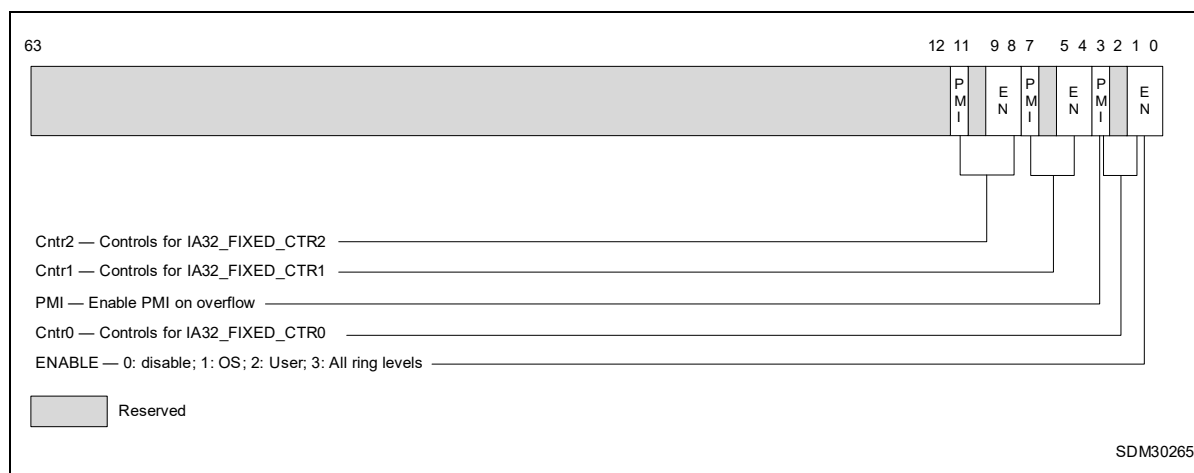


Figure 22-45. Layout of IA32_FIXED_CTR_CTRL MSR

- **PMI field (fourth bit in each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

22.6.2.2 Global Counter Control Facilities

Processors based on Intel Core microarchitecture provides simplified performance counter control that simplifies the most frequent operations in programming performance events, i.e., enabling/disabling event counting and checking the status of counter overflows. This is done by the following three MSRs:

- MSR_PERF_GLOBAL_CTRL enables/disables event counting for all or any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.

- MSR_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single RDMSR.
- MSR_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.

MSR_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting in each performance counter (see Figure 22-46). Each enable bit in MSR_PERF_GLOBAL_CTRL is ANDed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the ANDed results is true; counting is disabled when the result is false.

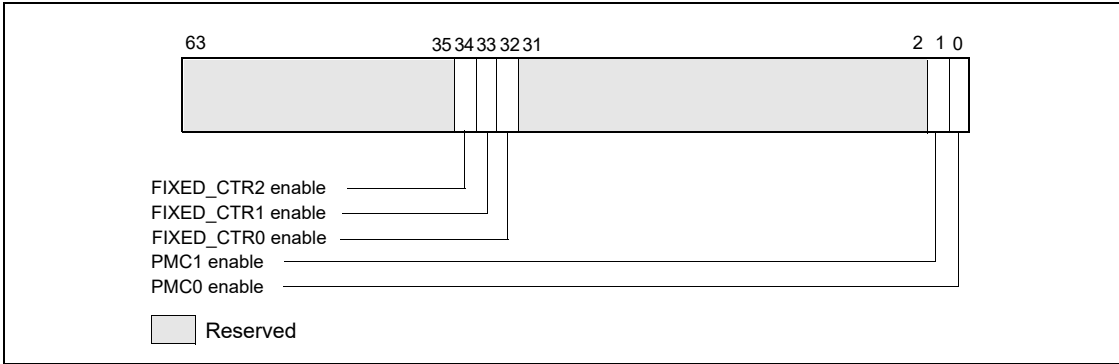


Figure 22-46. Layout of MSR_PERF_GLOBAL_CTRL MSR

MSR_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. MSR_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer. MSR_PERF_GLOBAL_STATUS[bit 63] provides a CondChgd bit to indicate changes to the state of performance monitoring hardware (see Figure 22-47). A value of 1 in bits 34:32, 1, 0 indicates an overflow condition has occurred in the associated counter.

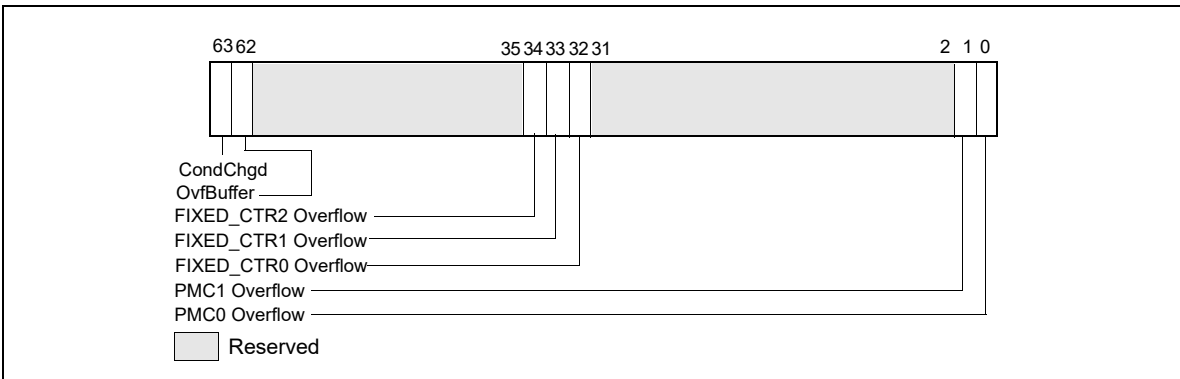


Figure 22-47. Layout of MSR_PERF_GLOBAL_STATUS MSR

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 20.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

MSR_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 22-48). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or interrupt-based event sampling.
- Reloading counter values to continue collecting next sample.
- Disabling event counting or interrupt-based event sampling.

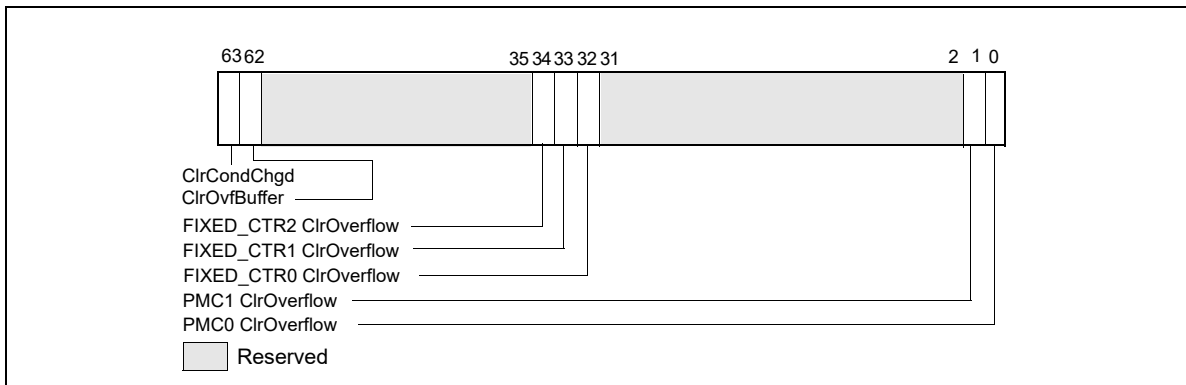


Figure 22-48. Layout of MSR_PERF_GLOBAL_OVF_CTRL MSR

22.6.2.3 At-Retirement Events

Many non-architectural performance events are impacted by the speculative nature of out-of-order execution. A subset of non-architectural performance events on processors based on Intel Core microarchitecture are enhanced with a tagging mechanism (similar to that found in Intel NetBurst® microarchitecture) that exclude contributions that arise from speculative execution. The at-retirement events available in processors based on Intel Core microarchitecture does not require special MSR programming control (see Section 22.6.3.6, "At-Retirement Counting"), but is limited to IA32_PMC0. See Table 22-87. "At-Retirement Performance Events for Intel Core Microarchitecture" for a list of events available to processors based on Intel Core microarchitecture.

Table 22-87. At-Retirement Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
ITLB_MISS_RETIRED	00H	C9H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

22.6.2.4 Processor Event Based Sampling (PEBS)

Processors based on Intel Core microarchitecture also support processor event based sampling (PEBS). This feature was introduced by processors based on Intel NetBurst microarchitecture.

PEBS uses a debug store mechanism and a performance monitoring interrupt to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 22.6.2.4.2 and Section 20.4.9).

In cases where the same instruction causes BTS and PEBS to be activated, PEBS is processed before BTS are processed. The PMI request is held until the processor completes processing of PEBS and BTS.

For processors based on Intel Core microarchitecture, precise events that can be used with PEBS are listed in Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture". The procedure for detecting availability of PEBS is the same as described in Section 22.6.3.8.1.

Table 22-88. PEBS Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
INSTR_RETIRED.ANY_P	00H	C0H
X87_OPS_RETIRED.ANY	FEH	C1H
BR_INST_RETIRED.MISPRED	00H	C5H
SIMD_INST_RETIRED.ANY	1FH	C7H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

22.6.2.4.1 Setting up the PEBS Buffer

For processors based on Intel Core microarchitecture, PEBS is available using IA32_PMC0 only. Use the following procedure to set up the processor and IA32_PMC0 counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area. In processors based on Intel Core microarchitecture, PEBS records consist of 64-bit address entries. See Figure 20-8 to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS on PMC0 flag (bit 0) in IA32_PEBS_ENABLE MSR.
3. Set up the IA32_PMC0 performance counter and IA32_PERFEVTSEL0 for an event listed in Table Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture".

22.6.2.4.2 PEBS Record Format

The PEBS record format may be extended across different processor implementations. The IA32_PERF_CAPABILITIES MSR defines a mechanism for software to handle the evolution of PEBS record format in processors that support architectural performance monitoring with version ID equals 2 or higher. The bit fields of IA32_PERF_CAPABILITIES are defined in Table Table 2-2. "IA-32 Architectural MSRs" of Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4. The relevant bit fields that governs PEBS are:

- **PEBSTrap [bit 6]:** When set, PEBS recording is trap-like. After the PEBS-enabled counter has overflowed, PEBS record is recorded for the next PEBS-able event at the completion of the sampled instruction causing the PEBS event. When clear, PEBS recording is fault-like. The PEBS record is recorded before the sampled instruction causing the PEBS event.
- **PEBSSaveArchRegs [bit 7]:** When set, PEBS will save architectural register and state information according to the encoded value of the PEBSRecordFormat field. When clear, only the return instruction pointer and flags are recorded. On processors based on Intel Core microarchitecture, this bit is always 1.
- **PEBSRecordFormat [bits 11:8]:** Valid encodings are:
 - 0000B: Only general-purpose registers, instruction pointer and RFLAGS registers are saved in each PEBS record (See Section 22.6.3.8).
 - 0001B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS and load latency data. (See Section 22.3.1.1.1).
 - 0010B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS, load latency data, and TSX tuning information. (See Section 22.3.6.2).
 - 0011B: PEBS record includes additional information of load latency data, TSX tuning information, TSC data, and the applicable counter field replaces IA32_PERF_GLOBAL_STATUS at offset 90H. (See Section 22.3.8.1.1).

- 0100B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 22.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 22-66 with Counter Reset fields allocation for 8 general-purpose counters followed by 4 fixed-function counters.
- 0101B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 22.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 22-66 with Counter Reset fields allocation for 32 general-purpose counters followed by 16 fixed-function counters.
- 0110B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG (see Figure 22-73 in Section 22.9.2.3) that is compatible with the previous MSR_PEBS_DATA_CFG (see Figure 22-72 in Section 22.9.2.3). The PEBS Config Buffer is defined as shown in Figure 22-72 with a Counter Reset fields allocation for 32 general-purpose counters followed by 16 fixed-function counters.

22.6.2.4.3 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the Interrupt-based event sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 20.4.9.1, "64 Bit Format of the DS Save Area," for guidelines when writing the DS ISR.

The service routine can query MSR_PERF_GLOBAL_STATUS to determine which counter(s) caused of overflow condition. The service routine should clear overflow indicator by writing to MSR_PERF_GLOBAL_OVF_CTL.

A comparison of the sequence of requirements to program PEBS for processors based on Intel Core and Intel NetBurst microarchitectures is listed in Table 22-89. "Requirements to Program PEBS".

Table 22-89. Requirements to Program PEBS

	For Processors based on Intel Core microarchitecture	For Processors based on Intel NetBurst microarchitecture
Verify PEBS support of processor/OS.	▪ IA32_MISC_ENABLE.EMON_AVAILABE (bit 7) is set. ▪ IA32_MISC_ENABLE.PEBS_UNAVAILABE (bit 12) is clear.	
Ensure counters are in disabled.	On initial set up or changing event configurations, write MSR_PERF_GLOBAL_CTRL MSR (38FH) with 0. On subsequent entries: ▪ Clear all counters if "Counter Freeze on PMI" is not enabled. ▪ If IA32_DebugCTL.Freeze is enabled, counters are automatically disabled. Counters MUST be stopped before writing. ¹	Optional
Disable PEBS.	Clear ENABLE PMCO bit in IA32_PEBS_ENABLE MSR (3F1H).	Optional
Check overflow conditions.	Check MSR_PERF_GLOBAL_STATUS MSR (38EH) handle any overflow conditions.	Check OVF flag of each CCCR for overflow condition
Clear overflow status.	Clear MSR_PERF_GLOBAL_STATUS MSR (38EH) using IA32_PERF_GLOBAL_OVF_CTRL MSR (390H).	Clear OVF flag of each CCCR.
Write "sample-after" values.	Configure the counter(s) with the sample after value.	
Configure specific counter configuration MSR.	▪ Set local enable bit 22 - 1. ▪ Do NOT set local counter PMI/INT bit, bit 20 - 0. ▪ Event programmed must be PEBS capable.	▪ Set appropriate OVF_PMI bits - 1. ▪ Only CCCR for MSR_IQ_COUNTER4 support PEBS.
Allocate buffer for PEBS states.	Allocate a buffer in memory for the precise information.	
Program the IA32_DS_AREA MSR.	Program the IA32_DS_AREA MSR.	
Configure the PEBS buffer management records.	Configure the PEBS buffer management records in the DS buffer management area.	
Configure/Enable PEBS.	Set Enable PMCO bit in IA32_PEBS_ENABLE MSR (3F1H).	Configure MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_PEBS_MATRIX_HORZ as needed.

Table 22-89. Requirements to Program PEBS (Contd.)

	For Processors based on Intel Core microarchitecture	For Processors based on Intel NetBurst microarchitecture
Enable counters.	Set Enable bits in MSR_PERF_GLOBAL_CTRL MSR (38FH).	Set each CCCR enable bit 12 - 1.

NOTES:

1. Counters read while enabled are not guaranteed to be precise with event counts that occur in timing proximity to the RDMSR.

22.6.2.4.4 Re-configuring PEBS Facilities

When software needs to reconfigure PEBS facilities, it should allow a quiescent period between stopping the prior event counting and setting up a new PEBS event. The quiescent period is to allow any latent residual PEBS records to complete its capture at their previously specified buffer address (provided by IA32_DS_AREA).

22.6.3 Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)

The performance monitoring mechanism provided in processors based on Intel NetBurst microarchitecture is different from that provided in the P6 family and Pentium processors. While the general concept of selecting, filtering, counting, and reading performance events through the WRMSR, RDMSR, and RDPMC instructions is unchanged, the setup mechanism and MSR layouts are incompatible with the P6 family and Pentium processor mechanisms. Also, the RDPMC instruction has been extended to support faster reading of counters and to read all performance counters available in processors based on Intel NetBurst microarchitecture.

The event monitoring mechanism consists of the following facilities:

- The IA32_MISC_ENABLE MSR, which indicates the availability in an Intel 64 or IA-32 processor of the performance monitoring and processor event-based sampling (PEBS) facilities.
- Event selection control (ESCR) MSRs for selecting events to be monitored with specific performance counters. The number available differs by family and model (43 to 45).
- 18 performance counter MSRs for counting events.
- 18 counter configuration control (CCCR) MSRs, with one CCCR associated with each performance counter. 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 22-90 lists the performance counters and their associated CCCRs, along with the ESCRs that select events to be counted for each performance counter. Predefined event metrics and events can be found at: <https://perfmon-events.intel.com/>.

**Table 22-90. Performance Counter MSRs and Associated CCCR and ESCR MSRs
(Processors Based on Intel NetBurst Microarchitecture)**

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_BPU_COUNTER0	0	300H	MSR_BPU_CCCR0	360H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER1	1	301H	MSR_BPU_CCCR1	361H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER2	2	302H	MSR_BPU_CCCR2	362H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_BPU_COUNTER3	3	303H	MSR_BPU_CCCR3	363H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_MS_COUNTER0	4	304H	MSR_MS_CCCR0	364H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER1	5	305H	MSR_MS_CCCR1	365H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER2	6	306H	MSR_MS_CCCR2	366H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_MS_COUNTER3	7	307H	MSR_MS_CCCR3	367H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_FLAME_COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAA_T_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTER1	9	309H	MSR_FLAME_CCCR1	369H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAA_T_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H

**Table 22-90. Performance Counter MSRs and Associated CCCR and ESCR MSRs
(Processors Based on Intel NetBurst Microarchitecture) (Contd.)**

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_FLAME_COUNTER2	10	30AH	MSR_FLAME_CCCR2	36AH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_FLAME_COUNTER3	11	30BH	MSR_FLAME_CCCR3	36BH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 ¹ MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 ¹ MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 ¹ MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER3	15	30FH	MSR_IQ_CCCR3	36FH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 ¹ MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 ¹ MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 ¹ MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH

NOTES:

1. MSR_IQ_ESCR0 and MSR_IQ_ESCR1 are available only on early processor builds (family 0FH, models 01H-02H). These MSRs are not available on later versions.

The types of events that can be counted with these performance monitoring facilities are divided into two classes: non-retirement events and at-retirement events.

- Non-retirement events are events that occur any time during instruction execution (such as bus transactions or cache transactions).
- At-retirement events are events that are counted at the retirement stage of instruction execution, which allows finer granularity in counting events and capturing machine state.

The at-retirement counting mechanism includes facilities for tagging μ ops that have encountered a particular performance event during instruction execution. Tagging allows events to be sorted between those that occurred on an execution path that resulted in architectural state being committed at retirement as well as events that occurred on an execution path where the results were eventually cancelled and never committed to architectural state (such as, the execution of a mispredicted branch).

The Pentium 4 and Intel Xeon processor performance monitoring facilities support the three usage models described below. The first two models can be used to count both non-retirement and at-retirement events; the third model is used to count a subset of at-retirement events:

- **Event counting** — A performance counter is configured to count one or more types of events. While the counter is counting, software reads the counter at selected intervals to determine the number of events that have been counted between the intervals.
- **Interrupt-based event sampling** — A performance counter is configured to count one or more types of events and to generate an interrupt when it overflows. To trigger an overflow, the counter is preset to a modulus value that will cause the counter to overflow after a specific number of events have been counted.
When the counter overflows, the processor generates a performance monitoring interrupt (PMI). The interrupt service routine for the PMI then records the return instruction pointer (RIP), resets the modulus, and restarts the counter. Code performance can be analyzed by examining the distribution of RIPs with a tool like the VTune™ Performance Analyzer.
- **Processor event-based sampling (PEBS)** — In PEBS, the processor writes a record of the architectural state of the processor to a memory buffer after the counter overflows. The records of architectural state provide additional information for use in performance tuning. Processor-based event sampling can be used to count only a subset of at-retirement events. PEBS captures more precise processor state information compared to interrupt based event sampling, because the latter need to use the interrupt service routine to re-construct the architectural states of processor.

The following sections describe the MSRs and data structures used for performance monitoring in the Pentium 4 and Intel Xeon processors.

22.6.3.1 ESCR MSRs

The 45 ESCR MSRs (see Table 22-90) allow software to select specific events to be countered. Each ESCR is usually associated with a pair of performance counters (see Table 22-90) and each performance counter has several ESCRs associated with it (allowing the events counted to be selected from a variety of events).

Figure 22-49 shows the layout of an ESCR MSR. The functions of the flags and fields are:

- **USR flag, bit 2** — When set, events are counted when the processor is operating at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.
- **OS flag, bit 3** — When set, events are counted when the processor is operating at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the OS and USR flags are set, events are counted at all privilege levels.)

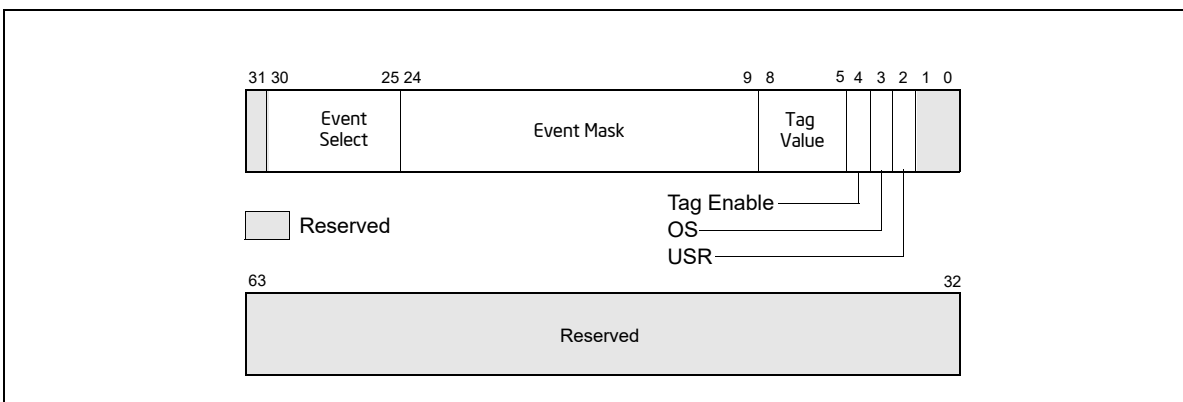


Figure 22-49. Event Selection Control Register (ESCR) for Pentium 4 and Intel® Xeon® Processors without Intel HT Technology Support

- **Tag enable, bit 4** — When set, enables tagging of μ ops to assist in at-retirement event counting; when clear, disables tagging. See Section 22.6.3.6, “At-Retirement Counting.”
- **Tag value field, bits 5 through 8** — Selects a tag value to associate with a μ op to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24** — Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30** — Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

When setting up an ESCR, the event select field is used to select a specific class of events to count, such as retired branches. The event mask field is then used to select one or more of the specific events within the class to be counted. For example, when counting retired branches, four different events can be counted: branch not taken predicted, branch not taken mispredicted, branch taken predicted, and branch taken mispredicted. The OS and USR flags allow counts to be enabled for events that occur when operating system code and/or application code are being executed. If neither the OS nor USR flag is set, no events will be counted.

The ESCRs are initialized to all 0s on reset. The flags and fields of an ESCR are configured by writing to the ESCR using the WRMSR instruction. Table 22-90 gives the addresses of the ESCR MSRs.

Writing to an ESCR MSR does not enable counting with its associated performance counter; it only selects the event or events to be counted. The CCCR for the selected performance counter must also be configured. Configuration of the CCCR includes selecting the ESCR and enabling the counter.

22.6.3.2 Performance Counters

The performance counters in conjunction with the counter configuration control registers (CCCRs) are used for filtering and counting the events selected by the ESCRs. Processors based on Intel NetBurst microarchitecture provide 18 performance counters organized into 9 pairs. A pair of performance counters is associated with a particular subset of events and ESCR's (see Table 22-90). The counter pairs are partitioned into four groups:

- The BPU group, includes two performance counter pairs:
 - MSR_BPU_COUNTER0 and MSR_BPU_COUNTER1.
 - MSR_BPU_COUNTER2 and MSR_BPU_COUNTER3.
- The MS group, includes two performance counter pairs:
 - MSR_MS_COUNTER0 and MSR_MS_COUNTER1.
 - MSR_MS_COUNTER2 and MSR_MS_COUNTER3.
- The FLAME group, includes two performance counter pairs:
 - MSR_FLAME_COUNTER0 and MSR_FLAME_COUNTER1.

- MSR_FLAME_COUNTER2 and MSR_FLAME_COUNTER3.
- The IQ group, includes three performance counter pairs:
 - MSR_IQ_COUNTER0 and MSR_IQ_COUNTER1.
 - MSR_IQ_COUNTER2 and MSR_IQ_COUNTER3.
 - MSR_IQ_COUNTER4 and MSR_IQ_COUNTER5.

The MSR_IQ_COUNTER4 counter in the IQ group provides support for the PEBS.

Alternate counters in each group can be cascaded: the first counter in one pair can start the first counter in the second pair and vice versa. A similar cascading is possible for the second counters in each pair. For example, within the BPU group of counters, MSR_BPU_COUNTER0 can start MSR_BPU_COUNTER2 and vice versa, and MSR_BPU_COUNTER1 can start MSR_BPU_COUNTER3 and vice versa (see Section 22.6.3.5.6, “Cascading Counters”). The cascade flag in the CCCR register for the performance counter enables the cascading of counters.

Each performance counter is 40-bits wide (see Figure 22-50). The RDPMC instruction is intended to allow reading of either the full counter-width (40-bits) or, if ECX[31] is set to 1, the low 32-bits of the counter. Reading the low 32-bits is faster than reading the full counter width and is appropriate in situations where the count is small enough to be contained in 32 bits. In such cases, counter bits 31:0 are written to EAX, while 0 is written to EDX.

The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

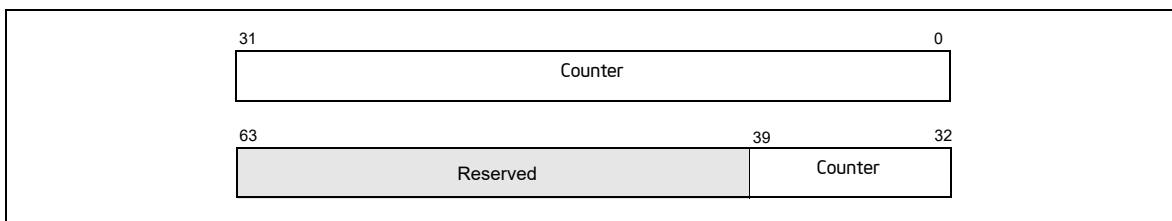


Figure 22-50. Performance Counter (Pentium 4 and Intel® Xeon® Processors)

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

Some uses of the performance counters require the counters to be preset before counting begins (that is, before the counter is enabled). This can be accomplished by writing to the counter using the WRMSR instruction. To set a counter to a specified number of counts before overflow, enter a 2s complement negative integer in the counter. The counter will then count from the preset value up to -1 and overflow. Writing to a performance counter in a Pentium 4 or Intel Xeon processor with the WRMSR instruction causes all 40 bits of the counter to be written.

22.6.3.3 CCCR MSRs

Each of the 18 performance counters has one CCCR MSR associated with it (see Table 22-90). The CCCRs control the filtering and counting of events as well as interrupt generation. Figure 22-51 shows the layout of an CCCR MSR. The functions of the flags and fields are as follows:

- **Enable flag, bit 12** — When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset.
- **ESCR select field, bits 13 through 15** — Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.

- **Compare flag, bit 18** — When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.
- **Complement flag, bit 19** — Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 22.6.3.5.2, “Filtering Events”). The complement flag is not active unless the compare flag is set.
- **Threshold field, bits 20 through 23** — Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 22.6.3.5.2, “Filtering Events”).
- **Edge flag, bit 24** — When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.

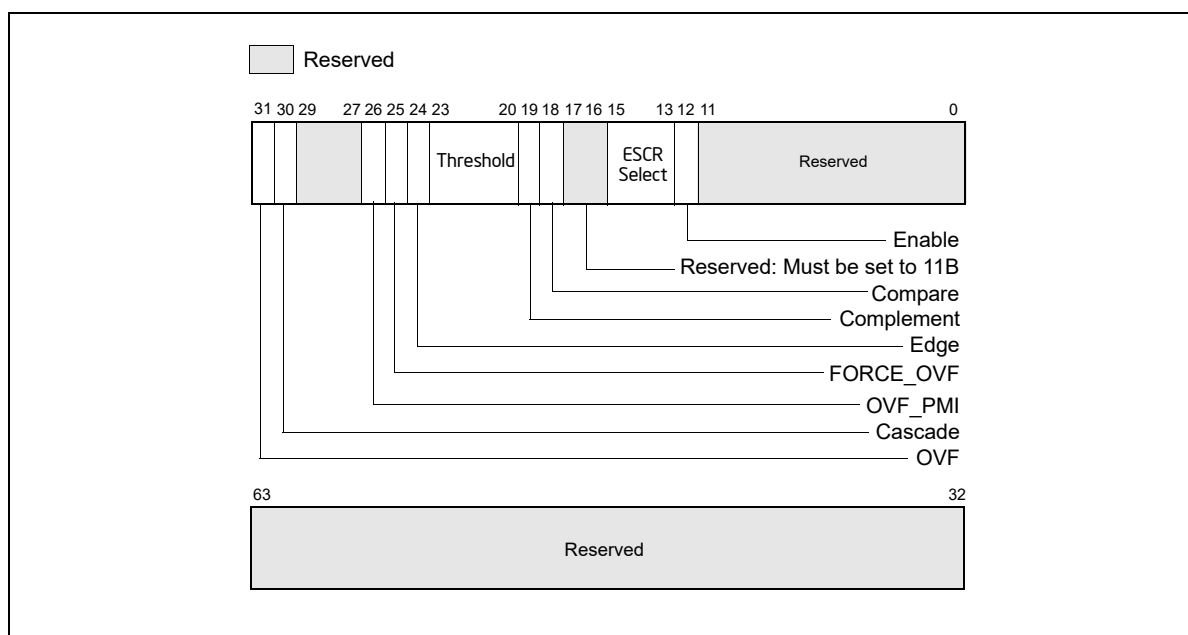


Figure 22-51. Counter Configuration Control Register (CCCR)

- **FORCE_OVF flag, bit 25** — When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI flag, bit 26** — When set, causes a performance monitor interrupt (PMI) to be generated when the counter overflows occurs; when clear, disables PMI generation. Note that the PMI is generated on the next event count after the counter has overflowed.
- **Cascade flag, bit 30** — When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 22.6.3.2, “Performance Counters,” for further details); when clear, disables cascading of counters.
- **OVF flag, bit 31** — Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

The CCCRs are initialized to all 0s on reset.

The events that an enabled performance counter actually counts are selected and filtered by the following flags and fields in the ESCR and CCCR registers and in the qualification order given:

1. The event select and event mask fields in the ESCR select a class of events to be counted and one or more event types within the class, respectively.

2. The OS and USR flags in the ESCR selected the privilege levels at which events will be counted.
3. The ESCR select field of the CCCR selects the ESCR. Since each counter has several ESCRs associated with it, one ESCR must be chosen to select the classes of events that may be counted.
4. The compare and complement flags and the threshold field of the CCCR select an optional threshold to be used in qualifying an event count.
5. The edge flag in the CCCR allows events to be counted only on rising-edge transitions.

The qualification order in the above list implies that the filtered output of one “stage” forms the input for the next. For instance, events filtered using the privilege level flags can be further qualified by the compare and complement flags and the threshold field, and an event that matched the threshold criteria, can be further qualified by edge detection.

The uses of the flags and fields in the CCCRs are discussed in greater detail in Section 22.6.3.5, “Programming the Performance Counters for Non-Retirement Events.”

22.6.3.4 Debug Store (DS) Mechanism

The debug store (DS) mechanism was introduced with processors based on Intel NetBurst microarchitecture to allow various types of information to be collected in memory-resident buffers for use in debugging and tuning programs. The DS mechanism can be used to collect two types of information: branch records and processor event-based sampling (PEBS) records. The availability of the DS mechanism in a processor is indicated with the DS feature flag (bit 21) returned by the CPUID instruction.

See Section 20.4.5, “Branch Trace Store (BTS),” and Section 22.6.3.8, “Processor Event-Based Sampling (PEBS),” for a description of these facilities. Records collected with the DS mechanism are saved in the DS save area. See Section 20.4.9, “BTS and DS Save Area.”

22.6.3.5 Programming the Performance Counters for Non-Retirement Events

The basic steps to program a performance counter and to count events include the following:

1. Select the event or events to be counted.
2. For each event, select an ESCR that supports the event.
3. Match the CCCR Select value and ESCR name to a value listed in Table 22-90; select a CCCR and performance counter.
4. Set up an ESCR for the specific event or events to be counted and the privilege levels at which they are to be counted.
5. Set up the CCCR for the performance counter by selecting the ESCR and the desired event filters.
6. Set up the CCCR for optional cascading of event counts, so that when the selected counter overflows its alternate counter starts.
7. Set up the CCCR to generate an optional performance monitor interrupt (PMI) when the counter overflows. If PMI generation is enabled, the local APIC must be set up to deliver the interrupt to the processor and a handler for the interrupt must be in place.
8. Enable the counter to begin counting.

22.6.3.5.1 Selecting Events to Count

There is a set of at-retirement events for processors based on Intel NetBurst microarchitecture. For each event, setup information is provided. Table 22-91 gives an example of one of the events.

Table 22-91. Event Example

Event Name	Event Parameters	Parameter Value	Description
branch_retired			Counts the retirement of a branch. Specify one or more mask bits to select any combination of branch taken, not-taken, predicted, and mispredicted.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	See Table 15-3 for the addresses of the ESCR MSRs.
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	The counter numbers associated with each ESCR are provided. The performance counters and corresponding CCCRs can be obtained from Table 15-3.
	ESCR Event Select	06H	ESCR[31:25]
	ESCR Event Mask	Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9] Branch Not-taken Predicted Branch Not-taken Mispredicted Branch Taken Predicted Branch Taken Mispredicted
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	
	Requires Additional MSRs for Tagging	No	

Event Parameters are described below.

- **ESCR restrictions** — Lists the ESCRs that can be used to program the event. Typically only one ESCR is needed to count an event.
- **Counter numbers per ESCR** — Lists which performance counters are associated with each ESCR. Table 22-90 gives the name of the counter and CCCR for each counter number. Typically only one counter is needed to count the event.
- **ESCR event select** — Gives the value to be placed in the event select field of the ESCR to select the event.
- **ESCR event mask** — Gives the value to be placed in the Event Mask field of the ESCR to select sub-events to be counted. The parameter value column defines the documented bits with relative bit position offset starting from 0, where the absolute bit position of relative offset 0 is bit 9 of the ESCR. All undocumented bits are reserved and should be set to 0.
- **CCCR select** — Gives the value to be placed in the ESCR select field of the CCCR associated with the counter to select the ESCR to be used to define the event. This value is not the address of the ESCR; it is the number of the ESCR from the Number column in Table 22-90.
- **Event specific notes** — Gives additional information about the event, such as the name of the same or a similar event defined for the P6 family processors.
- **Can support PEBS** — Indicates if PEBS is supported for the event (only supplied for at-retirement events).
- **Requires additional MSR for tagging** — Indicates which if any additional MSRs must be programmed to count the events (only supplied for the at-retirement events).

NOTE

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. The counter values reported are not guaranteed to be

absolutely accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The following procedure shows how to set up a performance counter for basic counting; that is, the counter is set up to count a specified event indefinitely, wrapping around whenever it reaches its maximum count. This procedure is continued through the following four sections.

An event to be counted can be selected as follows:

1. Select the event to be counted.
2. Select the ESCR to be used to select events to be counted from the ESCRs field.
3. Select the number of the counter to be used to count the event from the Counter Numbers Per ESCR field.
4. Determine the name of the counter and the CCCR associated with the counter, and determine the MSR addresses of the counter, CCCR, and ESCR from Table 22-90.
5. Use the WRMSR instruction to write the ESCR Event Select and ESCR Event Mask values into the appropriate fields in the ESCR. At the same time set or clear the USR and OS flags in the ESCR as desired.
6. Use the WRMSR instruction to write the CCCR Select value into the appropriate field in the CCCR.

NOTE

Typically all the fields and flags of the CCCR will be written with one WRMSR instruction; however, in this procedure, several WRMSR writes are used to more clearly demonstrate the uses of the various CCCR fields and flags.

This setup procedure is continued in the next section, Section 22.6.3.5.2, "Filtering Events."

22.6.3.5.2 Filtering Events

Each counter receives up to 4 input lines from the processor hardware from which it is counting events. The counter treats these inputs as binary inputs (input 0 has a value of 1, input 1 has a value of 2, input 2 has a value of 4, and input 3 has a value of 8). When a counter is enabled, it adds this binary input value to the counter value on each clock cycle. For each clock cycle, the value added to the counter can then range from 0 (no event) to 15.

For many events, only the 0 input line is active, so the counter is merely counting the clock cycles during which the 0 input is asserted. However, for some events two or more input lines are used. Here, the counter's threshold setting can be used to filter events. The compare, complement, threshold, and edge fields control the filtering of counter increments by input value.

If the compare flag is set, then a "greater than" or a "less than or equal to" comparison of the input value vs. a threshold value can be made. The complement flag selects "less than or equal to" (flag set) or "greater than" (flag clear). The threshold field selects a threshold value of from 0 to 15. For example, if the complement flag is cleared and the threshold field is set to 6, then any input value of 7 or greater on the 4 inputs to the counter will cause the counter to be incremented by 1, and any value less than 7 will cause an increment of 0 (or no increment) of the counter. Conversely, if the complement flag is set, any value from 0 to 6 will increment the counter and any value from 7 to 15 will not increment the counter. Note that when a threshold condition has been satisfied, the input to the counter is always 1, not the input value that is presented to the threshold filter.

The edge flag provides further filtering of the counter inputs when a threshold comparison is being made. The edge flag is only active when the compare flag is set. When the edge flag is set, the resulting output from the threshold filter (a value of 0 or 1) is used as an input to the edge filter. Each clock cycle, the edge filter examines the last and current input values and sends a count to the counter only when it detects a "rising edge" event; that is, a false-to-true transition. Figure 22-52 illustrates rising edge filtering.

The following procedure shows how to configure a CCCR to filter events using the threshold filter and the edge filter. This procedure is a continuation of the setup procedure introduced in Section 22.6.3.5.1, "Selecting Events to Count."

7. (Optional) To set up the counter for threshold filtering, use the WRMSR instruction to write values in the CCCR compare and complement flags and the threshold field:

- Set the compare flag.
- Set or clear the complement flag for less than or equal to or greater than comparisons, respectively.
- Enter a value from 0 to 15 in the threshold field.

8. (Optional) Select rising edge filtering by setting the CCCR edge flag.

This setup procedure is continued in the next section, Section 22.6.3.5.3, “Starting Event Counting.”

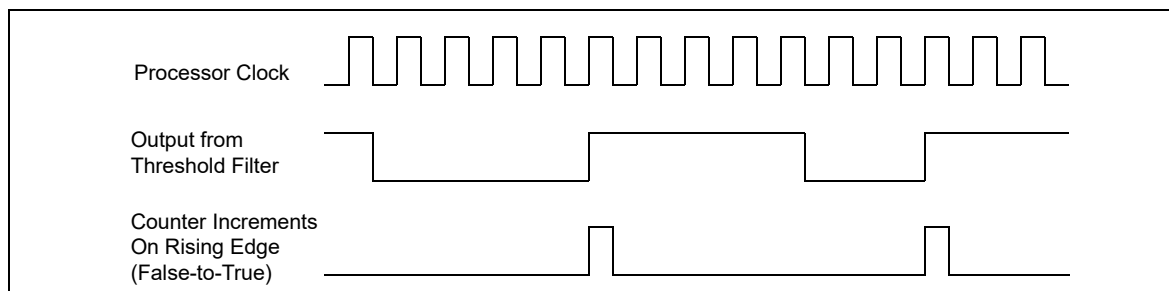


Figure 22-52. Effects of Edge Filtering

22.6.3.5.3 Starting Event Counting

Event counting by a performance counter can be initiated in either of two ways. The typical way is to set the enable flag in the counter’s CCCR. Following the instruction to set the enable flag, event counting begins and continues until it is stopped (see Section 22.6.3.5.5, “Halting Event Counting”).

The following procedural step shows how to start event counting. This step is a continuation of the setup procedure introduced in Section 22.6.3.5.2, “Filtering Events.”

9. To start event counting, use the WRMSR instruction to set the CCCR enable flag for the performance counter.

This setup procedure is continued in the next section, Section 22.6.3.5.4, “Reading a Performance Counter’s Count.”

The second way that a counter can be started by using the cascade feature. Here, the overflow of one counter automatically starts its alternate counter (see Section 22.6.3.5.6, “Cascading Counters”).

22.6.3.5.4 Reading a Performance Counter’s Count

Performance counters can be read using either the RDPMC or RDMSR instructions. The enhanced functions of the RDPMC instruction (including fast read) are described in Section 22.6.3.2, “Performance Counters.” These instructions can be used to read a performance counter while it is counting or when it is stopped.

The following procedural step shows how to read the event counter. This step is a continuation of the setup procedure introduced in Section 22.6.3.5.3, “Starting Event Counting.”

10. To read a performance counters current event count, execute the RDPMC instruction with the counter number obtained from Table 22-90 used as an operand.

This setup procedure is continued in the next section, Section 22.6.3.5.5, “Halting Event Counting.”

22.6.3.5.5 Halting Event Counting

After a performance counter has been started (enabled), it continues counting indefinitely. If the counter overflows (goes one count past its maximum count), it wraps around and continues counting. When the counter wraps around, it sets its OVF flag to indicate that the counter has overflowed. The OVF flag is a sticky flag that indicates that the counter has overflowed at least once since the OVF bit was last cleared.

To halt counting, the CCCR enable flag for the counter must be cleared.

The following procedural step shows how to stop event counting. This step is a continuation of the setup procedure introduced in Section 22.6.3.5.4, “Reading a Performance Counter’s Count.”

11. To stop event counting, execute a WRMSR instruction to clear the CCCR enable flag for the performance counter.

To halt a cascaded counter (a counter that was started when its alternate counter overflowed), either clear the Cascade flag in the cascaded counter's CCCR MSR or clear the OVF flag in the alternate counter's CCCR MSR.

22.6.3.5.6 Cascading Counters

As described in Section 22.6.3.2, "Performance Counters," eighteen performance counters are implemented in pairs. Nine pairs of counters and associated CCCRs are further organized as four blocks: BPU, MS, FLAME, and IQ (see Table 22-90). The first three blocks contain two pairs each. The IQ block contains three pairs of counters (12 through 17) with associated CCCRs (MSR_IQ_CCCR0 through MSR_IQ_CCCR5).

The first 8 counter pairs (0 through 15) can be programmed using ESCRs to detect performance monitoring events. Pairs of ESCRs in each of the four blocks allow many different types of events to be counted. The cascade flag in the CCCR MSR allows nested monitoring of events to be performed by cascading one counter to a second counter located in another pair in the same block (see Figure 22-51 for the location of the flag).

Counters 0 and 1 form the first pair in the BPU block. Either counter 0 or 1 can be programmed to detect an event via MSR_MOB_ESCR0. Counters 0 and 2 can be cascaded in any order, as can counters 1 and 3. It's possible to set up 4 counters in the same block to cascade on two pairs of independent events. The pairing described also applies to subsequent blocks. Since the IQ PUB has two extra counters, cascading operates somewhat differently if 16 and 17 are involved. In the IQ block, counter 16 can only be cascaded from counter 14 (not from 12); counter 14 cannot be cascaded from counter 16 using the CCCR cascade bit mechanism. Similar restrictions apply to counter 17.

Example 22-1. Counting Events

Assume a scenario where counter X is set up to count 200 occurrences of event A; then counter Y is set up to count 400 occurrences of event B. Each counter is set up to count a specific event and overflow to the next counter. In the above example, counter X is preset for a count of -200 and counter Y for a count of -400; this setup causes the counters to overflow on the 200th and 400th counts respectively.

Continuing this scenario, counter X is set up to count indefinitely and wraparound on overflow. This is described in the basic performance counter setup procedure that begins in Section 22.6.3.5.1, "Selecting Events to Count." Counter Y is set up with the cascade flag in its associated CCCR MSR set to 1 and its enable flag set to 0.

To begin the nested counting, the enable bit for the counter X is set. Once enabled, counter X counts until it overflows. At this point, counter Y is automatically enabled and begins counting. Thus counter X overflows after 200 occurrences of event A. Counter Y then starts, counting 400 occurrences of event B before overflowing. When performance counters are cascaded, the counter Y would typically be set up to generate an interrupt on overflow. This is described in Section 22.6.3.5.8, "Generating an Interrupt on Overflow."

The cascading counters mechanism can be used to count a single event. The counting begins on one counter then continues on the second counter after the first counter overflows. This technique doubles the number of event counts that can be recorded, since the contents of the two counters can be added together.

22.6.3.5.7 EXTENDED CASCADING

Extended cascading is a model-specific feature in the Intel NetBurst microarchitecture with CPUID DisplayFamily_DisplayModel 0F_02, 0F_03, 0F_04, 0F_06. This feature uses bit 11 in CCCRs associated with the IQ block. See Table 22-92.

Table 22-92. CCR Names and Bit Positions

CCCR Name:Bit Position	Bit Name	Description
MSR_IQ_CCCR1 2:11	Reserved	
MSR_IQ_CCCR0:11	CASCNT4INT00	Allow counter 4 to cascade into counter 0
MSR_IQ_CCCR3:11	CASCNT5INT03	Allow counter 5 to cascade into counter 3

Table 22-92. CCR Names and Bit Positions (Contd.)

MSR_IQ_CCCR4:11	CASCNT5INTO4	Allow counter 5 to cascade into counter 4
MSR_IQ_CCCR5:11	CASCNT4INTO5	Allow counter 4 to cascade into counter 5

The extended cascading feature can be adapted to the Interrupt based sampling usage model for performance monitoring. However, it is known that performance counters do not generate PMI in cascade mode or extended cascade mode due to an erratum. This erratum applies to processors with CPUID DisplayFamily_DisplayModel signature of 0F_02. For processors with CPUID DisplayFamily_DisplayModel signature of 0F_00 and 0F_01, the erratum applies to processors with stepping encoding greater than 09H.

Counters 16 and 17 in the IQ block are frequently used in processor event-based sampling or at-retirement counting of events indicating a stalled condition in the pipeline. Neither counter 16 or 17 can initiate the cascading of counter pairs using the cascade bit in a CCCR.

Extended cascading permits performance monitoring tools to use counters 16 and 17 to initiate cascading of two counters in the IQ block. Extended cascading from counter 16 and 17 is conceptually similar to cascading other counters, but instead of using CASCADE bit of a CCCR, one of the four CASCNTxINTOy bits is used.

Example 22-2. Scenario for Extended Cascading

A usage scenario for extended cascading is to sample instructions retired on logical processor 1 after the first 4096 instructions retired on logical processor 0. A procedure to program extended cascading in this scenario is outlined below:

1. Write the value 0 to counter 12.
2. Write the value 04000603H to MSR_CRU_ESCR0 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 1).
3. Write the value 04038800H to MSR_IQ_CCCR0. This enables CASCNT4INTO0 and OVF_PMI. An ISR can sample on instruction addresses in this case (do not set ENABLE, or CASCADE).
4. Write the value FFFF000H into counter 16.1.
5. Write the value 0400060CH to MSR_CRU_ESCR2 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 0).
6. Write the value 00039000H to MSR_IQ_CCCR4 (set ENABLE bit, but not OVF_PMI).

Another use for cascading is to locate stalled execution in a multithreaded application. Assume MOB replays in thread B cause thread A to stall. Getting a sample of the stalled execution in this scenario could be accomplished by:

1. Set up counter B to count MOB replays on thread B.
2. Set up counter A to count resource stalls on thread A; set its force overflow bit and the appropriate CASCNTx-INTOy bit.
3. Use the performance monitoring interrupt to capture the program execution data of the stalled thread.

22.6.3.5.8 Generating an Interrupt on Overflow

Any performance counter can be configured to generate a performance monitor interrupt (PMI) if the counter overflows. The PMI interrupt service routine can then collect information about the state of the processor or program when overflow occurred. This information can then be used with a tool like the Intel® VTune™ Performance Analyzer to analyze and tune program performance.

To enable an interrupt on counter overflow, the OVR_PMI flag in the counter's associated CCCR MSR must be set. When overflow occurs, a PMI is generated through the local APIC. (Here, the performance counter entry in the local vector table [LVT] is set up to deliver the interrupt generated by the PMI to the processor.)

The PMI service routine can use the OVF flag to determine which counter overflowed when multiple counters have been configured to generate PMIs. Also, note that these processors mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

When generating interrupts on overflow, the performance counter being used should be preset to value that will cause an overflow after a specified number of events are counted plus 1. The simplest way to select the preset value is to write a negative number into the counter, as described in Section 22.6.3.5.6, “Cascading Counters.” Here, however, if an interrupt is to be generated after 100 event counts, the counter should be preset to minus 100 plus 1 ($-100 + 1$), or -99. The counter will then overflow after it counts 99 events and generate an interrupt on the next (100th) event counted. The difference of 1 for this count enables the interrupt to be generated immediately after the selected event count has been reached, instead of waiting for the overflow to be propagation through the counter.

Because of latency in the microarchitecture between the generation of events and the generation of interrupts on overflow, it is sometimes difficult to generate an interrupt close to an event that caused it. In these situations, the `FORCE_OVF` flag in the CCCR can be used to improve reporting. Setting this flag causes the counter to overflow on every counter increment, which in turn triggers an interrupt after every counter increment.

22.6.3.5.9 Counter Usage Guideline

There are some instances where the user must take care to configure counting logic properly, so that it is not powered down. To use any ESCR, even when it is being used just for tagging, (any) one of the counters that the particular ESCR (or its paired ESCR) can be connected to should be enabled. If this is not done, 0 counts may result. Likewise, to use any counter, there must be some event selected in a corresponding ESCR (other than `no_event`, which generally has a select value of 0).

22.6.3.6 At-Retirement Counting

At-retirement counting provides a means counting only events that represent work committed to architectural state and ignoring work that was performed speculatively and later discarded.

One example of this speculative activity is branch prediction. When a branch misprediction occurs, the results of instructions that were decoded and executed down the mispredicted path are canceled. If a performance counter was set up to count all executed instructions, the count would include instructions whose results were canceled as well as those whose results committed to architectural state.

To provide finer granularity in event counting in these situations, the performance monitoring facilities provided in the Pentium 4 and Intel Xeon processors provide a mechanism for tagging events and then counting only those tagged events that represent committed results. This mechanism is called “at-retirement counting.”

There are predefined at-retirement events and event metrics that can be used to for tagging events when using at retirement counting. The following terminology is used in describing at-retirement counting:

- **Bogus, non-bogus, retire** — In at-retirement event descriptions, the term “bogus” refers to instructions or μ ops that must be canceled because they are on a path taken from a mispredicted branch. The terms “retired” and “non-bogus” refer to instructions or μ ops along the path that results in committed architectural state changes as required by the program being executed. Thus instructions and μ ops are either bogus or non-bogus, but not both. Several of the Pentium 4 and Intel Xeon processors’ performance monitoring events (such as, `Instruction_Retired` and `Uops_Retired`) can count instructions or μ ops that are retired based on the characterization of bogus” versus non-bogus.
- **Tagging** — Tagging is a means of marking μ ops that have encountered a particular performance event so they can be counted at retirement. During the course of execution, the same event can happen more than once per μ op and a direct count of the event would not provide an indication of how many μ ops encountered that event. The tagging mechanisms allow a μ op to be tagged once during its lifetime and thus counted once at retirement. The retired suffix is used for performance metrics that increment a count once per μ op, rather than once per event. For example, a μ op may encounter a cache miss more than once during its life time, but a “Miss Retired” metric (that counts the number of retired μ ops that encountered a cache miss) will increment only once for that μ op. A “Miss Retired” metric would be useful for characterizing the performance of the cache hierarchy for a particular instruction sequence. Details of various performance metrics and how these can be constructed using the Pentium 4 and Intel Xeon processors performance events are provided in the Intel® 64 and IA-32 Architectures Optimization Reference Manual (see Section 1.4, “Related Literature”).
- **Replay** — To maximize performance for the common case, the Intel NetBurst microarchitecture aggressively schedules μ ops for execution before all the conditions for correct execution are guaranteed to be satisfied. In the event that all of these conditions are not satisfied, μ ops must be reissued. The mechanism that the Pentium

4 and Intel Xeon processors use for this reissuing of μ ops is called replay. Some examples of replay causes are cache misses, dependence violations, and unforeseen resource constraints. In normal operation, some number of replays is common and unavoidable. An excessive number of replays is an indication of a performance problem.

- **Assist** — When the hardware needs the assistance of microcode to deal with some event, the machine takes an assist. One example of this is an underflow condition in the input operands of a floating-point operation. The hardware must internally modify the format of the operands in order to perform the computation. Assists clear the entire machine of μ ops before they begin and are costly.

22.6.3.6.1 Using At-Retirement Counting

Processors based on Intel NetBurst microarchitecture allow counting both events and μ ops that encountered a specified event. For a subset of the at-retirement events, a μ op may be tagged when it encounters that event. The tagging mechanisms can be used in Interrupt-based event sampling, and a subset of these mechanisms can be used in PEBS. There are four independent tagging mechanisms, and each mechanism uses a different event to count μ ops tagged with that mechanism:

- **Front-end tagging** — This mechanism pertains to the tagging of μ ops that encountered front-end events (for example, trace cache and instruction counts) and are counted with the `Front_end_event` event.
- **Execution tagging** — This mechanism pertains to the tagging of μ ops that encountered execution events (for example, instruction types) and are counted with the `Execution_Event` event.
- **Replay tagging** — This mechanism pertains to tagging of μ ops whose retirement is replayed (for example, a cache miss) and are counted with the `Replay_event` event. Branch mispredictions are also tagged with this mechanism.
- **No tags** — This mechanism does not use tags. It uses the `Instr_retired` and the `Uops_retired` events.

Each tagging mechanism is independent from all others; that is, a μ op that has been tagged using one mechanism will not be detected with another mechanism's tagged- μ op detector. For example, if μ ops are tagged using the front-end tagging mechanisms, the `Replay_event` will not count those as tagged μ ops unless they are also tagged using the replay tagging mechanism. However, execution tags allow up to four different types of μ ops to be counted at retirement through execution tagging.

The independence of tagging mechanisms does not hold when using PEBS. When using PEBS, only one tagging mechanism should be used at a time.

Certain kinds of μ ops that cannot be tagged, including I/O, uncacheable and locked accesses, returns, and far transfers.

There are performance monitoring events that support at-retirement counting: specifically the `Front_end_event`, `Execution_event`, `Replay_event`, `Inst_retired`, and `Uops_retired` events. The following sections describe the tagging mechanisms for using these events to tag μ op and count tagged μ ops.

22.6.3.6.2 Tagging Mechanism for Front_end_event

The `Front_end_event` counts μ ops that have been tagged as encountering any of the following events:

- **μ op decode events** — Tagging μ ops for μ op decode events requires specifying bits in the ESCR associated with the performance-monitoring event, `Uop_type`.
- **Trace cache events** — Tagging μ ops for trace cache events may require specifying certain bits in the `MSR_TC_PRECISE_EVENT` MSR.

The MSRs that are supported by the front-end tagging mechanism must be set and one or both of the `NBOGUS` and `BOGUS` bits in the `Front_end_event` event mask must be set to count events. None of the events currently supported requires the use of the `MSR_TC_PRECISE_EVENT` MSR.

22.6.3.6.3 Tagging Mechanism For Execution_event

The execution tagging mechanism differs from other tagging mechanisms in how it causes tagging. One *upstream* ESCR is used to specify an event to detect and to specify a tag value (bits 5 through 8) to identify that event. A

second *downstream* ESCR is used to detect μ ops that have been tagged with that tag value identifier using `Execution_event` for the event selection.

The upstream ESCR that counts the event must have its tag enable flag (bit 4) set and must have an appropriate tag value mask entered in its tag value field. The 4-bit tag value mask specifies which of tag bits should be set for a particular μ op. The value selected for the tag value should coincide with the event mask selected in the downstream ESCR. For example, if a tag value of 1 is set, then the event mask of `NBOGUS0` should be enabled, correspondingly in the downstream ESCR. The downstream ESCR detects and counts tagged μ ops. The normal (not tag value) mask bits in the downstream ESCR specify which tag bits to count. If any one of the tag bits selected by the mask is set, the related counter is incremented by one. The tag enable and tag value bits are irrelevant for the downstream ESCR used to select the `Execution_event`.

The four separate tag bits allow the user to simultaneously but distinctly count up to four execution events at retirement. (This applies for interrupt-based event sampling. There are additional restrictions for PEBS as noted in Section 22.6.3.8.3, "Setting Up the PEBS Buffer.") It is also possible to detect or count combinations of events by setting multiple tag value bits in the upstream ESCR or multiple mask bits in the downstream ESCR. For example, use a tag value of 3H in the upstream ESCR and use `NBOGUS0/NBOGUS1` in the downstream ESCR event mask.

22.6.3.7 Tagging Mechanism for `Replay_event`

The replay mechanism enables tagging of μ ops for a subset of all replays before retirement. Use of the replay mechanism requires selecting the type of μ op that may experience the replay in the `MSR_PEBS_MATRIX_VERT` MSR and selecting the type of event in the `MSR_PEBS_ENABLE` MSR. Replay tagging must also be enabled with the `UOP_Tag` flag (bit 24) in the `MSR_PEBS_ENABLE` MSR.

The replay tags defined in Table A-5 also enable Processor Event-Based Sampling (PEBS, see Section 20.4.9). Each of these replay tags can also be used in normal sampling by not setting Bit 24 nor Bit 25 in `IA_32_PEBS_ENABLE_MSR`. Each of these metrics requires that the `Replay_Event` be used to count the tagged μ ops.

22.6.3.8 Processor Event-Based Sampling (PEBS)

The debug store (DS) mechanism in processors based on Intel NetBurst microarchitecture allow two types of information to be collected for use in debugging and tuning programs: PEBS records and BTS records. See Section 20.4.5, "Branch Trace Store (BTS)," for a description of the BTS mechanism.

PEBS permits the saving of precise architectural information associated with one or more performance events in the precise event records buffer, which is part of the DS save area (see Section 20.4.9, "BTS and DS Save Area"). To use this mechanism, a counter is configured to overflow after it has counted a preset number of events. After the counter overflows, the processor copies the current state of the general-purpose and EFLAGS registers and instruction pointer into a record in the precise event records buffer. The processor then resets the count in the performance counter and restarts the counter. When the precise event records buffer is nearly full, an interrupt is generated, allowing the precise event records to be saved. A circular buffer is not supported for precise event records.

PEBS is supported only for a subset of the at-retirement events: `Execution_event`, `Front_end_event`, and `Replay_event`. Also, PEBS can only be carried out using the one performance counter, the `MSR_IQ_COUNTER4` MSR.

In processors based on Intel Core microarchitecture, a similar PEBS mechanism is also supported using `IA32_PMC0` and `IA32_PERFVTSEL0` MSRs (See Section 22.6.2.4).

22.6.3.8.1 Detection of the Availability of the PEBS Facilities

The DS feature flag (bit 21) returned by the `CPUID` instruction indicates (when set) the availability of the DS mechanism in the processor, which supports the PEBS (and BTS) facilities. When this bit is set, the following PEBS facilities are available:

- The `PEBS_UNAVAILABLE` flag in the `IA32_MISC_ENABLE` MSR indicates (when clear) the availability of the PEBS facilities, including the `MSR_PEBS_ENABLE` MSR.
- The enable PEBS flag (bit 24) in the `MSR_PEBS_ENABLE` MSR allows PEBS to be enabled (set) or disabled (clear).

- The IA32_DS_AREA MSR can be programmed to point to the DS save area.

22.6.3.8.2 Setting Up the DS Save Area

Section 20.4.9.2, “Setting Up the DS Save Area,” describes how to set up and enable the DS save area. This procedure is common for PEBS and BTS.

22.6.3.8.3 Setting Up the PEBS Buffer

Only the MSR_IQ_COUNTER4 performance counter can be used for PEBS. Use the following procedure to set up the processor and this counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, and precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area (see Figure 20-5) to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS flag (bit 24) in MSR_PEBS_ENABLE MSR.
3. Set up the MSR_IQ_COUNTER4 performance counter and its associated CCCR and one or more ESCRs for PEBS.

22.6.3.8.4 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the non-precise event-based sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 20.4.9.5, “Writing the DS Interrupt Service Routine,” for guidelines for writing the DS ISR.

22.6.3.8.5 Other DS Mechanism Implications

The DS mechanism is not available in the SMM. It is disabled on transition to the SMM mode. Similarly the DS mechanism is disabled on the generation of a machine check exception and is cleared on processor RESET and INIT.

The DS mechanism is available in real address mode.

22.6.3.9 Operating System Implications

The DS mechanism can be used by the operating system as a debugging extension to facilitate failure analysis. When using this facility, a 25 to 30 times slowdown can be expected due to the effects of the trace store occurring on every taken branch.

Depending upon intended usage, the instruction pointers that are part of the branch records or the PEBS records need to have an association with the corresponding process. One solution requires the ability for the DS specific operating system module to be chained to the context switch. A separate buffer can then be maintained for each process of interest and the MSR pointing to the configuration area saved and setup appropriately on each context switch.

If the BTS facility has been enabled, then it must be disabled and state stored on transition of the system to a sleep state in which processor context is lost. The state must be restored on return from the sleep state.

It is required that an interrupt gate be used for the DS interrupt as opposed to a trap gate to prevent the generation of an endless interrupt loop.

Pages that contain buffers must have mappings to the same physical address for all processes/logical processors, such that any change to CR3 will not change DS addresses. If this requirement cannot be satisfied (that is, the feature is enabled on a per thread/process basis), then the operating system must ensure that the feature is enabled/disabled appropriately in the context switch code.

22.6.4 Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

The performance monitoring capability of processors based on Intel NetBurst microarchitecture and supporting Intel Hyper-Threading Technology is similar to that described in Section 22.6.3. However, the capability is extended so that:

- Performance counters can be programmed to select events qualified by logical processor IDs.
- Performance monitoring interrupts can be directed to a specific logical processor within the physical processor.

The sections below describe performance counters, event qualification by logical processor ID, and special purpose bits in ESCRs/CCCRs. They also describe MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_TC_PRECISE_EVENT.

22.6.4.1 ESCR MSRs

Figure 22-53 shows the layout of an ESCR MSR in processors supporting Intel Hyper-Threading Technology.

The functions of the flags and fields are as follows:

- **T1_USR flag, bit 0** — When set, events are counted when thread 1 (logical processor 1) is executing at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.

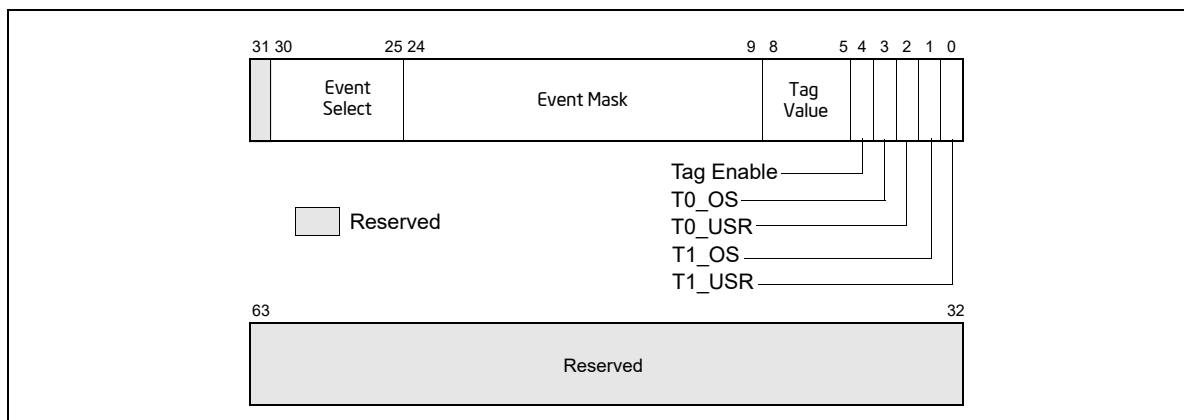


Figure 22-53. Event Selection Control Register (ESCR) for the Pentium 4 Processor, Intel® Xeon® Processor, and Intel® Xeon® Processor MP Supporting Hyper-Threading Technology

- **T1_OS flag, bit 1** — When set, events are counted when thread 1 (logical processor 1) is executing at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the T1_OS and T1_USR flags are set, thread 1 events are counted at all privilege levels.)
- **T0_USR flag, bit 2** — When set, events are counted when thread 0 (logical processor 0) is executing at a CPL of 1, 2, or 3.
- **T0_OS flag, bit 3** — When set, events are counted when thread 0 (logical processor 0) is executing at CPL of 0. (When both the T0_OS and T0_USR flags are set, thread 0 events are counted at all privilege levels.)
- **Tag enable, bit 4** — When set, enables tagging of μ ops to assist in at-retirement event counting; when clear, disables tagging. See Section 22.6.3.6, "At-Retirement Counting."
- **Tag value field, bits 5 through 8** — Selects a tag value to associate with a μ op to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24** — Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30** — Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

The T0_OS and T0_USR flags and the T1_OS and T1_USR flags allow event counting and sampling to be specified for a specific logical processor (0 or 1) within an Intel Xeon processor MP (See also: Section 11.4.5, “Identifying Logical Processors in an MP System,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).

Not all performance monitoring events can be detected within an Intel Xeon processor MP on a per logical processor basis (see Section 22.6.4.4, “Performance Monitoring Events”). Some sub-events (specified by an event mask bits) are counted or sampled without regard to which logical processor is associated with the detected event.

22.6.4.2 CCCR MSRs

Figure 22-54 shows the layout of a CCCR MSR in processors supporting Intel Hyper-Threading Technology. The functions of the flags and fields are as follows:

- **Enable flag, bit 12** — When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset
- **ESCR select field, bits 13 through 15** — Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.
- **Active thread field, bits 16 and 17** — Enables counting depending on which logical processors are active (executing a thread). This field enables filtering of events based on the state (active or inactive) of the logical processors. The encodings of this field are as follows:
 - 00** — None. Count only when neither logical processor is active.
 - 01** — Single. Count only when one logical processor is active (either 0 or 1).
 - 10** — Both. Count only when both logical processors are active.
 - 11** — Any. Count when either logical processor is active.

A halted logical processor or a logical processor in the “wait for SIPI” state is considered inactive.
- **Compare flag, bit 18** — When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.

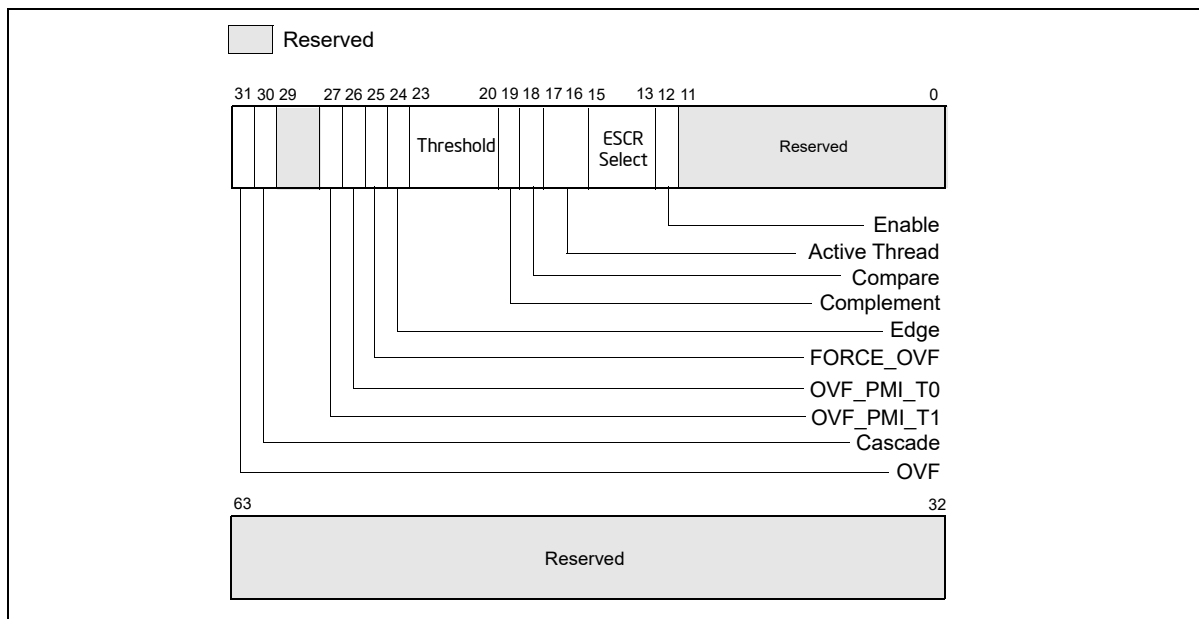


Figure 22-54. Counter Configuration Control Register (CCCR)

- **Complement flag, bit 19** — Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being

delivered to the performance counter (see Section 22.6.3.5.2, “Filtering Events”). The compare flag is not active unless the compare flag is set.

- **Threshold field, bits 20 through 23** — Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 22.6.3.5.2, “Filtering Events”).
- **Edge flag, bit 24** — When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.
- **FORCE_OVF flag, bit 25** — When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI_T0 flag, bit 26** — When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 0 when the counter overflows occurs; when clear, disables PMI generation for logical processor 0. Note that the PMI is generate on the next event count after the counter has overflowed.
- **OVF_PMI_T1 flag, bit 27** — When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 1 when the counter overflows occurs; when clear, disables PMI generation for logical processor 1. Note that the PMI is generate on the next event count after the counter has overflowed.
- **Cascade flag, bit 30** — When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 22.6.3.2, “Performance Counters,” for further details); when clear, disables cascading of counters.
- **OVF flag, bit 31** — Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

22.6.4.3 IA32_PEBS_ENABLE MSR

In a processor supporting Intel Hyper-Threading Technology and based on the Intel NetBurst microarchitecture, PEBS is enabled and qualified with two bits in the MSR_PEBS_ENABLE MSR: bit 25 (ENABLE_PEBS_MY_THR) and 26 (ENABLE_PEBS_OTH_THR) respectively. These bits do not explicitly identify a specific logical processor by logic processor ID(T0 or T1); instead, they allow a software agent to enable PEBS for subsequent threads of execution on the same logical processor on which the agent is running (“my thread”) or for the other logical processor in the physical package on which the agent is not running (“other thread”).

PEBS is supported for only a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can be carried out only with two performance counters: MSR_IQ_CCCR4 (MSR address 370H) for logical processor 0 and MSR_IQ_CCCR5 (MSR address 371H) for logical processor 1.

Performance monitoring tools should use a processor affinity mask to bind the kernel mode components that need to modify the ENABLE_PEBS_MY_THR and ENABLE_PEBS_OTH_THR bits in the MSR_PEBS_ENABLE MSR to a specific logical processor. This is to prevent these kernel mode components from migrating between different logical processors due to OS scheduling.

22.6.4.4 Performance Monitoring Events

When Intel Hyper-Threading Technology is active, many performance monitoring events can be can be qualified by the logical processor ID, which corresponds to bit 0 of the initial APIC ID. This allows for counting an event in any or all of the logical processors. However, not all the events have this logic processor specificity, or thread specificity.

Here, each event falls into one of two categories:

- **Thread specific (TS)** — The event can be qualified as occurring on a specific logical processor.
- **Thread independent (TI)** — The event cannot be qualified as being associated with a specific logical processor.

If for example, a TS event occurred in logical processor T0, the counting of the event (as shown in Table 22-93) depends only on the setting of the T0_USR and T0_OS flags in the ESCR being used to set up the event counter. The T1_USR and T1_OS flags have no effect on the count.

Table 22-93. Effect of Logical Processor and CPL Qualification for Logical-Processor-Specific (TS) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while T1 in USR	Counts while T1 in OS or USR	Counts while T1 in OS
T0_OS/T0_USR = 01	Counts while T0 in USR	Counts while T0 in USR or T1 in USR	Counts while (a) T0 in USR or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 11	Counts while T0 in OS or USR	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in OS
T0_OS/T0_USR = 10	Counts T0 in OS	Counts T0 in OS or T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS

When a bit in the event mask field is TI, the effect of specifying bit-0-3 of the associated ESCR are described in Table 15-6. For events that are marked as TI, the effect of selectively specifying T0_USR, T0_OS, T1_USR, T1_OS bits is shown in Table 22-94.

Table 22-94. Effect of Logical Processor and CPL Qualification for Non-logical-Processor-specific (TI) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 01	Counts while (a) T0 in USR or (b) T1 in USR	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 11	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 0	Counts while (a) T0 in OS or (b) T1 in OS	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS

22.6.4.5 Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

22.6.4.5.1 Non-Halted Clockticks

Use the following procedure to program ESCRs and CCCRs to obtain non-halted clockticks on processors based on Intel NetBurst microarchitecture:

1. Select an ESCR for the global_power_events and specify the RUNNING sub-event mask and the desired T0_OS/T0_USR/T1_OS/T1_USR bits for the targeted processor.
2. Select an appropriate counter.
3. Enable counting in the CCCR for that counter by setting the enable bit.

22.6.4.5.2 Non-Sleep Clockticks

Performance monitoring counters can be configured to count clockticks whenever the performance monitoring hardware is not powered-down. To count Non-sleep Clockticks with a performance-monitoring counter, do the following:

1. Select one of the 18 counters.
2. Select any of the ESCRs whose events the selected counter can count. Set its event select to anything other than "no_event"; the counter may be disabled if this is not done.

3. Turn threshold comparison on in the CCCR by setting the compare bit to “1”.
4. Set the threshold to “15” and the complement to “1” in the CCCR. Since no event can exceed this threshold, the threshold condition is met every cycle and the counter counts every cycle. Note that this overrides any qualification (e.g., by CPL) specified in the ESCR.
5. Enable counting in the CCCR for the counter by setting the enable bit.

In most cases, the counts produced by the non-halted and non-sleep metrics are equivalent if the physical package supports one logical processor and is not placed in a power-saving state. Operating systems may execute an HLT instruction and place a physical processor in a power-saving state.

On processors that support Intel Hyper-Threading Technology (Intel HT Technology), each physical package can support two or more logical processors. Current implementation of Intel HT Technology provides two logical processors for each physical processor. While both logical processors can execute two threads simultaneously, one logical processor may halt to allow the other logical processor to execute without sharing execution resources between two logical processors.

Non-halted Clockticks can be set up to count the number of processor clock cycles for each logical processor whenever the logical processor is not halted (the count may include some portion of the clock cycles for that logical processor to complete a transition to a halted state). Physical processors that support Intel HT Technology enter into a power-saving state if all logical processors halt.

The Non-sleep Clockticks mechanism uses a filtering mechanism in CCCRs. The mechanism will continue to increment as long as one logical processor is not halted or in a power-saving state. Applications may cause a processor to enter into a power-saving state by using an OS service that transfers control to an OS's idle loop. The idle loop then may place the processor into a power-saving state after an implementation-dependent period if there is no work for the processor.

22.6.5 Performance Monitoring and Dual-Core Technology

The performance monitoring capability of dual-core processors duplicates the microarchitectural resources of a single-core processor implementation. Each processor core has dedicated performance monitoring resources.

In the case of Pentium D processor, each logical processor is associated with dedicated resources for performance monitoring. In the case of Pentium processor Extreme edition, each processor core has dedicated resources, but two logical processors in the same core share performance monitoring resources (see Section 22.6.4, “Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture”).

22.6.6 Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache

The 64-bit Intel Xeon processor MP with up to 8-MByte L3 cache has a CUID signature of family [0FH], model [03H or 04H]. Performance monitoring capabilities available to Pentium 4 and Intel Xeon processors with the same values (see Section 22.1 and Section 22.6.4) apply to the 64-bit Intel Xeon processor MP with an L3 cache.

The level 3 cache is connected between the system bus and IOQ through additional control logic. See Figure 22-55.

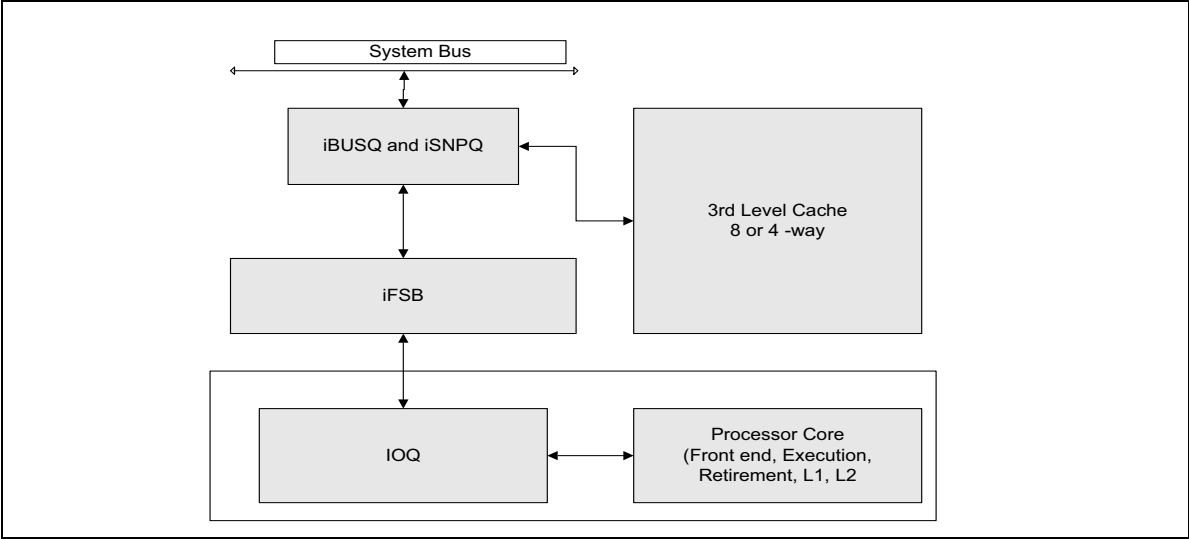


Figure 22-55. Block Diagram of 64-bit Intel® Xeon® Processor MP with 8-MByte L3

Additional performance monitoring capabilities and facilities unique to 64-bit Intel Xeon processor MP with an L3 cache are described in this section. The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs), each dedicated to a specific event. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values.

The lower 32-bits of the MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers. These performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

The performance monitoring capabilities consist of four events. These are:

- **IBUSQ event** — This event detects the occurrence of micro-architectural conditions related to the iBUSQ unit. It provides two MSRs: MSR_IFSB_IBUSQ0 and MSR_IFSB_IBUSQ1. Configure sub-event qualification and enable/disable functions using the high 32 bits of these MSRs. The low 32 bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32 bits. See Figure 22-56.

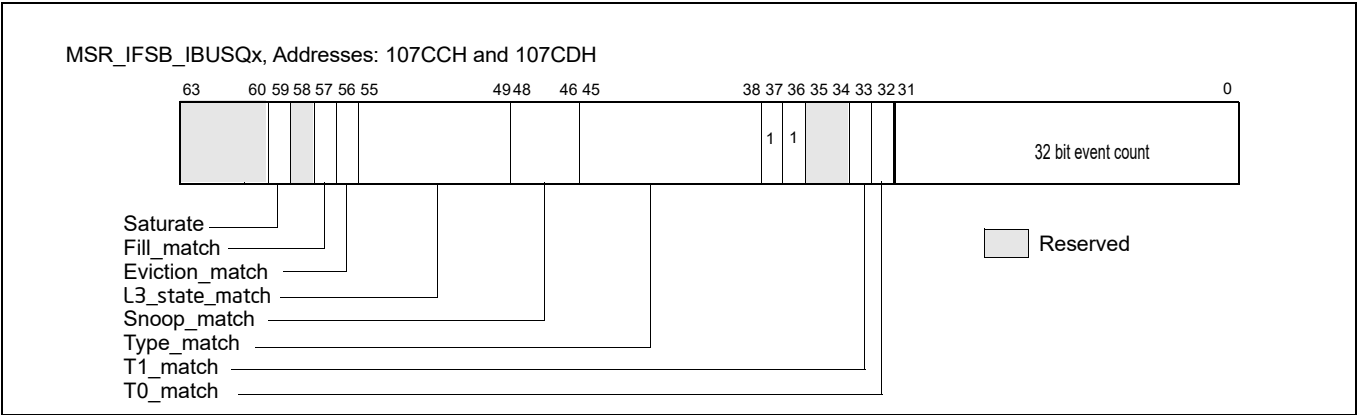


Figure 22-56. MSR_IFSB_IBUSQx, Addresses: 107CCH and 107CDH

- **ISNPQ event** — This event detects the occurrence of microarchitectural conditions related to the iSNPQ unit. It provides two MSRs: MSR_IFSB_ISNPQ0 and MSR_IFSB_ISNPQ1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the MSRs. The low 32-bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32-bits. See Figure 22-57.

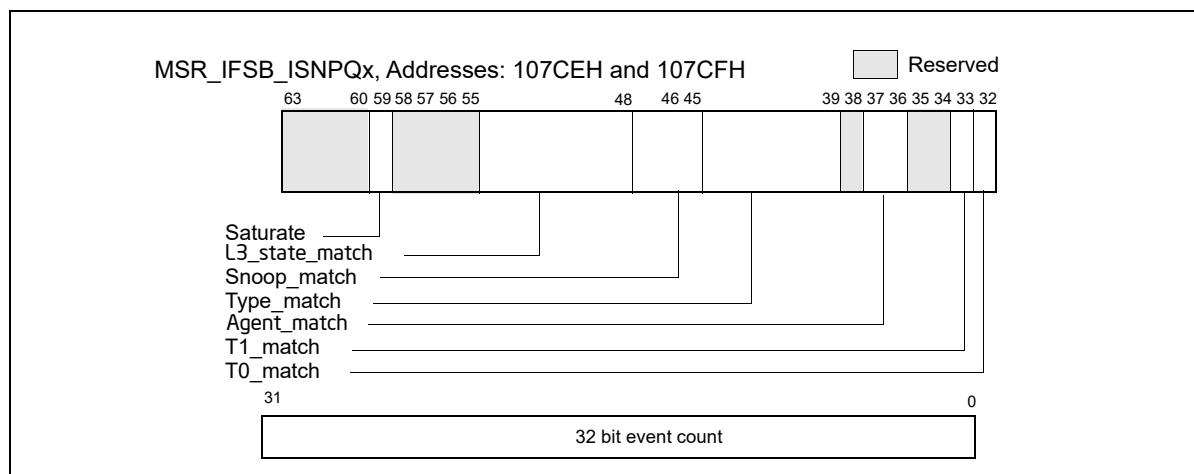


Figure 22-57. MSR_IFSB_ISNPQx, Addresses: 107CEH and 107CFH

- EFSB event** — This event can detect the occurrence of micro-architectural conditions related to the iFSB unit or system bus. It provides two MSRs: MSR_EFSB_DRDY0 and MSR_EFSB_DRDY1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the 64-bit MSR. The low 32-bit act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the qualification bits in the upper 32-bits of the MSR. See Figure 22-58.

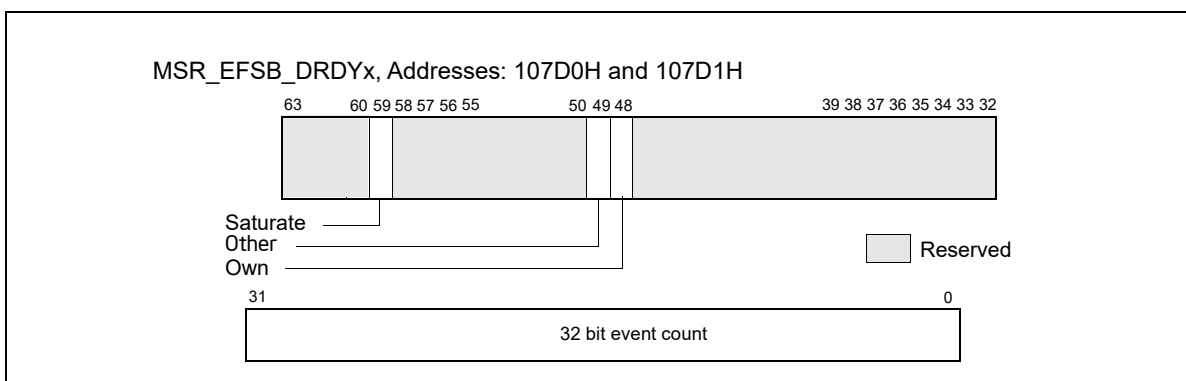


Figure 22-58. MSR_EFSB_DRDYx, Addresses: 107D0H and 107D1H

- IBUSQ Latency event** — This event accumulates weighted cycle counts for latency measurement of transactions in the iBUSQ unit. The count is enabled by setting MSR_IFSB_CTRL6[bit 26] to 1; the count freezes after software sets MSR_IFSB_CTRL6[bit 26] to 0. MSR_IFSB_CNTR7 acts as a 64-bit event counter for this event. See Figure 22-59.

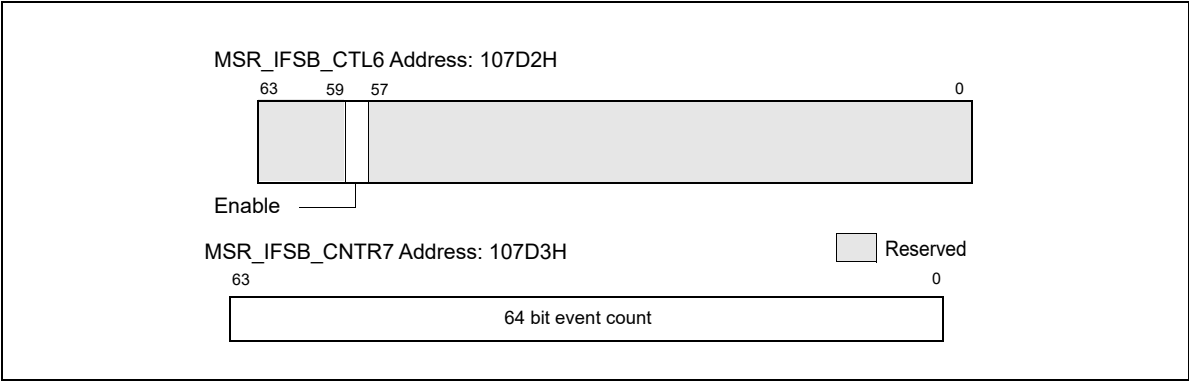


Figure 22-59. MSR_IFSB_CTL6, Address: 107D2H; MSR_IFSB_CNTR7, Address: 107D3H

22.6.7 Performance Monitoring on L3 and Caching Bus Controller Sub-Systems

The Intel Xeon processor 7400 series and Dual-Core Intel Xeon processor 7100 series employ a distinct L3/caching bus controller sub-system. These sub-system have a unique set of performance monitoring capability and programming interfaces that are largely common between these two processor families.

Intel Xeon processor 7400 series are based on 45 nm enhanced Intel Core microarchitecture. The CUID signature is indicated by DisplayFamily_DisplayModel value of 06_1DH (see the CUID instruction in Chapter 3, "Instruction Set Reference, A-L," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). Intel Xeon processor 7400 series have six processor cores that share an L3 cache.

Dual-Core Intel Xeon processor 7100 series are based on Intel NetBurst microarchitecture, have a CUID signature of family [0FH], model [06H] and a unified L3 cache shared between two cores. Each core in an Intel Xeon processor 7100 series supports Intel Hyper-Threading Technology, providing two logical processors per core.

Both Intel Xeon processor 7400 series and Intel Xeon processor 7100 series support multi-processor configurations using system bus interfaces. In Intel Xeon processor 7400 series, the L3/caching bus controller sub-system provides three Simple Direct Interface (SDI) to service transactions originated the XQ-replacement SDI logic in each dual-core modules. In Intel Xeon processor 7100 series, the IOQ logic in each processor core is replaced with a Simple Direct Interface (SDI) logic. The L3 cache is connected between the system bus and the SDI through additional control logic. See Figure 22-60 for the block configuration of six processor cores and the L3/Caching bus controller sub-system in Intel Xeon processor 7400 series. Figure 22-60 shows the block configuration of two processor cores (four logical processors) and the L3/Caching bus controller sub-system in Intel Xeon processor 7100 series.

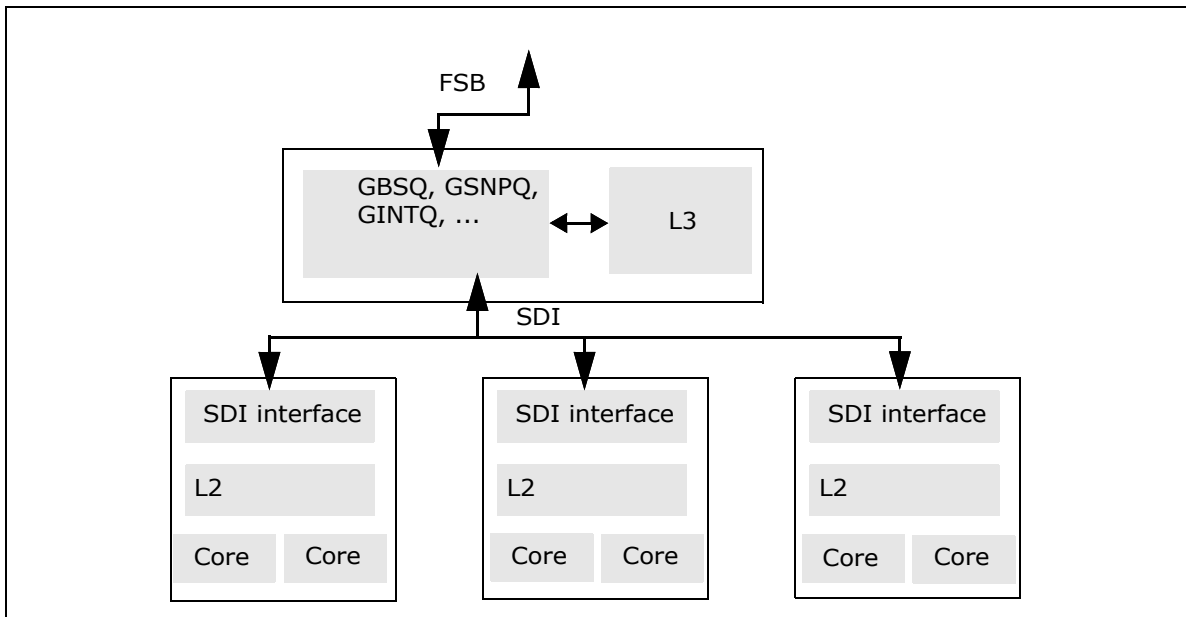


Figure 22-60. Block Diagram of the Intel® Xeon® Processor 7400 Series

Almost all of the performance monitoring capabilities available to processor cores with the same CPUID signatures (see Section 22.1 and Section 22.6.4) apply to Intel Xeon processor 7100 series. The MSRs used by performance monitoring interface are shared between two logical processors in the same processor core.

The performance monitoring capabilities available to processor with DisplayFamily_DisplayModel signature 06_17H also apply to Intel Xeon processor 7400 series. Each processor core provides its own set of MSRs for performance monitoring interface.

The IOQ_allocation and IOQ_active_entries events are not supported in Intel Xeon processor 7100 series and 7400 series. Additional performance monitoring capabilities applicable to the L3/caching bus controller sub-system are described in this section.

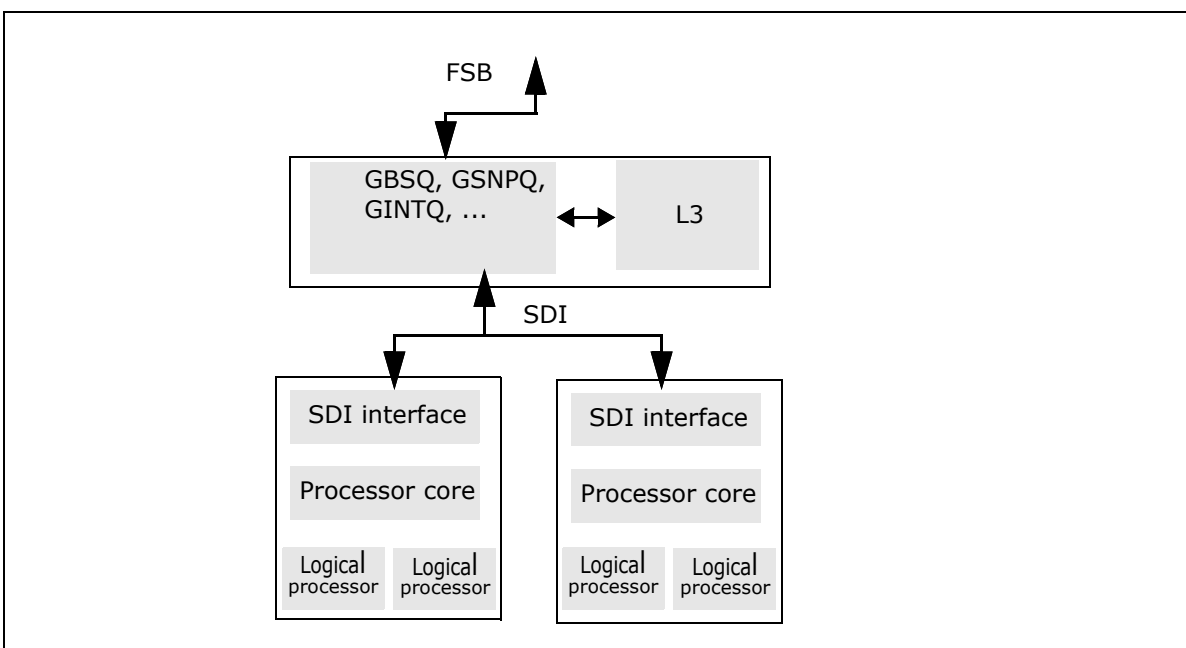


Figure 22-61. Block Diagram of the Intel® Xeon® Processor 7100 Series

22.6.7.1 Overview of Performance Monitoring with L3/Caching Bus Controller

The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs). There are eight event select/counting MSRs that are dedicated to counting events associated with specified microarchitectural conditions. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values. In addition, an MSR MSR_EMON_L3_GL_CTL provides simplified interface to control freezing, resetting, re-enabling operation of any combination of these event select/counting MSRs.

The eight MSRs dedicated to count occurrences of specific conditions are further divided to count three sub-classes of microarchitectural conditions:

- Two MSRs (MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1) are dedicated to counting GBSQ events. Up to two GBSQ events can be programmed and counted simultaneously.
- Two MSRs (MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3) are dedicated to counting GSNPQ events. Up to two GBSQ events can be programmed and counted simultaneously.
- Four MSRs (MSR_EMON_L3_CTR_CTL4, MSR_EMON_L3_CTR_CTL5, MSR_EMON_L3_CTR_CTL6, and MSR_EMON_L3_CTR_CTL7) are dedicated to counting external bus operations.

The bit fields in each of eight MSRs share the following common characteristics:

- Bits 63:32 is the event control field that includes an event mask and other bit fields that control counter operation. The event mask field specifies details of the microarchitectural condition, and its definition differs across GBSQ, GSNPQ, FSB.
- Bits 31:0 is the event count field. If the specified condition is met during each relevant clock domain of the event logic, the matched condition signals the counter logic to increment the associated event count field. The lower 32-bits of these 8 MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers.

In Dual-Core Intel Xeon processor 7100 series, the uncore performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

In Intel Xeon processor 7400 series, RDPMC with ECX between 2 and 9 can be used to access the eight uncore performance counter/control registers.

22.6.7.2 GBSQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1 is given in Figure 22-62. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following eight attributes:

- Agent_Select (bits 35:32): The definition of this field differs slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series, each bit specifies a logical processor in the physical package. The lower two bits corresponds to two logical processors in the first processor core, the upper two bits corresponds to two logical processors in the second processor core. 0FH encoding matches transactions from any logical processor.

For Intel Xeon processor 7400 series, each bit of [34:32] specifies the SDI logic of a dual-core module as the originator of the transaction. A value of 0111B in bits [35:32] specifies transaction from any processor core.

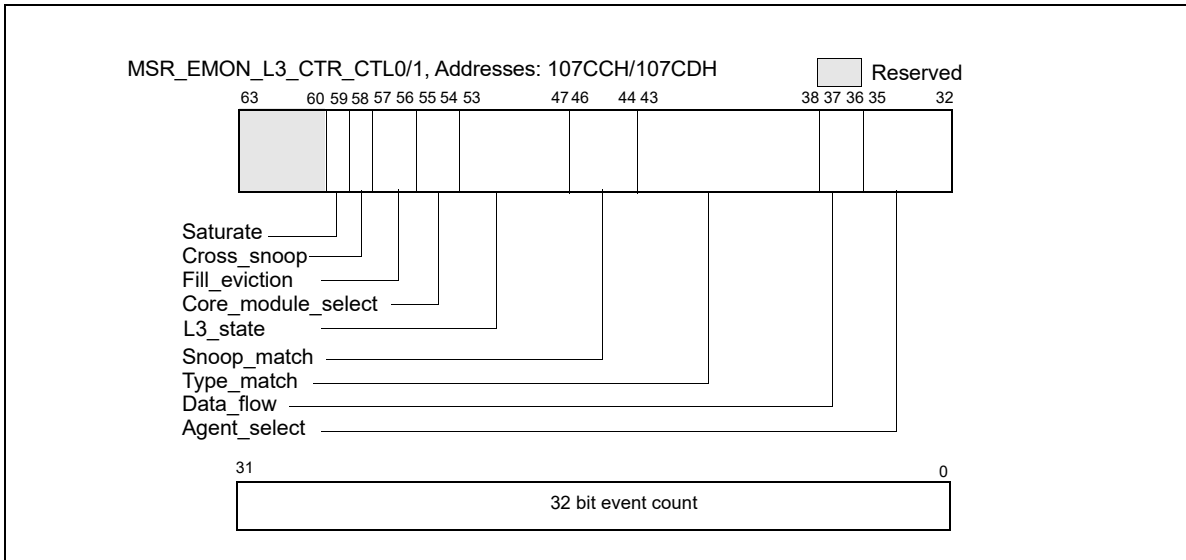


Figure 22-62. MSR_EMON_L3_CTL0/1, Addresses: 107CCH/107CDH

- Data_Flow (bits 37:36): Bit 36 specifies demand transactions, bit 37 specifies prefetch transactions.
- Type_Match (bits 43:38): Specifies transaction types. If all six bits are set, event count will include all transaction types.
- Snoop_Match: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- L3_State (bits 53:47): Each bit specifies an L2 coherency state.
- Core_Module_Select (bits 55:54): The valid encodings for L3 lookup differ slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series,

- 00B: Match transactions from any core in the physical package
- 01B: Match transactions from this core only
- 10B: Match transactions from the other core in the physical package
- 11B: Match transaction from both cores in the physical package

For Intel Xeon processor 7400 series,

- 00B: Match transactions from any dual-core module in the physical package
- 01B: Match transactions from this dual-core module only
- 10B: Match transactions from either one of the other two dual-core modules in the physical package
- 11B: Match transaction from more than one dual-core modules in the physical package

- Fill_Eviction (bits 57:56): The valid encodings are
 - 00B: Match any transactions
 - 01B: Match transactions that fill L3
 - 10B: Match transactions that fill L3 without an eviction
 - 11B: Match transaction fill L3 with an eviction
- Cross_Snoop (bit 58): The encodings are
 - 0B: Match any transactions
 - 1B: Match cross snoop transactions

For each counting clock domain, if all eight attributes match, event logic signals to increment the event count field.

22.6.7.3 GSNPQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3 is given in Figure 22-63. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following six attributes:

- **Agent_Select** (bits 37:32): The definition of this field differs slightly between Intel Xeon processor 7100 and 7400.
- For Intel Xeon processor 7100 series, each of the lowest 4 bits specifies a logical processor in the physical package. The lowest two bits corresponds to two logical processors in the first processor core, the next two bits corresponds to two logical processors in the second processor core. Bit 36 specifies other symmetric agent transactions. Bit 37 specifies central agent transactions. 3FH encoding matches transactions from any logical processor.

For Intel Xeon processor 7400 series, each of the lowest 3 bits specifies a dual-core module in the physical package. Bit 37 specifies central agent transactions.

- **Type_Match** (bits 43:38): Specifies transaction types. If all six bits are set, event count will include any transaction types.
- **Snoop_Match**: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- **L2_State** (bits 53:47): Each bit specifies an L3 coherency state.
- **Core_Module_Select** (bits 56:54): Bit 56 enables Core_Module_Select matching. If bit 56 is clear, Core_Module_Select encoding is ignored. The valid encodings for the lower two bits (bit 55, 54) differ slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series, if bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are

- 00B: Match transactions from only one core (irrespective which core) in the physical package
- 01B: Match transactions from this core and not the other core
- 10B: Match transactions from the other core in the physical package, but not this core
- 11B: Match transaction from both cores in the physical package

For Intel Xeon processor 7400 series, if bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are

- 00B: Match transactions from only one dual-core module (irrespective which module) in the physical package.
- 01B: Match transactions from one or more dual-core modules.
- 10B: Match transactions from two or more dual-core modules.
- 11B: Match transaction from all three dual-core modules in the physical package.

- **Block_Snoop** (bit 57): specifies blocked snoop.

For each counting clock domain, if all six attributes match, event logic signals to increment the event count field.

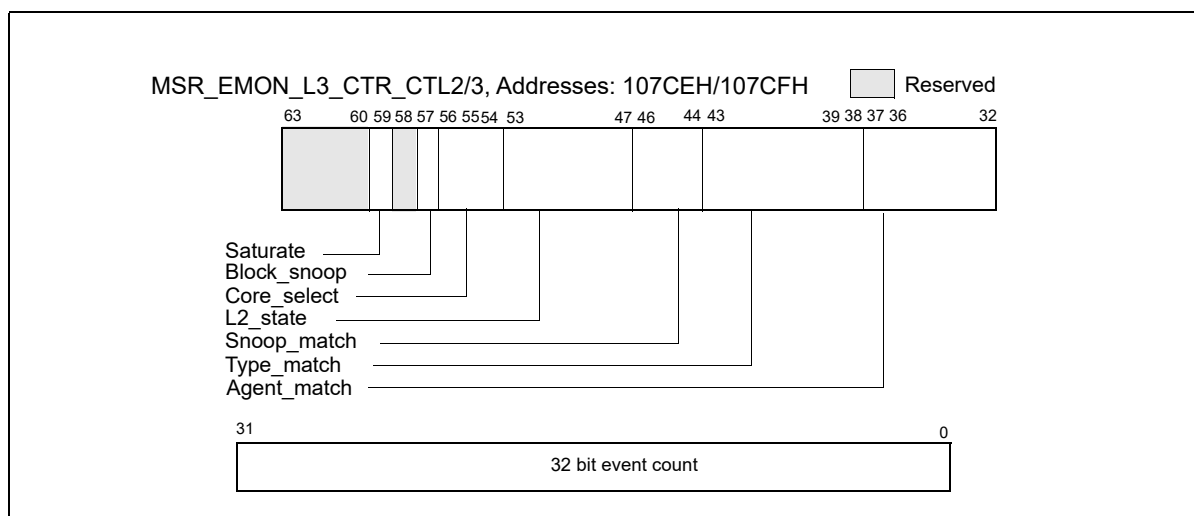


Figure 22-63. MSR_EMON_L3_CTR_CTL2/3, Addresses: 107CEH/107CFH

22.6.7.4 FSB Event Interface

The layout of MSR_EMON_L3_CTR_CTL4 through MSR_EMON_L3_CTR_CTL7 is given in Figure 22-64. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) is organized as follows:

- Bit 58: must set to 1.
- FSB_Submask (bits 57:32): Specifies FSB-specific sub-event mask.

The FSB sub-event mask defines a set of independent attributes. The event logic signals to increment the associated event count field if one of the attribute matches. Some of the sub-event mask bit counts durations. A duration event increments at most once per cycle.

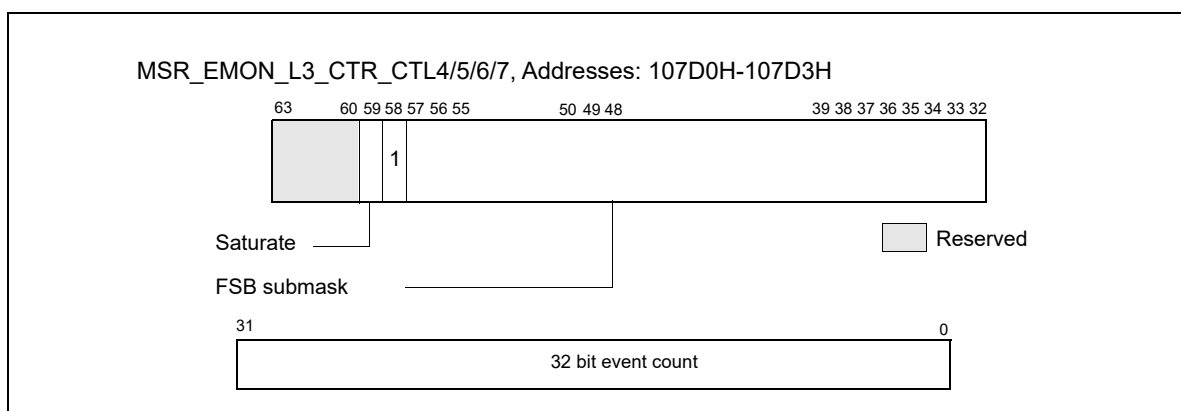


Figure 22-64. MSR_EMON_L3_CTR_CTL4/5/6/7, Addresses: 107D0H-107D3H

22.6.7.4.1 FSB Sub-Event Mask Interface

- FSB_type (bit 37:32): Specifies different FSB transaction types originated from this physical package.
- FSB_L_clear (bit 38): Count clean snoop results from any source for transaction originated from this physical package.
- FSB_L_hit (bit 39): Count HIT snoop results from any source for transaction originated from this physical package.

- FSB_L_hitm (bit 40): Count HITM snoop results from any source for transaction originated from this physical package.
- FSB_L_defer (bit 41): Count DEFER responses to this processor's transactions.
- FSB_L_retry (bit 42): Count RETRY responses to this processor's transactions.
- FSB_L_snoop_stall (bit 43): Count snoop stalls to this processor's transactions.
- FSB_DBSY (bit 44): Count DBSY assertions by this processor (without a concurrent DRDY).
- FSB_DRDY (bit 45): Count DRDY assertions by this processor.
- FSB_BNR (bit 46): Count BNR assertions by this processor.
- FSB_IOQ_empty (bit 47): Counts each bus clocks when the IOQ is empty.
- FSB_IOQ_full (bit 48): Counts each bus clocks when the IOQ is full.
- FSB_IOQ_active (bit 49): Counts each bus clocks when there is at least one entry in the IOQ.
- FSB_WW_data (bit 50): Counts back-to-back write transaction's data phase.
- FSB_WW_issue (bit 51): Counts back-to-back write transaction request pairs issued by this processor.
- FSB_WR_issue (bit 52): Counts back-to-back write-read transaction request pairs issued by this processor.
- FSB_RW_issue (bit 53): Counts back-to-back read-write transaction request pairs issued by this processor.
- FSB_other_DBSY (bit 54): Count DBSY assertions by another agent (without a concurrent DRDY).
- FSB_other_DRDY (bit 55): Count DRDY assertions by another agent.
- FSB_other_snoop_stall (bit 56): Count snoop stalls on the FSB due to another agent.
- FSB_other_BNR (bit 57): Count BNR assertions from another agent.

22.6.7.5 Common Event Control Interface

The MSR_EMON_L3_GL_CTL MSR provides simplified access to query overflow status of the GBSQ, GSNPQ, FSB event counters. It also provides control bit fields to freeze, unfreeze, or reset those counters. The following bit fields are supported:

- GL_freeze_cmd (bit 0): Freeze the event counters specified by the GL_event_select field.
- GL_unfreeze_cmd (bit 1): Unfreeze the event counters specified by the GL_event_select field.
- GL_reset_cmd (bit 2): Clear the event count field of the event counters specified by the GL_event_select field. The event select field is not affected.
- GL_event_select (bit 23:16): Selects one or more event counters to subject to specified command operations indicated by bits 2:0. Bit 16 corresponds to MSR_EMON_L3_CTR_CTL0, bit 23 corresponds to MSR_EMON_L3_CTR_CTL7.
- GL_event_status (bit 55:48): Indicates the overflow status of each event counters. Bit 48 corresponds to MSR_EMON_L3_CTR_CTL0, bit 55 corresponds to MSR_EMON_L3_CTR_CTL7.

In the event control field (bits 63:32) of each MSR, if the saturate control (bit 59, see Figure 22-62 for example) is set, the event logic forces the value FFFF_FFFFH into the event count field instead of incrementing it.

22.6.8 Performance Monitoring (P6 Family Processor)

The P6 family processors provide two 40-bit performance counters, allowing two types of events to be monitored simultaneously. These can either count events or measure duration. When counting events, a counter increments each time a specified event takes place or a specified number of events takes place. When measuring duration, it counts the number of processor clocks that occur while a specified condition is true. The counters can count events or measure durations that occur at any privilege level.

NOTE

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The performance-monitoring counters are supported by four MSRs: the performance event select MSRs (PerfEvtSel0 and PerfEvtSel1) and the performance counter MSRs (PerfCtr0 and PerfCtr1). These registers can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0. The PerfCtr0 and PerfCtr1 MSRs can be read from any privilege level using the RDPNC (read performance-monitoring counters) instruction.

NOTE

The PerfEvtSel0, PerfEvtSel1, PerfCtr0, and PerfCtr1 MSRs and the events listed for P6 family processors are model-specific for P6 family processors. They are not guaranteed to be available in other IA-32 processors.

22.6.8.1 PerfEvtSel0 and PerfEvtSel1 MSRs

The PerfEvtSel0 and PerfEvtSel1 MSRs control the operation of the performance-monitoring counters, with one register used to set up each counter. They specify the events to be counted, how they should be counted, and the privilege levels at which counting should take place. Figure 22-65 shows the flags and fields in these MSRs.

The functions of the flags and fields in the PerfEvtSel0 and PerfEvtSel1 MSRs are as follows:

- **Event select field (bits 0 through 7)** — Selects the event logic unit to detect certain microarchitectural conditions.
- **Unit mask (UMASK) field (bits 8 through 15)** — Further qualifies the event logic unit selected in the event select field to detect a specific microarchitectural condition. For example, for some cache events, the mask is used as a MESI-protocol qualifier of cache states.

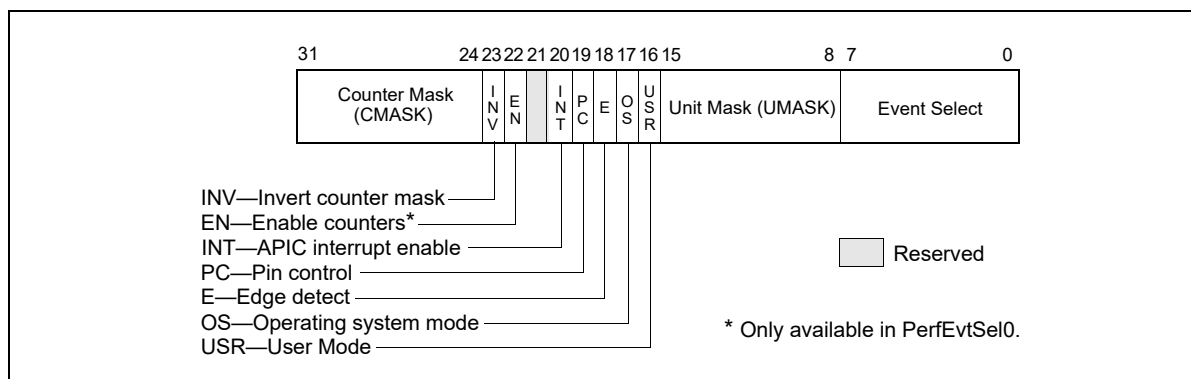


Figure 22-65. PerfEvtSel0 and PerfEvtSel1 MSRs

- **USR (user mode) flag (bit 16)** — Specifies that events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.
- **OS (operating system mode) flag (bit 17)** — Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USR flag.
- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of events. The processor counts the number of deasserted to asserted transitions of any condition that can be expressed by the other fields. The mechanism is limited in that it does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

- **PC (pin control) flag (bit 19)** — When set, the processor toggles the PMi pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PMi pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.
- **INT (APIC interrupt enable) flag (bit 20)** — When set, the processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — This flag is only present in the PerfEvtSel0 MSR. When set, performance counting is enabled in both performance-monitoring counters; when clear, both counters are disabled.
- **INV (invert) flag (bit 23)** — When set, inverts the counter-mask (CMASK) comparison, so that both greater than or equal to and less than comparisons can be made (0: greater than or equal; 1: less than). Note if counter-mask is programmed to zero, INV flag is ignored.
- **Counter mask (CMASK) field (bits 24 through 31)** — When nonzero, the processor compares this mask to the number of events counted during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented. This mask can be used to count events only if multiple occurrences happen per clock (for example, two or more instructions retired per clock). If the counter-mask field is 0, then the counter is incremented each cycle by the number of events that occurred that cycle.

22.6.8.2 PerfCtr0 and PerfCtr1 MSRs

The performance-counter MSRs (PerfCtr0 and PerfCtr1) contain the event or duration counts for the selected events being counted. The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

The WRMSR instruction cannot arbitrarily write to the performance-monitoring counter MSRs (PerfCtr0 and PerfCtr1). Instead, the lower-order 32 bits of each MSR may be written with any value, and the high-order 8 bits are sign-extended according to the value of bit 31. This operation allows writing both positive and negative values to the performance counters.

22.6.8.3 Starting and Stopping the Performance-Monitoring Counters

The performance-monitoring counters are started by writing valid setup information in the PerfEvtSel0 and/or PerfEvtSel1 MSRs and setting the enable counters flag in the PerfEvtSel0 MSR. If the setup is valid, the counters begin counting following the execution of a WRMSR instruction that sets the enable counter flag. The counters can be stopped by clearing the enable counters flag or by clearing all the bits in the PerfEvtSel0 and PerfEvtSel1 MSRs. Counter 1 alone can be stopped by clearing the PerfEvtSel1 MSR.

22.6.8.4 Event and Time-Stamp Monitoring Software

To use the performance-monitoring counters and time-stamp counter, the operating system needs to provide an event-monitoring device driver. This driver should include procedures for handling the following operations:

- Feature checking.
- Initialize and start counters.
- Stop counters.
- Read the event counters.
- Read the time-stamp counter.

The event monitor feature determination procedure must check whether the current processor supports the performance-monitoring counters and time-stamp counter. This procedure compares the family and model of the processor returned by the CPUID instruction with those of processors known to support performance monitoring. (The Pentium and P6 family processors support performance counters.) The procedure also checks the MSR and TSC flags returned to register EDX by the CPUID instruction to determine if the MSRs and the RDTSC instruction are supported.

The initialize and start counters procedure sets the PerfEvtSel0 and/or PerfEvtSel1 MSRs for the events to be counted and the method used to count them and initializes the counter MSRs (PerfCtr0 and PerfCtr1) to starting counts. The stop counters procedure stops the performance counters (see Section 22.6.8.3, “Starting and Stopping the Performance-Monitoring Counters”).

The read counters procedure reads the values in the PerfCtr0 and PerfCtr1 MSRs, and a read time-stamp counter procedure reads the time-stamp counter. These procedures would be provided in lieu of enabling the RDTSC and RDPMC instructions that allow application code to read the counters.

22.6.8.5 Monitoring Counter Overflow

The P6 family processors provide the option of generating a local APIC interrupt when a performance-monitoring counter overflows. This mechanism is enabled by setting the interrupt enable flag in either the PerfEvtSel0 or the PerfEvtSel1 MSR. The primary use of this option is for statistical performance sampling.

To use this option, the operating system should do the following things on the processor for which performance events are required to be monitored:

- Provide an interrupt vector for handling the counter-overflow interrupt.
- Initialize the APIC PERF local vector entry to enable handling of performance-monitor counter overflow events.
- Provide an entry in the IDT that points to a stub exception handler that returns without executing any instructions.
- Provide an event monitor driver that provides the actual interrupt handler and modifies the reserved IDT entry to point to its interrupt routine.

When interrupted by a counter overflow, the interrupt handler needs to perform the following actions:

- Save the instruction pointer (EIP register), code-segment selector, TSS segment selector, counter values and other relevant information at the time of the interrupt.
- Reset the counter to its initial setting and return from the interrupt.

An event monitor application utility or another application program can read the information collected for analysis of the performance of the profiled application.

22.6.9 Performance Monitoring (Pentium Processors)

The Pentium processor provides two 40-bit performance counters, which can be used to count events or measure duration. The counters are supported by three MSRs: the control and event select MSR (CESR) and the performance counter MSRs (CTR0 and CTR1). These can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0.

Each counter has an associated external pin (PM0/BP0 and PM1/BP1), which can be used to indicate the state of the counter to external hardware.

NOTES

The CESR, CTR0, and CTR1 MSRs and the events listed for Pentium processors are model-specific for the Pentium processor.

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

22.6.9.1 Control and Event Select Register (CESR)

The 32-bit control and event select MSR (CESR) controls the operation of performance-monitoring counters CTR0 and CTR1 and the associated pins (see Figure 22-66). To control each counter, the CESR register contains a 6-bit event select field (ES0 and ES1), a pin control flag (PC0 and PC1), and a 3-bit counter control field (CC0 and CC1). The functions of these fields are as follows:

- **ES0 and ES1 (event select) fields (bits 0-5, bits 16-21)** — Selects (by entering an event code in the field) up to two events to be monitored.

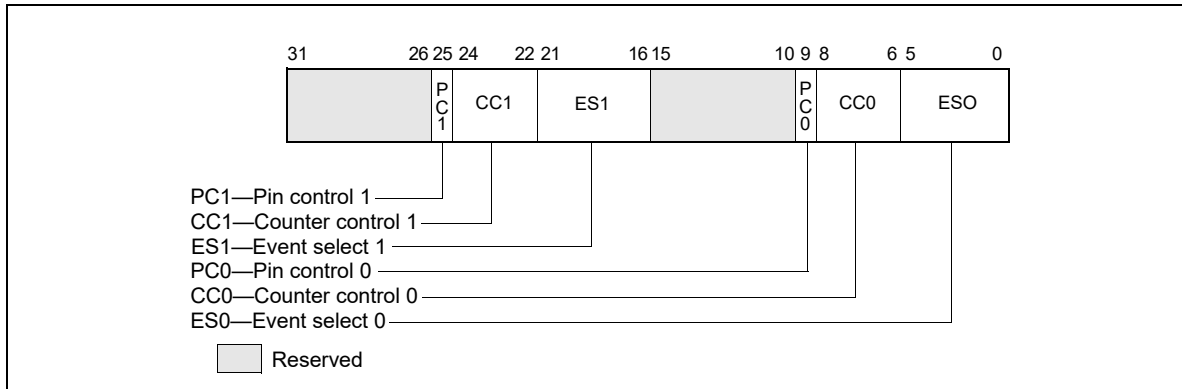


Figure 22-66. CESR MSR (Pentium Processor Only)

- **CC0 and CC1 (counter control) fields (bits 6-8, bits 22-24)** — Controls the operation of the counter. Control codes are as follows:

- 000 — Count nothing (counter disabled).
- 001 — Count the selected event while CPL is 0, 1, or 2.
- 010 — Count the selected event while CPL is 3.
- 011 — Count the selected event regardless of CPL.
- 100 — Count nothing (counter disabled).
- 101 — Count clocks (duration) while CPL is 0, 1, or 2.
- 110 — Count clocks (duration) while CPL is 3.
- 111 — Count clocks (duration) regardless of CPL.

The highest order bit selects between counting events and counting clocks (duration); the middle bit enables counting when the CPL is 3; and the low-order bit enables counting when the CPL is 0, 1, or 2.

- **PC0 and PC1 (pin control) flags (bits 9, 25)** — Selects the function of the external performance-monitoring counter pin (PM0/BP0 and PM1/BP1). Setting one of these flags to 1 causes the processor to assert its associated pin when the counter has overflowed; setting the flag to 0 causes the pin to be asserted when the counter has been incremented. These flags permit the pins to be individually programmed to indicate the overflow or incremented condition. The external signaling of the event on the pins will lag the internal event by a few clocks as the signals are latched and buffered.

While a counter need not be stopped to sample its contents, it must be stopped and cleared or preset before switching to a new event. It is not possible to set one counter separately. If only one event needs to be changed, the CESR register must be read, the appropriate bits modified, and all bits must then be written back to CESR. At reset, all bits in the CESR register are cleared.

22.6.9.2 Use of the Performance-Monitoring Pins

When performance-monitor pins PM0/BP0 and/or PM1/BP1 are configured to indicate when the performance-monitor counter has incremented and an "occurrence event" is being counted, the associated pin is asserted (high) each time the event occurs. When a "duration event" is being counted, the associated PM pin is asserted for the

entire duration of the event. When the performance-monitor pins are configured to indicate when the counter has overflowed, the associated PM pin is asserted when the counter has overflowed.

When the PM0/BP0 and/or PM1/BP1 pins are configured to signal that a counter has incremented, it should be noted that although the counters may increment by 1 or 2 in a single clock, the pins can only indicate that the event occurred. Moreover, since the internal clock frequency may be higher than the external clock frequency, a single external clock may correspond to multiple internal clocks.

A “count up to” function may be provided when the event pin is programmed to signal an overflow of the counter. Because the counters are 40 bits, a carry out of bit 39 indicates an overflow. A counter may be preset to a specific value less than $2^{40} - 1$. After the counter has been enabled and the prescribed number of events has transpired, the counter will overflow.

Approximately 5 clocks later, the overflow is indicated externally and appropriate action, such as signaling an interrupt, may then be taken.

The PM0/BP0 and PM1/BP1 pins also serve to indicate breakpoint matches during in-circuit emulation, during which time the counter increment or overflow function of these pins is not available. After RESET, the PM0/BP0 and PM1/BP1 pins are configured for performance monitoring, however a hardware debugger may reconfigure these pins to indicate breakpoint matches.

22.6.9.3 Events Counted

Events that performance-monitoring counters can be set to count and record (using CTR0 and CTR1) are divided in two categories: occurrence and duration:

- **Occurrence events** — Counts are incremented each time an event takes place. If PM0/BP0 or PM1/BP1 pins are used to indicate when a counter increments, the pins are asserted each clock counters increment. But if an event happens twice in one clock, the counter increments by 2 (the pins are asserted only once).
- **Duration events** — Counters increment the total number of clocks that the condition is true. When used to indicate when counters increment, PM0/BP0 and/or PM1/BP1 pins are asserted for the duration.

22.7 COUNTING CLOCKS

The count of cycles, also known as clockticks, forms the basis for measuring how long a program takes to execute. Clockticks are also used as part of efficiency ratios like cycles per instruction (CPI). Processor clocks may stop ticking under circumstances like the following:

- The processor is halted when there is nothing for the CPU to do. For example, the processor may halt to save power while the computer is servicing an I/O request. When Intel Hyper-Threading Technology is enabled, both logical processors must be halted for performance-monitoring counters to be powered down.
- The processor is asleep as a result of being halted or because of a power-management scheme. There are different levels of sleep. In the some deep sleep levels, the time-stamp counter stops counting.

In addition, processor core clocks may undergo transitions at different ratios relative to the processor's bus clock frequency. Some of the situations that can cause processor core clock to undergo frequency transitions include:

- TM2 transitions.
- Enhanced Intel SpeedStep Technology transitions (P-state transitions).

For Intel processors that support TM2, the processor core clocks may operate at a frequency that differs from the Processor Base frequency (as indicated by processor frequency information reported by CPUID instruction). See Section 22.7.2 for more detail.

Due to the above considerations there are several important clocks referenced in this manual:

- **Base Clock** — The frequency of this clock is the frequency of the processor when the processor is not in turbo mode, and not being throttled via Intel SpeedStep.
- **Maximum Clock** — This is the maximum frequency of the processor when turbo mode is at the highest point.
- **Bus Clock** — These clockticks increment at a fixed frequency and help coordinate the bus on some systems.

- **Core Crystal Clock** — This is a clock that runs at fixed frequency; it coordinates the clocks on all packages across the system.
- **Non-halted Clockticks** — Measures clock cycles in which the specified logical processor is not halted and is not in any power-saving state. When Intel Hyper-Threading Technology is enabled, ticks can be measured on a per-logical-processor basis. There are also performance events on dual-core processors that measure clockticks per logical processor when the processor is not halted.
- **Non-sleep Clockticks** — Measures clock cycles in which the specified physical processor is not in a sleep mode or in a power-saving state. These ticks cannot be measured on a logical-processor basis.
- **Time-stamp Counter** — See Section 20.17, “Time-Stamp Counter.”
- **Reference Clockticks** — TM2 or Enhanced Intel SpeedStep technology are two examples of processor features that can cause processor core clockticks to represent non-uniform tick intervals due to change of bus ratios. Performance events that counts clockticks of a constant reference frequency was introduced Intel Core Duo and Intel Core Solo processors. The mechanism is further enhanced on processors based on Intel Core microarchitecture.

Some processor models permit clock cycles to be measured when the physical processor is not in deep sleep (by using the time-stamp counter and the RDTSC instruction). Note that such ticks cannot be measured on a per-logical-processor basis. See Section 20.17, “Time-Stamp Counter,” for detail on processor capabilities.

The first two methods use performance counters and can be set up to cause an interrupt upon overflow (for sampling). They may also be useful where it is easier for a tool to read a performance counter than to use a time stamp counter (the timestamp counter is accessed using the RDTSC instruction).

For applications with a significant amount of I/O, there are two ratios of interest:

- **Non-halted CPI** — Non-halted clockticks/instructions retired measures the CPI for phases where the CPU was being used. This ratio can be measured on a logical-processor basis when Intel Hyper-Threading Technology is enabled.
- **Nominal CPI** — Time-stamp counter ticks/instructions retired measures the CPI over the duration of a program, including those periods when the machine halts while waiting for I/O.

22.7.1 Non-Halted Reference Clockticks

Software can use UnHalted Reference Cycles on either a general purpose performance counter using event mask 0x3C and UMASK 0x01 or on fixed function performance counter 2 to count at a constant rate. These events count at a consistent rate irrespective of P-state, TM2, or frequency transitions that may occur to the processor. The UnHalted Reference Cycles event may count differently on the general purpose event and fixed counter.

22.7.2 Cycle Counting and Opportunistic Processor Operation

As a result of the state transitions due to opportunistic processor performance operation (see Chapter 17, “Power and Thermal Management”), a logical processor or a processor core can operate at frequency different from the Processor Base frequency.

The following items are expected to hold true irrespective of when opportunistic processor operation causes state transitions:

- The time stamp counter operates at a fixed-rate frequency of the processor.
- The IA32_MPERF counter increments at a fixed frequency irrespective of any transitions caused by opportunistic processor operation.
- The IA32_FIXED_CTR2 counter increments at the same TSC frequency irrespective of any transitions caused by opportunistic processor operation.
- The Local APIC timer operation is unaffected by opportunistic processor operation.
- The TSC, IA32_MPERF, and IA32_FIXED_CTR2 operate at close to the maximum non-turbo frequency, which is equal to the product of scalable bus frequency and maximum non-turbo ratio.

22.7.3 Determining the Processor Base Frequency

For Intel processors in which the nominal core crystal clock frequency is enumerated in CPUID.15H:ECX and the core crystal clock ratio is encoded in CPUID.15H, the nominal TSC frequency can be determined by using the following equation:

$$\text{Nominal TSC frequency} = (\text{CPUID.15H:ECX}[31:0] * \text{CPUID.15H:EBX}[31:0]) \div \text{CPUID.15H:EAX}[31:0]$$

For Intel processors in which CPUID.15H:EBX[31:0] $\div$ CPUID.15H:EAX[31:0] is enumerated but CPUID.15H:ECX is not enumerated, Table 22-95 can be used to look up the nominal core crystal clock frequency.

Table 22-95. Nominal Core Crystal Clock Frequency

Processor Families/Processor Number Series ¹	Nominal Core Crystal Clock Frequency
Intel® Xeon® Scalable Processor Family with CPUID signature 06_55H.	25 MHz
6th and 7th generation Intel® Core™ processors and Intel® Xeon® W Processor Family.	24 MHz
Next Generation Intel Atom® processors based on Goldmont Microarchitecture with CPUID signature 06_5CH (does not include Intel Xeon processors).	19.2 MHz

NOTES:

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.

22.7.3.1 For Intel® Processors Based on Sandy Bridge, Ivy Bridge, Haswell, and Broadwell Microarchitectures

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by a bus speed of 100 MHz.

22.7.3.2 For Intel® Processors Based on Nehalem Microarchitecture

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by a bus speed of 133.33 MHz.

22.7.3.3 For Intel Atom® Processors Based on Silvermont Microarchitecture (Including Intel Processors Based on Airmont Microarchitecture)

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by the scalable bus frequency. The scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] for Intel Atom processors based on the Silvermont microarchitecture, and in bit field MSR_FSB_FREQ[3:0] for processors based on the Airmont microarchitecture; see Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

22.7.3.4 For Intel® Core™ 2 Processor Family and for Intel® Xeon® Processors Based on Intel Core Microarchitecture

For processors based on Intel Core microarchitecture, the scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] at (0CDH), see Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4. The maximum resolved bus ratio can be read from the following bit field:

- If XE operation is disabled, the maximum resolved bus ratio can be read in MSR_PLATFORM_ID[12:8]. It corresponds to the Processor Base frequency.

- If XE operation is enabled, the maximum resolved bus ratio is given in MSR_PERF_STATUS[44:40], it corresponds to the maximum XE operation frequency configured by BIOS.

XE operation of an Intel 64 processor is implementation specific. XE operation can be enabled only by BIOS. If MSR_PERF_STATUS[31] is set, XE operation is enabled. The MSR_PERF_STATUS[31] field is read-only.

22.8 IA32_PERF_CAPABILITIES MSR ENUMERATION

The layout of IA32_PERF_CAPABILITIES MSR is shown in Figure 22-67; it provides enumeration of a variety of interfaces:

- IA32_PERF_CAPABILITIES.LBR_FMT[bits 5:0]: encodes the LBR format, details are described in Section 20.4.8.1.
- IA32_PERF_CAPABILITIES.PEBSTrap[6]: Trap/Fault-like indicator of PEBS recording assist; see Section 22.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBSArchRegs[7]: Indicator of PEBS assist save architectural registers; see Section 22.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBS_FMT[bits 11:8]: Specifies the encoding of the layout of PEBS records; see Section 22.6.2.4.2.
- IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[12]: Indicates IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if 1. See Section 22.8.1.
- IA32_PERF_CAPABILITIES.FULL_WRITE[13]: Indicates the processor supports IA32_A_PMCx interface for updating bits 32 and above of IA32_PMCx; see Section 22.2.8.
- IA32_PERF_CAPABILITIES.PEBS_BASELINE [bit 14]: If set, the following is true:
 - The IA32_PEBS_ENABLE MSR (address 3F1H) exists and all architecturally enumerated fixed and general-purpose counters have corresponding bits in IA32_PEBS_ENABLE that enable generation of PEBS records. The general-purpose counter bits start at bit IA32_PEBS_ENABLE[0], and the fixed counter bits start at bit IA32_PEBS_ENABLE[32].
 - The format of the PEBS record is enumerated by IA32_PERF_CAPABILITIES.PEBS_FMT; see Section 22.6.2.4.2.
 - Extended PEBS is supported. All counters support the PEBS facility, and all events (both precise and non-precise) can generate PEBS records when PEBS is enabled for that counter. Note that not all events may be available on all counters.
 - Adaptive PEBS is supported. The PEBS_DATA_CFG MSR (address 3F2H) and adaptive record enable bits (IA32_PERFEVTSELx.Adaptive_Record and IA32_FIXED_CTR_CTRL.FCx_Adaptive_Record) are supported. The definition of the PEBS_DATA_CFG MSR, including which bits are supported and how they affect the record, is enumerated by IA32_PERF_CAPABILITIES.PEBS_FMT. See Section 22.9.2.3.
 - NOTE: Software is recommended to feature PEBS Baseline when the following is true: IA32_PERF_CAPABILITIES.PEBS_BASELINE[14] && IA32_PERF_CAPABILITIES.PEBS_FMT[11:8] ≥ 4.
- IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE[15]: If set, indicates that the architecture provides built in support for TMA L1 metrics through the PERF_METRICS MSR. See Section 22.3.9.3.
- IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16]: If set on parts that enumerate support for Intel PT (CPUID.07H.00H:EBX[25]=1), setting IA32_PEBS_ENABLE.PEBS_OUTPUT to 01B will result in PEBS output being written into the Intel PT trace stream. See Section 22.5.5.2.
- IA32_PERF_CAPABILITIES.PEBS_TIMING_INFO[17]: If set, indicates that the processor supports the Timed PEBS capability. See Section 22.9.9.
- IA32_PERF_CAPABILITIES.RDPMC_METRICS_CLEAR[19]: If set, indicates that the processor supports RDPMC Metrics Clear Mode.

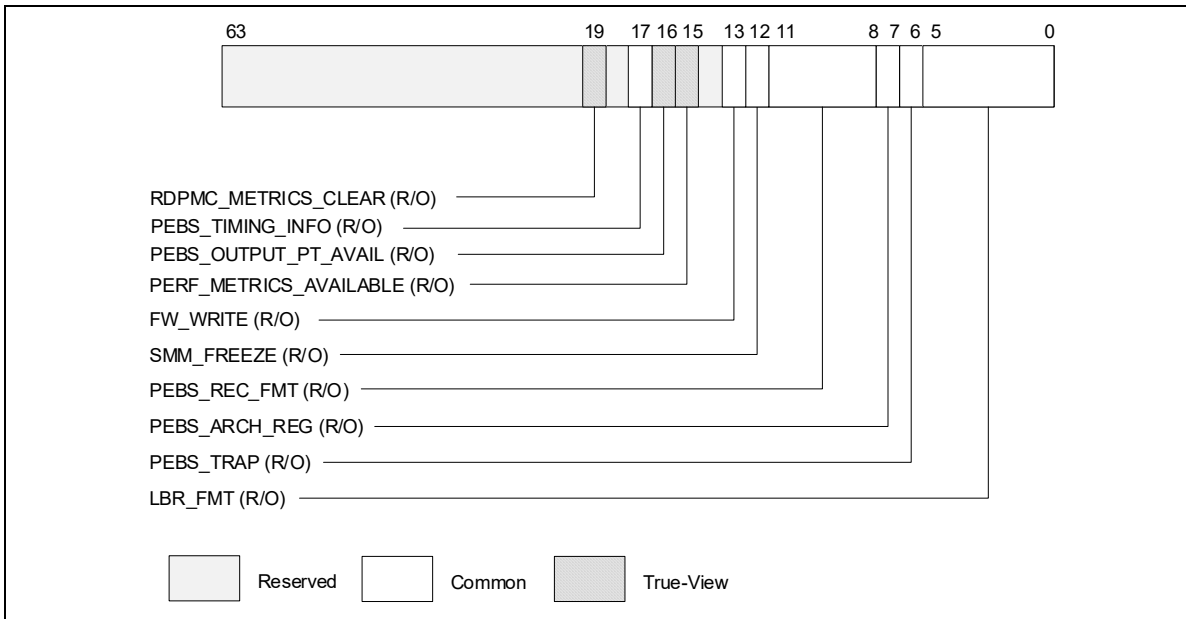


Figure 22-67. Layout of IA32_PERF_CAPABILITIES MSR

22.8.1 Filtering of SMM Handler Overhead

When performance monitoring facilities and/or branch profiling facilities (see Section 20.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)”) are enabled, these facilities capture event counts, branch records and branch trace messages occurring in a logical processor. The occurrence of interrupts, instruction streams due to various interrupt handlers all contribute to the results recorded by these facilities.

If CPUID.01H:ECX.PERF_CAPABILITIES[15] is 1, the processor supports the IA32_PERF_CAPABILITIES MSR. If IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is 1, the processor supports the ability for system software using performance monitoring and/or branch profiling facilities to filter out the effects of servicing system management interrupts.

If the FREEZE_WHILE_SMM capability is enabled on a logical processor and after an SMI is delivered, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler.

The enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its servicing.

It is the responsibility of the SMM code to ensure the state of the performance monitoring and branch profiling facilities are preserved upon entry or until prior to exiting the SMM. If any of this state is modified due to actions by the SMM code, the SMM code is required to restore such state to the values present at entry to the SMM handler.

System software is allowed to set IA32_DEBUGCTL.FREEZE_WHILE_SMM[bit 14] to 1 only supported as indicated by IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] reporting 1.

22.9 PEBS FACILITY

22.9.1 Extended PEBS

The Extended PEBS feature supports Processor Event Based Sampling (PEBS) on all counters, both fixed function and general purpose; and all performance monitoring events, both precise and non-precise. PEBS can be enabled for the general purpose counters using PEBS_EN_PMCi bits of IA32_PEBS_ENABLE (i = 0, 1, ...m). PEBS can be enabled for 'i' fixed function counters using the PEBS_EN_FIXEDi bits of IA32_PEBS_ENABLE (i = 0, 1, ...n).

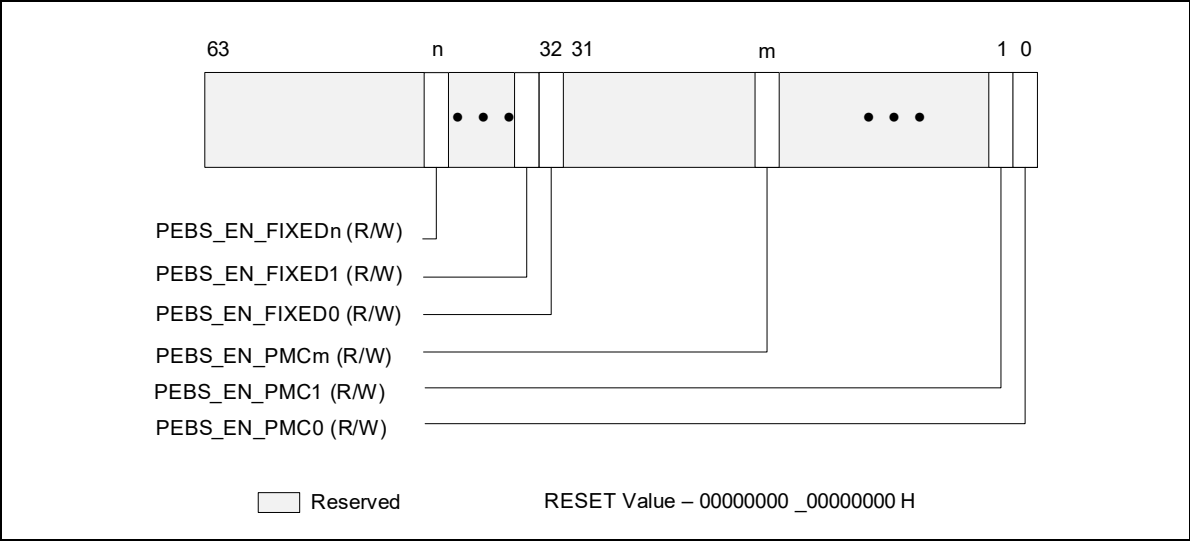


Figure 22-68. Layout of IA32_PEBS_ENABLE MSR

A PEBS record due to a precise event will be generated after an instruction that causes the event when the counter has already overflowed. A PEBS record due to a non-precise event will occur at the next opportunity after the counter has overflowed, including immediately after an overflow is set by an MSR write.

Currently, IA32_FIXED_CTR0 counts instructions retired and is a precise event. IA32_FIXED_CTR1, IA32_FIXED_CTR2 ... IA32_FIXED_CTRm count as non-precise events.

The Applicable Counter field in the Basic Info Group of the PEBS record indicates which counters caused the PEBS record to be generated. It is in the same format as the enable bits for each counter in IA32_PEBS_ENABLE. As an example, an Applicable Counter field with bits 2 and 32 set would indicate that both general purpose counter 2 and fixed function counter 0 generated the PEBS record.

To properly use PEBS for the additional counters, software will need to set up the counter reset values in PEBS portion of the DS_BUFFER_MANAGEMENT_AREA data structure that is indicated by the IA32_DS_AREA register. The layout of the DS_BUFFER_MANAGEMENT_AREA is shown in Figure 22-69. When a counter generates a PEBS records, the appropriate counter reset values will be loaded into that counter. In the above example where general purpose counter 2 and fixed function counter 0 generated the PEBS record, general purpose counter 2 would be reloaded with the value contained in PEBS GP Counter 2 Reset (offset 50H) and fixed function counter 0 would be reloaded with the value contained in PEBS Fixed Counter 0 Reset (offset 80H).

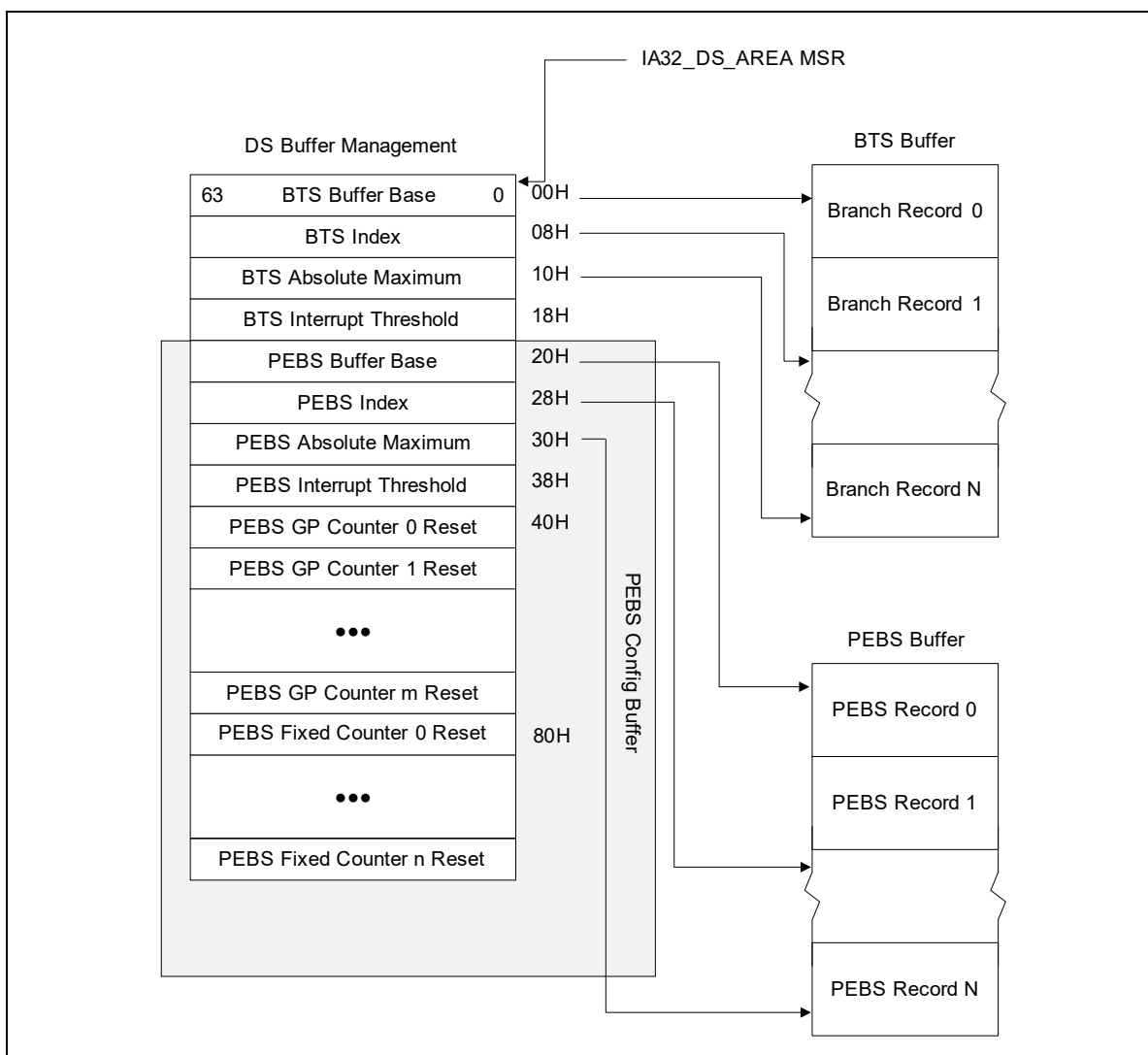


Figure 22-69. PEBS Programming Environment

Extended PEBS support debuts on Intel Atom® processors based on the Goldmont Plus microarchitecture and future Intel® Core™ processors based on the Ice Lake microarchitecture.

22.9.2 Adaptive PEBS

The PEBS facility has been enhanced to collect the following CPU state in addition to GPRs, EventingIP, TSC, and memory access related information collected by legacy PEBS:

- XMM registers
- LBR records (TO/FROM/INFO)
- Counters Snapshotting

The PEBS record is restructured where fields are grouped into Basic group, Memory group, GPR group, XMM group, LBR group, and Counters group. A new register MSR_PEBS_DATA_CFG provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.

By default, the PEBS record will only contain the Basic group. Optionally, each counter can be configured to generate a PEBS records with the groups specified in MSR_PEBS_DATA_CFG.

Details and examples for the Adaptive PEBS capability follow below.

22.9.2.1 Adaptive_Record Counter Control

IA32_PERFEVTSELx.Adaptive_Record[34]: If this bit is set and IA32_PEBS_ENABLE.PEBS_EN_PMCx is set for the corresponding GP counter, an overflow of PMCx results in generation of an adaptive PEBS record with state information based on the selections made in MSR_PEBS_DATA_CFG. If this bit is not set, a basic record is generated.

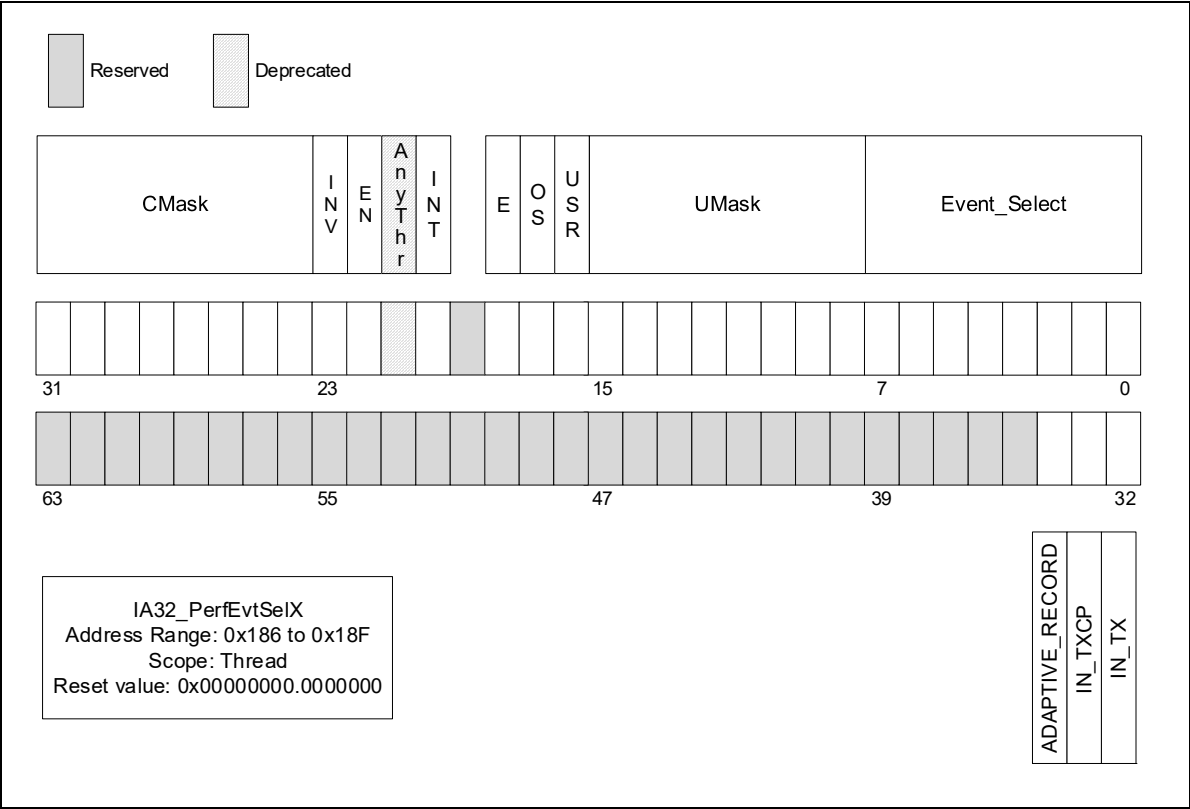


Figure 22-70. Layout of IA32_PerfEvtSelX MSR Supporting Adaptive PEBS

IA32_FIXED_CTR_CTRL.FCx_Adaptive_Record: If this bit is set and IA32_PEBS_ENABLE.PEBS_EN_FIXEDx is set for the corresponding Fixed counter, an overflow of FixedCtrx results in generation of an adaptive PEBS record with state information based on the selections made in MSR_PEBS_DATA_CFG. If this bit is not set, a basic record is generated.

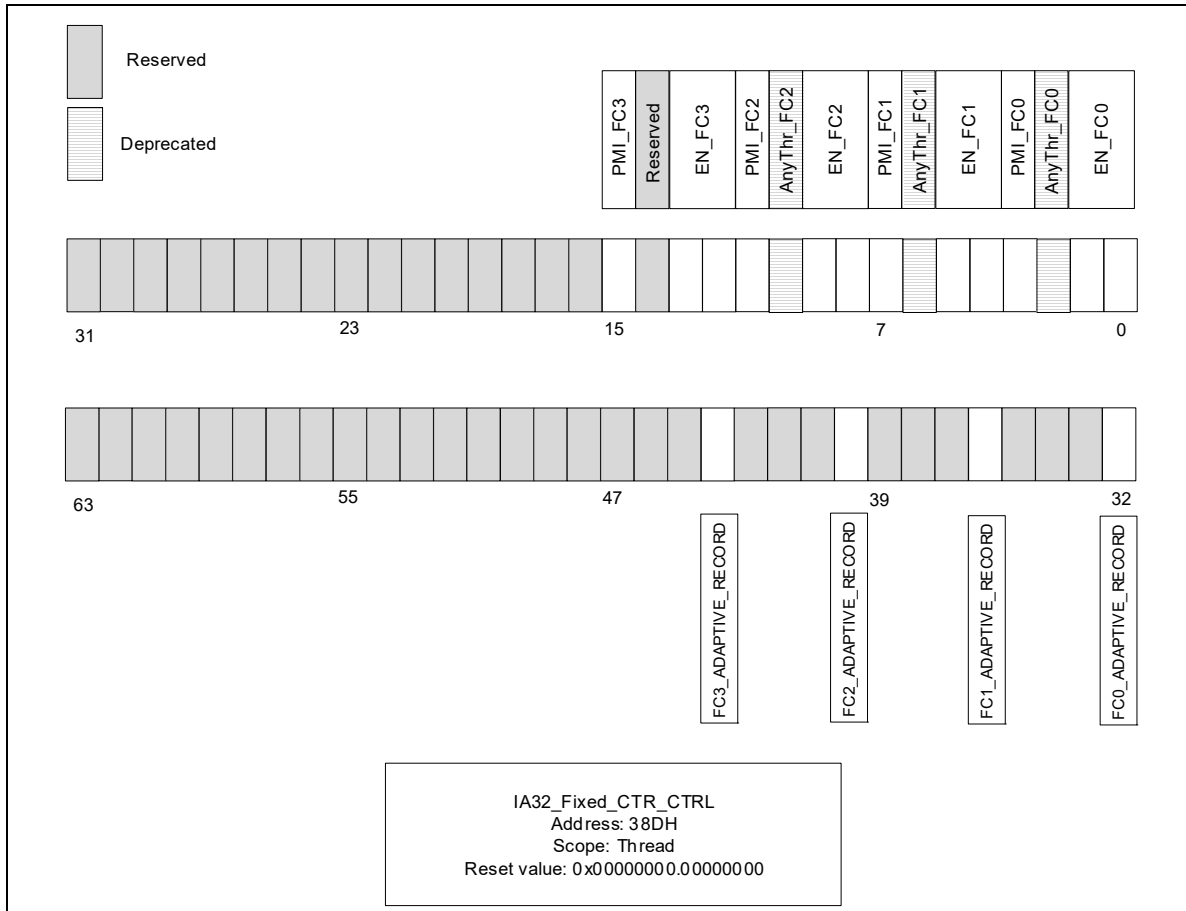


Figure 22-71. Layout of IA32_FIXED_CTR_CTRL MSR Supporting Adaptive PEBS

22.9.2.2 PEBS Record Format

The data fields in the PEBS record are aggregated into five groups which are described in the sub-sections below. Processors that support Adaptive PEBS implement a new MSR called MSR_PEBS_DATA_CFG which allows software to select the data groups to be captured. The data groups are not placed at fixed locations in the PEBS record, but are positioned immediately after one another, thus making the record format/size variable based on the groups selected.

22.9.2.2.1 Basic Info

The Basic group contains essential information for software to parse a record along with several critical fields. It is always collected.

Table 22-96. Basic Info Group

Field Name	Bit Width	Description
Record Format	[47:0]	This field indicates which data groups are included in the record. The field is zero if none of the counters that triggered the current PEBS record have their Adaptive_Record bit set. Otherwise it contains the value of MSR_PEBS_DATA_CFG.
	[63:48]	This field provides the size of the current record in bytes. Selected groups are packed back-to-back in the record without gaps or padding for unselected groups.

Table 22-96. Basic Info Group (Contd.)

Instruction Pointer	[63:0]	This field reports the Eventing Instruction Pointer (EventingIP) of the retired instruction that triggered the PEBS record generation. Note that this field is different than R/EIP which records the instruction pointer of the next instruction to be executed after record generation. The legacy R/EIP field has been removed.
Applicable Counters	[63:0]	The Applicable Counters field indicates which counters triggered the generation of the PEBS record, linking the record to specific events. This allows software to correlate the PEBS record entry properly with the instruction that caused the event, even when multiple counters are configured to generate PEBS records and multiple bits are set in the field.
TSC	[63:0]	This field provides the time stamp counter value when the PEBS record was generated.

22.9.2.2.2 Memory Access Info

This group contains the legacy PEBS memory-related fields; see Section 22.3.1.1.2.

Table 22-97. Memory Access Info Group

Field Name	Bit Width	Description
Memory Access Address	[63:0]	This field contains the linear address of the source of the load, or linear address of the destination (target) of the store. This value is written as a 64-bit address in canonical form.
Memory Auxiliary Info	[63:0]	When a MEM_TRANS_RETIRE.* event is configured in a General Purpose counter, this field contains an encoded value indicating the memory hierarchy source which satisfied the load. These encodings are detailed in Table 22-5 and Table 22-14. If the PEBS assist was triggered for a store uop, this field will contain information indicating the status of the store, as detailed in Table 22-15.
Memory Access Latency ¹	[63:0]	When a MEM_TRANS_RETIRE.* event is configured in a General Purpose counter, this field contains the latency to service the load in core clock cycles.
TSX Auxiliary Info	[31:0]	This field contains the number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
	[63:32]	This field contains the abort details. Refer to Section 22.3.6.5.1.

NOTES:

1. In certain conditions, high latencies in fields under “Memory Access Latency” may be observed even when the Data Src of the “Memory Auxiliary Info” field indicates a close source.

Beginning with 12th generation Intel Core processors, the memory access information group has been updated. New fields added are shaded gray in Table 22-98.

Table 22-98. Updated Memory Access Info Group

Field Name	Sub-field Name	Bits	Description
Access Address (offset 0H)	DLA	[63:0]	This field reports the data linear address (DLA) of the memory access in canonical form. A zero value indicates the processor could not retrieve the address of the particular access.
Access Info (offset 8H)	Data Src	[3:0]	An encoded value indicating the memory hierarchy source which satisfied the access. These encodings are detailed in Table 22-5. A zero value indicates the processor could not retrieve the data source of the particular access.
	STLB-miss	[4]	A value of 1 indicates the access has missed the Second-level TLB (STLB).
	Is-Lock	[5]	A value of 1 indicates the access was part of a locked (atomic) memory transaction.
	Data-Blk	[6]	A value of 1 indicates the load was blocked since its data could not be forwarded from a preceding store.
	Address-Blk	[7]	A value of 1 indicates the load was blocked due to potential address conflict with a preceding store.
Access Latency (offset 10H)	Instruction Latency	[15:0]	Measured instruction latency in core cycles. For loads, the latency starts by the dispatch of the load operation for execution and lasts until completion of the instruction it belongs to. This field includes the entire latency including time for data-dependency resolution or TLB lookups.
	Cache Latency	[47:32]	Measured cache access latency in core cycles. For loads, the latency starts by the actual cache access until the data is returned by the memory subsystem. For stores, the latency starts when the demand write accesses the L1 data-cache and lasts until the cacheline write is completed in the memory subsystem. This field does not include non-data-cache latency such as memory ordering checks or TLB lookups.
TSX (offset 18H)	Transaction Latency	[31:0]	This field contains the number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
	Abort Info	[63:32]	This field contains the abort details. Refer to Section 22.3.6.5.1.

To determine which fields are supported for certain performance monitoring events, consult the Memory Info attribute in the event lists at <https://github.com/intel/perfmon>.

NOTE

There may be additional block reasons, even if Data-Blk and Address-Blk are both clear, e.g., non-optimal instruction latency.

On P-core, the new Data-Blk and Address-Blk bits require the event LD_BLOCKS.STORE_FORWARD (r8203) to be configured in a programmable counter.

22.9.2.2.3 GPRs

This group is captured when the GPR bit is enabled in MSR_PEBS_DATA_CFG. GPRs are always 64 bits wide. If they are selected for non 64-bit mode, the upper 32-bit of the legacy RAX - RDI and all contents of R8-15 GPRs will be filled with 0s. In 64bit mode, the full 64 bit value of each register is written.

The order differs from legacy. The table below shows the order of the GPRs in Ice Lake microarchitecture.

Table 22-99. GPRs in Ice Lake Microarchitecture

Field Name	Bit Width
RFLAGS	[63:0]
RIP	[63:0]
RAX	[63:0]
RCX*	[63:0]
RDX*	[63:0]
RBX*	[63:0]
RSP*	[63:0]
RBP*	[63:0]
RSI*	[63:0]
RDI*	[63:0]
R8	[63:0]
...	...
R15	[63:0]

The machine state reported in the PEBS record is the committed machine state immediately after the instruction that triggers PEBS completes.

For instance, consider the following instruction sequence:

MOV eax, [eax]; triggers PEBS record generation

NOP

If the mov instruction triggers PEBS record generation, the EventingIP field in the PEBS record will report the address of the mov, and the value of EAX in the PEBS record will show the value read from memory, not the target address of the read operation. And the value of RIP will contain the linear address of the nop.

22.9.2.2.4 XMMs

This group is captured when the XMM bit is enabled in MSR_PEBS_DATA_CFG and SSE is enabled. If SSE is not enabled, the fields will contain zeroes. XMM8-XMM15 will also contain zeroes if not in 64-bit mode.

Table 22-100. XMMs

Field Name	Bit Width
XMM0	[127:0]
...	...
XMM15	[127:0]

22.9.2.2.5 LBRs

To capture LBR data in the PEBS record, the LBR bit in MSR_PEBS_DATA_CFG must be enabled. The number of LBR entries included in the record can be configured in the LBR_entries field of MSR_PEBS_DATA_CFG.

Table 22-101. LBRs

Field Name	Bit Width	Description
LBR[.].FROM	[63:0]	Branch from address.
LBR[.].TO	[63:0]	Branch to address.
LBR[.].INFO	[63:0]	Other LBR information, like timing. This field is described in more detail in Section 20.12.1, “MSR_LBR_INFO_x MSR.”

LBR entries are recorded into the record starting at LBR[TOS] and proceeding to LBR[TOS-1] and following. Note that LBR index is modulo the number of LBRs supporting on the processor.

22.9.2.3 MSR_PEBS_DATA_CFG

Bits in MSR_PEBS_DATA_CFG can be set to include data field blocks/groups into adaptive records. The Basic Info group is always included in the record. Additionally, the number of LBR entries included in the record is configurable.

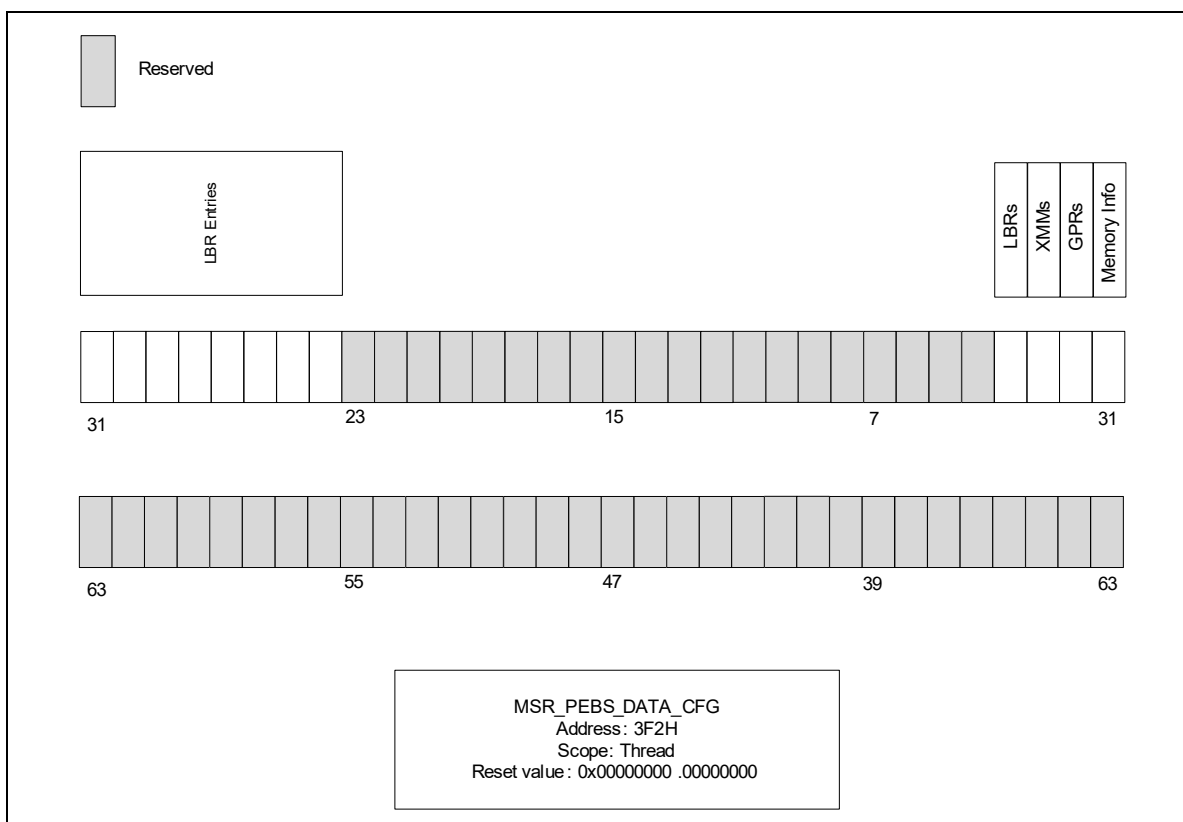


Figure 22-72. Legacy MSR_PEBS_DATA_CFG

Beginning with the Intel Series 2 Core Ultra processor, which counters are included in the Counters group is configurable. See Figure 22-73.

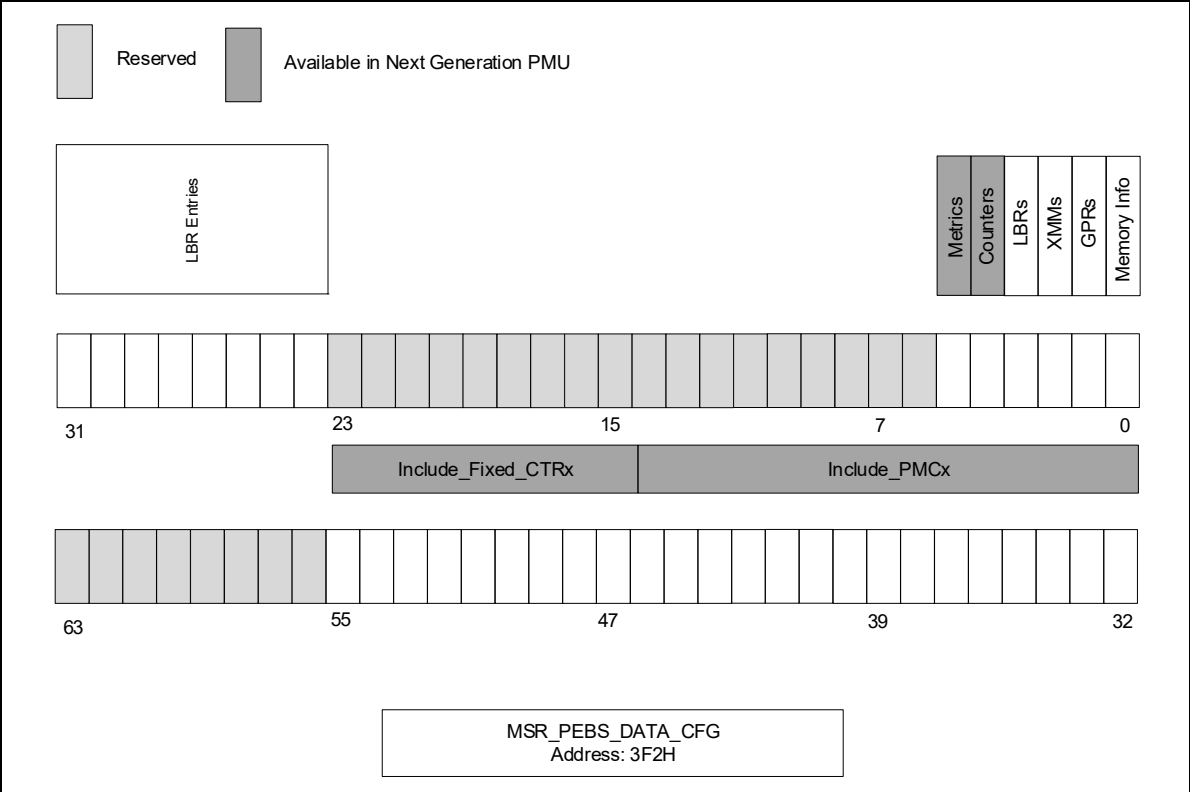


Figure 22-73. MSR_PEBS_DATA_CFG in PEBS_FMT=6

Table 22-102. MSR_PEBS_CFG Programming¹

Bit Name	Bit Index	Access	Description	Availability
Memory Info	0	R/W	Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.	PEBS_FMT=4 and later
GPRs	1	R/W	Setting this bit will capture the contents of the General Purpose registers in the PEBS record.	PEBS_FMT=4 and later
XMMs	2	R/W	Setting this bit will capture the contents of the XMM registers in the PEBS record.	PEBS_FMT=4 and later
LBRs	3	R/W	Setting this bit will capture LBR TO, FROM, and INFO in the PEBS record.	PEBS_FMT=4 and later
Counters	4	R/W	Setting this bit will allow recording of the IA32_PMCx MSRs and the IA32_FIXED_CTRx counters. The Include_PMCx and Include_Fixed_CTRx bits are also set.	PEBS_FMT=6 ²
Metrics	5	R/W	Setting this bit will allow recording and clearing of the MSR_PERF_METRICS register (when the Include_Fixed_CTR3 bit is also set).	PEBS_FMT=6 ² && PERF_METRICS_AVAILABLE =1
Reserved ³	23:6	NA	Reserved.	

Table 22-102. MSR_PEBS_CFG Programming¹ (Contd.)

LBR Entries	31:24	R/W	Set the field to the desired number of entries minus 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, it is recommended to select an even number of LBR entries (programmed into LBR_entries as an odd number).	PEBS_FMT=4 and later
Include_PMCx	47:32	R/W	A bit mask of the general-purpose counters that are allowed to be captured into the PEBS record. Note that only bits that match reporting of CPUID.23H.01H:EAX are writable.	PEBS_FMT=6 ²
Include_FIXED_CTRx	55:48	R/W	A bit mask of the fixed-function counters that are allowed to be captured into the PEBS record. Note that only bits that match reporting of CPUID.23H.01H:EBX are writable.	PEBS_FMT=6 ²
Reserved	63:56	NA	Reserved.	

NOTES:

1. A write to the MSR will be ignored when IA32_MISC_ENABLE.PERFMON_AVAILABLE is zero (default).
2. These fields are available starting with the IA32_PERF_CAPABILITIES.PEBS_FMT of 6 in addition to a subset of processors with a CPUID signature value of DisplayFamily_DisplayModel 06_C5H or 06_C6H (though they report IA32_PERF_CAPABILITIES.PEBS_FMT as 5).
3. Writing to the reserved bits will cause a GP fault.

22.9.2.3.1 Counters and Metrics Group

To capture the counters group, either the COUNTERS bit or the METRICS bit must be enabled in MSR_PEBS_DATA_CFG. The group allows recording of the IA32_PMCx MSRs, IA32_FIXED_CTRx MSRs, and the Performance Metrics.

The counters group first captures a 128-bit header with the bit vector of the counters that are captured later. The format of the counters header and the payload is shown in Table 22-103.

The group is available starting with IA32_PERF_CAPABILITIES.PEBS_FMT of 6. Additionally, the group is available in a subset of processors with a CPUID signature value of DisplayFamily_DisplayModel 06_C5H or 06_C6H (though they report IA32_PERF_CAPABILITIES.PEBS_FMT as 5).

Table 22-103. Counters Group

Field Name	Sub-Field Name	Bit Width	Description
Counters Group Header	PMC BitVector	[31:0]	Bit vector of IA32_PMCx MSRs. IA32_PMCx is recorded if bit x is set.
	FIXED_CTR BitVector	[31:0]	Bit vector of IA32_FIXED_CTRx MSRs. IA32_FIXED_CTRx is recorded if bit x is set.
	Metrics BitVector	[31:0]	Bit vector of the performance metrics counters.
	Reserved	[31:0]	Reserved.

Table 22-103. Counters Group (Contd.)

Field Name	Sub-Field Name	Bit Width	Description
Counters/Metrics Values	PMCx	[63:0]	PMCx will be captured if PMC BitVector x is set.
	...		
	FIXED_CTRx	[63:0]	FIXED_CTRx will be captured if FIXED_CTRx BitVector x is set.
	...		
	Metrics Base	[63:0]	The performance metrics base, mapped to IA32_FIXED_CTR3, if Metrics BitVector bit 0 is set.
	Metrics Data	[63:0]	MSR_PERF_METRICS, if Metrics BitVector bit 1 is set.

IA32_PMCx will be captured if both Counters and MSR_PEBS_DATA_CFG bit 32 + x are set. In this case, the PMC BitVector field bit x will be set too.

IA32_FIXED_CTRx will be captured if both Counters and MSR_PEBS_DATA_CFG bit 48 + x are set. In this case, the FIXED_CTR BitVector field bit x will be set too.

The performance metrics will be recorded if both Metrics and MSR_PEBS_DATA_CFG bit 51 (the bit used for IA32_FIXED_CTR3) are set. The Metrics record will have two 64-bit fields, MSR_PERF_METRICS and the PERF_METRICS_BASE that is derived from IA32_FIXED_CTR3. In this case, the Metrics BitVector will be 3. Note that MSR_PERF_METRICS and the IA32_FIXED_CTR3 MSR will be cleared after they are recorded.

Size of the group can be calculated in bytes by: 16 + popcount(BitVectors[127:0]) * 8.

22.9.2.4 PEBS Record Examples

The following example shows the layout of the PEBS record when all data groups are selected (all valid bits in MSR_PEBS_DATA_CFG are set) and maximum number of LBRs are selected. There are no gaps in the PEBS record when a subset of the groups are selected, thus keeping the layout compact. Implementations that do not support some features will have to pad zeroes in the corresponding fields.

Table 22-104. PEBS Record Example 1

Offset	Group Name	Field Name	Legacy Name (If Different)
0x0	Basic Info	Record Format	New
		Record Size	New
0x8		Instruction Pointer	EventingRIP
0x10		Applicable Counters	
0x18		TSC	
0x20	Memory Info	Memory Access Address	DLA
0x28		Memory Auxiliary Info	DATA_SRC
0x30		Memory Access Latency	Load Latency
0x38		TSX Auxiliary Info	HLE Information

Table 22-104. PEBS Record Example 1 (Contd.)

0x40	GPRs	RFLAGS	
0x48		RIP	
0x50		RAX	
...		...	
0x88		RDI	
0x90		R8	
...		...	
0xC8		R15	
0xD0	XMMs	XMM0	New
...		...	
0x1C0		XMM15	
0x1D0	LBRs	LBR[TOS].FROM	New
0x1D8		LBR[TOS].TO	
0x1E0		LBR[TOS].INFO	
...		...	
0x4B8		LBR[TOS + 1].FROM	
0x4C0		LBR[TOS + 1].TO	
0x4C8		LBR[TOS + 1].INFO	

The following example shows the layout of the PEBS record when Basic, GPR, and LBR group with 3 LBR entries are selected.

Table 22-105. PEBS Record Example 2

Offset	Group Name	Field Name	Legacy Name (If Different)
0x0	Basic Info	Record Format	New
		Record Size	New
0x8		Instruction Pointer	EventingRIP
0x10		Applicable Counters	
0x18		TSC	
0x20	GPRs	RFLAGS	
0x28		RIP	
0x30		RAX	
...		...	
0x68		RDI	
0x70		R8	
...		...	
0xA8		R15	
0xB0	LBRs	LBR[TOS].FROM	New
0xB8		LBR[TOS].TO	
0xC0		LBR[TOS].INFO	
...		...	
0xE0		LBR[TOS + 1].FROM	
0xE8		LBR[TOS + 1].TO	
0xF0		LBR[TOS + 1].INFO	

22.9.3 Precise Distribution of Instructions Retired (PDIR) Facility

Precise Distribution of Instructions Retired Facility is available via PEBS on some microarchitectures. Refer to Section 22.3.4.4.4. Counters that support PDIR also vary. See the processor specific sections for availability.

22.9.4 Reduced Skid PEBS

For precise events, upon triggering a PEBS assist, there will be a finite delay between the time the counter overflows and when the microcode starts to carry out its data collection obligations. The Reduced Skid mechanism mitigates the “skid” problem by providing an early indication of when the counter is about to overflow, allowing the machine to more precisely trap on the instruction that actually caused the counter overflow thus greatly reducing skid.

This mechanism is a superset of the PDIR mechanism available in the Sandy Bridge microarchitecture. See Section 22.3.4.4.4

In the Goldmont microarchitecture, the mechanism applies to all precise events including, INST_RETIRE, except for UOPS_RETIRE. However, the Reduced Skid mechanism is disabled for any counter when the INV, ANY, E, or CMASK fields are set.

With Reduced Skid PEBS, the skid is precisely one event occurrence. Hence if counting INST_RETIRE, PEBS will indicate the instruction that follows that which caused the counter to overflow.

For the Reduced Skid mechanism to operate correctly, the performance monitoring counters should not be reconfigured or modified when they are running with PEBS enabled. The counters need to be disabled (e.g., via IA32_PERF_GLOBAL_CTRL MSR) before changes to the configuration (e.g., what event is specified in IA32_PERFEVTSELx or whether PEBS is enabled for that counter via IA32_PEBS_ENABLE) or counter value (MSR write to IA32_PMCx and IA32_A_PMCx).

22.9.5 EPT-Friendly PEBS

The 3rd generation Intel Xeon Scalable Family of processors based on Ice Lake microarchitecture (and later processors) and the 12th generation Intel Core processor (and later processors) support VMX guest use of PEBS when the DS Area (including the PEBS Buffer and DS Management Area) is allocated from a paged pool of EPT pages. In such a configuration PEBS DS Area accesses may result in VM exits (e.g., EPT violations due to “lazy” EPT page-table entry propagation), and in such cases the PEBS record will not be lost but instead will “skid” to after the subsequent VM Entry back to the guest. For precise events the guest will observe that the record skid by one event occurrence, while for non-precise events the record will skid by one instruction.

22.9.6 PDist: Precise Distribution

PDist eliminates any skid or shadowing effects from PEBS. With PDist, the PEBS record will be generated precisely upon completion of the instruction or operation that causes the counter to overflow (there is no “wait for next occurrence” by default).

PDist is supported by selected counters, and is only supported when those counters are programmed to count select precise events¹. The legacy PEBS behavior applies to counters that do not support PDist, unless specified otherwise. PDist requires that the INV, ANY, E, EQ, and CMASK fields are cleared. Which counters support PDist, and which events are supported for PDist, is model-specific. Further, the counter reload value must not be less than 256 for PDist to operate.

For the PDist mechanism to operate correctly, the performance monitoring counters should not be reconfigured or modified when they are running with PEBS enabled. The counters need to be disabled (e.g., via IA32_PERF_GLOBAL_CTRL MSR) before changes to the configuration (e.g., what event is specified in IA32_PERFEVTSELx or IA32_FIXED_CTR_CTRL or whether PEBS is enabled for that counter via IA32_PEBS_ENABLE) or counter value (MSR write to IA32_PMCx and IA32_A_PMCx or IA32_FIXED_CTRx).

22.9.7 Load Latency Facility

The load latency facility provides software a means to characterize the latencies of memory load operations to different levels of cache/memory hierarchy. This facility requires a processor supporting the enhanced PEBS record format in the PEBS buffer.

Beginning with 12th generation Intel Core processors, the load latency facility supports all fields in Table 22-98, “Updated Memory Access Info Group,” in addition to the Memory Access Address field:

- The **Instruction Latency** field measures the load latency from the load's first dispatch until final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches and data dependencies).
- The **Cache Latency** field measures the subset of cache access latency in core cycles. It starts from the actual cache access until the data is returned by the memory subsystem. The latency is reported for retired demand load operations in core cycles (it does not account for memory ordering blocks).
- The **Data Source** field is an encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 22-106. “Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures)”. In the descriptions, local memory refers to system memory physically attached to a

1. To determine whether an event is precise or supports PDist, consult the relevant attribute in the event lists at <https://github.com/intel/perfmon>.

processor package, and remote memory refers to system memory or cache physically attached to another processor package (in a server product).

- Through the **Access Info** field, load latency features binary indications on certain blocks that the load operation may have encountered. Refer to STLB-miss, Is-Lock, Data-Blk and Address-Blk fields in Table 22-98.

NOTE

For loads triggered by software prefetch instructions, the cache related fields including Data Source and Cache Latency, report values as if the load was an L1 cache hit (the prefetch completes without waiting for data return, for performance reasons).

Table 22-106. Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures)

Encoding [3:0]	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request).
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. This request was merged into an outstanding cache miss to same cache-line address.
03H	L2 HIT. This request was satisfied by the L2 cache.
04H	L3 HIT. This request was satisfied by the L3 cache with no coherency actions performed (snooping).
05H	XCORE MISS. This request was satisfied by the L3 cache but involved a coherency check in some sibling core(s).
06H	XCORE HIT. This request was satisfied by the L3 cache but involved a coherency check that hit a non-modified copy in a sibling core.
07H	XCORE FWD. This request was satisfied by a sibling core where either a modified (cross-core HITM) or a non-modified (cross-core FWD) cache-line copy was found.
08H	Local Far Memory. This request has missed the L3 cache and was serviced by local far memory.
09H	Remote Far Memory. This request has missed the L3 cache and was serviced by remote far memory.
0AH	Local Near Memory. This request has missed the L3 cache and was serviced by local near memory.
0BH	Remote Near Memory. This request has missed the L3 cache and was serviced by remote near memory.
0CH	Remote FWD. This request has missed the L3 cache and a non-modified cache-line copy was forwarded from a remote cache.
0DH	Remote HITM. This request has missed the L3 cache and a modified cache-line was forwarded from a remote cache.
0EH	I/O. Request of input/output operation.
0FH	UC. The request was to uncacheable memory.

Table 22-107. Data Source Encoding for Memory Accesses (Lion Cove and Next Generation Microarchitectures)

Encoding [4:0]	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request).
01H or 02H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
03H	FB merge. L1 mishandling buffer.
05H	L2 HIT. This request was satisfied by the L2 cache.
06H	XQ merge. L2 mishandling buffer.
08H	L3 HIT. This request was satisfied by the L3 cache.
0CH	L3 Hit, x-core forward.
0DH	L3 Hit, x-core modified.
0FH	L3 Miss, x-core modified.

Table 22-107. Data Source Encoding for Memory Accesses (Lion Cove and Next Generation Microarchitectures)

Encoding [4:0]	Description
10H	L3 Miss, MSC Hit (memory-side cache).
11H	L3 Miss, memory.

To use this feature, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- One of the relevant IA32_PERFEVTSELx MSRs is programmed to specify the event unit MEM_TRANS_RETIRED.LOAD_LATENCY (IA32_PerfEvtSelX[15:0] = 1CDH). The corresponding counter, IA32_PMCx, will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in an MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with instruction latency greater than this value are eligible for counting and PEBS data reporting. The minimum value that may be programmed in this register is 1.
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register.

Refer to Section 22.3.4.4.2 for further implementation details of Load Latency.

22.9.8 Store Latency Facility

Store latency support is available on the 12th generation Intel Core processor. Store latency is a PEBS extension that provides a means to profile store memory accesses in the system. It complements the load latency facility.

Store latency leverages the PEBS facility where it can provide additional information about sampled stores. The additional information includes the data address, memory auxiliary information, and the cache latency of the store access. Normal stores (those preceded with a read-for-ownership) as well as streaming stores are supported by the store latency facility.

Memory store operations typically do not limit performance since they update the memory with no operation that directly depends on them. Thus, data out of this facility should be carefully used once stores are suspected as a performance limiter; for example, once the TMA node of Backend_Bound.Memory_Bound.Store_Bound is flagged¹.

To enable the store latency facility, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- Program the MEM_TRANS_RETIRED.STORE_SAMPLE event on general-purpose performance-monitoring counter 0 (IA32_PERFEVTSEL0[15:0] = 2CDH).
- Setup the PEBS buffer to hold at least two records, setting both 'PEBS Absolute Maximum' and 'PEBS Interrupt Threshold', should any other counter be used by PEBS (that is whenever IA32_PEBS_ENABLE[x] ≠ 0 for x ≠ 0).
- Set IA32_PEBS_ENABLE[0].

The store latency information is written into a PEBS record as shown in Table Table 22-49. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (Processors Based on Ice Lake Microarchitecture)".

The store latency relies on the PEBS facility, so the PEBS configuration must be completed first. Unlike load latency, there is no option to filter on a subset of stores that exceed a certain threshold.

1. For more details about the method, refer to Section B.1, "Top-Down Analysis Method" of the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

22.9.9 Timed Processor Event Based Sampling

Timed Processor Event Based Sampling (Timed PEBS) enables recording of time in every PEBS record. It extends all PEBS records with timing information in a new “Retire Latency” field that is placed in the Basic Info group of the PEBS record as shown in Table 22-108.

Table 22-108. PEBS Basic Info Group

Offset	Field Name	Bits
0x0	Record Format	[31:0]
	Retire Latency	[47:32]
	Record Size	[63:48]
0x08	Instruction Pointer	[63:0]
0x10	Applicable Counters	[63:0]
0x18	TSC	[63:0]

The Retire Latency field reports the number of Unhalted Core Cycles between the retirement of the current instruction (as indicated by the Instruction Pointer field of the PEBS record) and the retirement of the prior instruction. All ones are reported when the number exceeds 16 bits.

Processors that support this enhancement set a new bit: IA32_PERF_CAPABILITIES.PEBS_TIMING_INFO[bit 17].

NOTE

Timed PEBS is not supported when PEBS is programmed on fixed-function counter 0. The Retire Latency field of such record is undefined.

22.9.10 Counters Snapshotting

Counters Snapshotting extends Adaptive PEBS with the PEBS Counters and Metrics group. This extension enables software to capture general-purpose counters, fixed-function counters, and performance metrics in the PEBS record. For additional details, see Section 22.9.2.3.1, “Counters and Metrics Group.”

22.10 ARCHITECTURAL PEBS

Processor Event-Based Sampling (PEBS) is a non-destructive profiling facility that enables capturing the CPU state, e.g., the Instruction Pointer register, memory access address, etc., when the counter of a Performance Monitoring (PerfMon) event, such as a cache miss or a retired instruction, overflows. Architectural PEBS creates a permanent, enumerated, and stable event-based sampling architecture based on the positive evolution of legacy PEBS.

A counter can select, independently from other counters, any of the following state groups to be included on its samples:

- Basic group (Instruction Pointer, time-stamp)
- Memory Auxiliary group
- General-Purpose Registers group
- XSAVE-Enabled Registers (XER) group
- Last Branch Records (LBRs) group
- Performance-monitoring Counters group

Additionally, the counter reload value dictates the sampling rate of PEBS. The counter can be configured for either a precise or non-precise event.

Architectural PEBS expands on the usability and applicability of legacy PEBS versions. The Output Buffer, a memory area where the processor dumps the CPU state, is physically addressed and configured via MSRs, making it virtualization-friendly for system-wide profiling (e.g., for hypervisors).

Architectural PEBS configuration is per counter in support of multi-tenant environments. The PEBS record is restructured, too, for a consistent yet expandable format for tools and enriched with additional state information (e.g., the XER group). Architectural PEBS ensures a state is not exposed across protection boundaries. It also improves the performance over previous PEBS versions using PMI-based sampling. Note that any cores using architectural PEBS will not support legacy PEBS, as it will have been deprecated.

22.10.1 Configuration

If CPUID.23H,0H:EAX[5] is set to 1, the processor supports Architectural PEBS. In this case, sub-leaves 04H and 05H of CPUID Leaf 23H indicate details of Architectural PEBS capabilities (see Chapter 21 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). MSR enumeration of legacy PEBS is also modified in IA32_PERF_CAPABILITIES and IA32_MISC_ENABLE.

22.10.1.1 IA32_PERF_CAPABILITIES

Architectural PEBS requires changes to the IA32_PERF_CAPABILITIES MSR (address 345H):

- BASELINE[14] = 0.
- PEBS_TRAP[6], PEBS_ARCH_REG[7], and PEBS_TIMING_INFO[17] = 1.
- PEBS_RECORD_FORMAT[11:8] = 0xF (legacy PEBS is unavailable).
- PEBS_Output_PT_Available[16] = 0.

22.10.1.2 IA32_MISC_ENABLE

Architectural PEBS requires changes to the IA32_MISC_ENABLE MSR (address 1A0H). Specifically, the read-only field PEBS_UNAVAILABLE[12] is removed, but its value is kept as 1.

22.10.2 Behavior

The processor effectively stores the selected CPU state to memory when a PEBS event occurs, namely after a performance-monitoring counter overflows. The counter can be configured with a precise or non-precise performance-monitoring event. PEBS generates a record for precise events immediately after the instruction (as specified by the Instruction Pointer field of the Basic group) that experienced the event. For non-precise events, PEBS generates a record for some instruction when the counter overflows.

A PEBS record is always generated at an instruction boundary, reducing the need for expensive performance monitoring interrupt (PMI)-based sampling.

22.10.2.1 Counters

Section 22.2 describes general-purpose and fixed-function counters. Section 22.2.6 provides further details on the performance monitoring architecture. CPUID is used to enumerate the number and type of counters.

Reload values associated with individual counters are set in the lowest 32 bits of the extension registers, specifically IA32_PMC_GpN_CFG_C.RELOAD_VALUE[31:0] for general-purpose counters and IA32_PMC_Fxm_CFG_C.RELOAD_VALUE[31:0] for fixed-function counters, where n and m are the counter numbers.

Since the counters count up, the reload values are configured by storing the two's-complement negation first. For example, the value 1024 would be written as -1024 or 0xFFFFFC00 using 32 bits. On a reload, for a 48-bit Architectural performance-monitoring counter, the processor 1-extends this to 48 bits (0xFFFF_FFFFC00).

Furthermore, PEBS is enabled for a counter when the logical processor's CPL (Current Privilege Level) satisfies the counter configuration.

Architectural PEBS / Architectural PerfMon does not mandate a particular sequence for enabling counting. However, our clear recommendation is to enable IA32_PERF_GLOBAL_CTRL last / disable it first as it provides atomic starting/stopping of the counters.

To reiterate, the PEBS enable bits are distributed per counter, unlike the “global” register in model-specific PEBS.

22.10.2.2 Record Data

The CPU writes a PEBS Record for each counter that has overflowed at the instruction boundary. Each PEBS Record contains a header followed by a collection of record groups. The header contains information about the size of the PEBS record and a bit vector indicating the presence of selected record groups. Available record groups are the Basic group, the Memory Auxiliary group, the General-Purpose Registers group, the XSAVE-Enabled Registers group, the Last Branch Records group, and the Performance-Monitoring Counters group. The Basic group is present by default in PEBS records as it includes necessary data for processing a record. These groups are listed in Table 22-109 and described in detail in Section 22.10.4

Table 22-109. Architectural PEBS Configuration Locations

Register/Format	Bit Description								
	31	30	29	28:24	23:17	16:10	9:8	7:3	2:0
CPUID.23H.04H:EBX	0	AUX	GPR	0	XER	0	LBR	CNTR/A	0
	63	62	61	60:56	55:49	48:42	41:40	39:35	34:32
IA32_PMC_GPn_CFG_C IA32_PMC_FXm_CFG_C	PEBS_EN	AUX	GPR	MBZ	XER	MBZ	LBR	CNTR/A	MBZ
	63	62	61	60:56	55:49	48:42	41:40	39:35	
PEBS Record Format	BASIC	AUX	GPR	RSVD	XER	RSVD	LBR	CNTR/ RSVD	RSVD

22.10.2.3 Programming PEBS

This section lists the programming guidelines¹ that system software should follow in the specified order. Then, it illustrates the behavior through an example.

22.10.2.3.1 Programming Guidelines at Initialization Phase

1. Disable counting globally.
2. Setup the PEBS output buffer via the IA32_PEBS_BASE and IA32_PEBS_INDEX registers.
3. Configure counter(s).²
 - a. Configure a performance-monitoring event into a counter.³
 - 1) Select a performance-monitoring event via the IA32_PMC_GPn_CFG_A MSR for general-purpose counters. This step does not apply for fixed-function counters.
 - 2) Set CPL filters via the IA32_PMC_GPn_CFG_A MSR for general-purpose counters or via the IA32_FIXED_CTR_CTRL MSR for fixed-function counters.

1. The first and last steps in both lists may rely on IA32_PERF_GLOBAL_CTRL. Architectural PEBS and Architectural Performance-Monitoring do not mandate a particular sequence for enabling counting. However, the recommendation is to enable IA32_PERF_GLOBAL_CTRL last or disable it first, as it provides atomic starting and stopping of the counters and ensures the least disruption to profiling.

2. This step may be repeated for each counter that software wishes to generate a PEBS record, at initialization, or when another event is configured (e.g., event-counter multiplexing usage).

3. The counter registers are accessed in the IA32_PMC_GPn_CTR MSR for general-purpose counter n, or in the IA32_PMC_FXm_CTR MSR for fixed-function counter m.

- 3) Enable the counter locally.
- b. Configure PEBS for the same counter via the IA32_PMC_GPn/FXm_CFG_C MSR.¹
 - 1) Program a Reload value.
 - 2) Select the groups to be included in the PEBS records; see Table 22-109.
 - 3) Set the PEBS_EN bit.
- c. Initialize the counter register with its Reload value.
4. Re-enable counting globally.

22.10.2.3.2 Programming Guidelines at Runtime (e.g., Inside the PMI Interrupt Service Routine)

1. Disable counting globally.
2. Handle PEBS.
 - a. Drain data from the PEBS buffer; for example, copy-over data to another buffer. This is the most-common case for which PEBS-triggered PMI is invoked.
 - b. Update IA32_PEBS_INDEX[WR_OFFSET] to the start of the PEBS buffer, if needed.
 - c. Clear IA32_PERF_GLOBAL_STATUS[ARCH_PEBS] via the IA32_PERF_GLOBAL_STATUS_RESET MSR if the PEBS buffer Threshold is enabled and/or Clear IA32_PEBS_INDEX[Full].
3. Handle counter n overflow.
 - a. Set the counter register to its Reload value. Retrieve n, the counter index, via one of the following methods:
 - 1) The Applicable Counter field of the Basic group of most recent record(s) in step 2a above.
 - 2) Traversing the counter registers' values.
 - b. In some usages, software may need to save the current value of counter n, prior to reloading the counter value.
4. Re-enable counting globally.

For general-purpose counters, PEBS is enabled when IA32_PMC_GPn_CFG_C.PEBS_EN && IA32_PERF_GLOBAL_CTRL.EN[n] && IA32_PMC_GPn_CFG_A.CNTR_EN is true, where n is the counter index.

For fixed-function counters, PEBS is enabled when IA32_PMC_FXm_CFG_C.PEBS_EN && IA32_PERF_GLOBAL_CTRL.EN[32+m] && IA32_FIXED_CTR_CTRL.[USR|OS]m is true, where m is the counter index.

Furthermore, PEBS is enabled for a counter when the logical processor's Current Privilege Level (CPL) satisfies the counter configuration. The PEBS enable bits are distributed per counter, unlike the "global" register in legacy PEBS.

Example 22-3. Programming PEBS

To get the records, as shown in Table 22-74, software should use the following steps:

1. Disable counting globally by writing zero to the IA32_PERF_GLOBAL_CTRL MSR (address 38FH).
2. Set up IA32_PEBS_BASE and IA32_PEBS_INDEX registers.
3. For the general-purpose counter 1:
 - a. Configure the event BR_INST_RETIRED.ALL_BRANCHES, which counts retired branches.
 - b. Set the PEBS_EN[63] bit in the IA32_PMC_GP1_CFG_C MSR (address 1907H). This enables PEBS.
 - c. Put the 32-bit reload value into IA32_PMC_GP1_CFG_C.RELOAD_VALUE.
 - d. There is no need to set other bits for the optional groups since the Basic group is included by default in PEBS records.
4. For the fixed-function counter 1:

1. Details on the referenced IA32_* registers can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

- a. Locally enable it by setting the CPL bits IA32_FIXED_CTR_CTRL.[USR1|OS1]. This counter counts CPU_CLK_UNHALTED.THREAD – Unhalted Core Cycles.
 - b. Set the PEBS_EN[63] bit in the IA32_PMC_FX1_CFG_C MSR (address 1987H). This enables PEBS.
 - c. Put the 32-bit reload value into IA32_PMC_FX1_CFG_C.RELOAD_VALUE.
 - d. Set the AUX[62] bit and GPR[61] bit in the IA32_PMC_FX1_CFG_C MSR. This includes the AUX and GPR groups in the record.
5. Re-enable counting globally by setting bits 1 and 33 in the IA32_PERF_GLOBAL_CTRL MSR to start counting on both counters. See Chapter 2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for details of the MSR.
- When the general-purpose counter and the fixed-function counter overflow, the two records are written to memory.

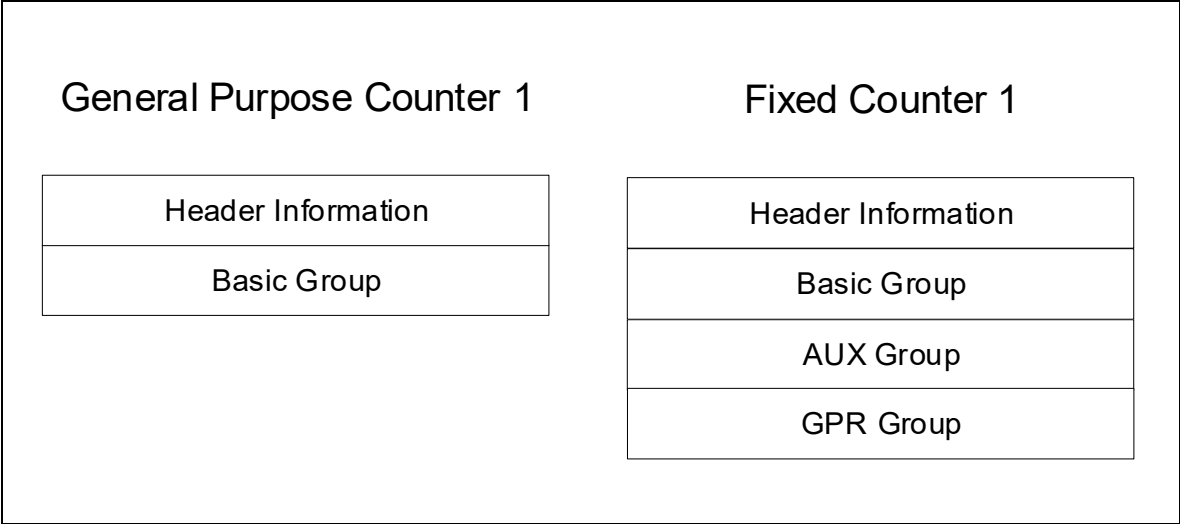


Figure 22-74. Example PEBS Record

22.10.3 Model-Specific Registers (MSRs)

The Architectural PEBS MSRs are IA32_PEBS_BASE, IA32_PEBS_INDEX, IA32_PMC_GPn_CFG_C, and IA32_PMC_FXm_CFG_C.

22.10.3.1 IA32_PEBS_BASE MSR

Figure 22-75 shows the layout of the IA32_PEBS_BASE MSR, the Output Buffer Base Register MSR.

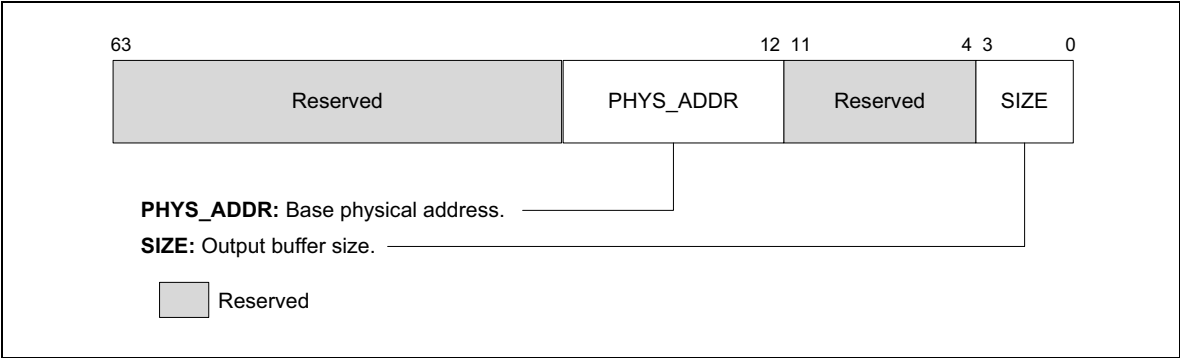


Figure 22-75. IA32_PEBS_BASE Register

The functions of the flags and fields are:

- **SIZE**, bits 3:0 – Specifies the output buffer size in powers of 2, where buffer size = 4 KB * 2^{SIZE}, e.g., SIZE = 7 results in a 512 KB output buffer and SIZE = 9 results in a 2 MB output buffer. The maximum SIZE value is 15, or 128 MB output buffer.
- **PHYS_ADDR**, bits (MAXPHYADDR – 1):12 – Specifies the value of the base physical address of a single, contiguous physical output region.

The MSR is initialized to all zeros on reset.

22.10.3.2 IA32_PEBS_INDEX MSR

Figure 22-76 shows the layout of the IA32_PEBS_INDEX MSR, the Output Buffer Index Register MSR.

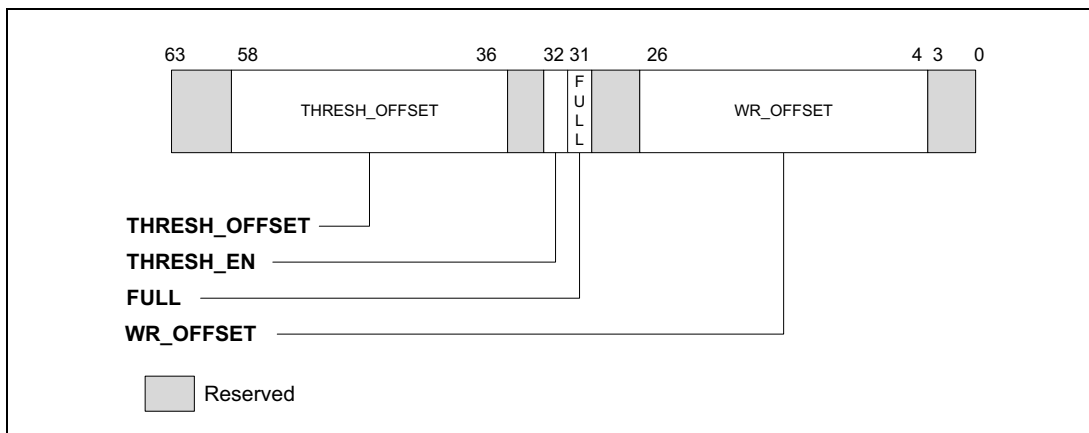


Figure 22-76. IA32_PEBS_INDEX Register

The functions of the flags and fields are:

- **WR_OFFSET**, bits 26:4 – Specifies the value of the next-byte-to-write offset from IA32_PEBS_BASE.PHYS_ADDR.
- **FULL**, bit 31 – When set, the output buffer is full, and no further records will be generated.
- **THRESH_EN**, bit 32 – When set, the PMI threshold is enabled.
- **THRESH_OFFSET**, bits 58:36 – Specifies the value of bits [26:4] of the PMI threshold offset from the output buffer base.

The MSR is initialized to all zeros on reset.

22.10.3.3 IA32_PMC_GPn_CFG_C and IA32_PMC_FXm_CFG_C MSRs

Table 22-110 shows the layout of the Performance Event Select Extended Register MSRs. The IA32_PMC_GPn_CFG_C MSR is the Event Select Extended register for a general-purpose counter, while the IA32_PMC_FXm_CFG_C MSR is the Event Select Extended register for a fixed-function counter, where n and m are the counter numbers.

Table 22-110. IA32_PMC_GPn_CFG_C and IA32_PMC_FXm_CFG_C MSRs

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1903H, 1907H, ..., 1903H+(4*n), 6403, 6407, ..., 6403 +(4*n)	IA32_PMC_GPn_CFG_C	
Extended Performance Event Selector for GP Counter x (R/W) For general-purpose counters, the IA32_PMC_GPn_CFG_C MSR is available if either CPUID.23H.05H:EAX[n] or CPUID.03H.02H:EAX[n] is set. Reserved bits must be zero.		
31:0	RELOAD_VALUE (R/W) Contains the reload value to be loaded into the associated counter by Auto Counter Reload. It will be 1-extended to 48 bits.	
34:32	Reserved.	
35	ALLOW_IN_RECORD (R/W) If 1, indicates that this counter's value will be included in the PEBS record if the associated CNTR subgroup is enabled.	
38:36	CNTR (R/W) When any bit is set, the counter group will be included in the PEBS record. Bit 36: If 1, the general-purpose counters subgroup will be included. Bit 37: If 1, the fixed-function counters subgroup will be included. Bit 38: If 1, the performance metrics subgroup will be included and will be reset	
39	Reserved.	
41:40	LBR (R/W) Identifies the number of Last Branch Records (LBRs) to record. The encodings of this field are as follows: 00 – LBR group will not be recorded. The group is disabled. 01 – 8 LBRs will be recorded. 10 – 16 LBRs will be recorded. 11 – The number of LBRs to be recorded is IA32_LBR_DEPTH.DEPTH.	
48:42	Reserved.	
55:49	XER (R/W) When set, enables particular XSAVE-Enabled Registers to be included in the PEBS record. The encodings for the register subgroup are: - Bit 49 – SSER: Record XMM0-XMM15. - Bit 50 – YMMHIR: Record the upper 128 bits of YMM0-YMM15. - Bit 51 – EGPR: Record R16-R31. - Bit 52 – Reserved and must be zero. - Bit 53 – OPMASKR: Record KO-K7. - Bit 54 – ZMMHIR: Record the upper 256 bits of ZMM0-ZMM15. - Bit 55 – Hi16ZMMR: Record ZMM16-ZMM31.	
60:56	Reserved.	
61	GPR (R/W) If 1, enables the General-Purpose Registers Group to be included in the PEBS record.	
62	AUX (R/W) If 1, enables the Auxiliary Group to be included in the PEBS record.	
63	PEBS (R/W) If 1, PEBS is enabled for this counter.	

Table 22-110. IA32_PMC_GPn_CFG_C and IA32_PMC_FXm_CFG_C MSRs (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1983H, 1987H, ..., 1983H+(4*m), 6531, 6535, ..., 6531 +(4*m)	IA32_PMC_FXm_CFG_C	
Extended Performance Event Selector for Fixed-Function Counter x (R/W) For fixed-function counters, the IA32_PMC_FXm_CFG_C MSR is available if either CPUID.23H.05H:ECX[m] or CPUID.23H.02H:EBX[m] is set. Reserved bits must be zero.		
63:0	This MSR has identical bit definitions to the IA32_PMC_GPn_CFG_C MSR.	

The functions of the flags and fields are:

- **RELOAD_VALUE**, bits 31:0 – Selects a 32-bit value loaded into IA32_PMC_GPn_CTR when a PEBS record is generated due to counter n. For a 48-bit counter, the value will be 1-extended into a 48-bit value.
- **ALLOW_IN_RECORD**, bit 35 – When set, indicates that this counter's value will be included in the PEBS record if the associated CNTR subgroup is enabled.
- **CNTR**, bits 38:36 – When any bit is set, the counter group will be included in the PEBS record. If bit 36 is set, the general-purpose counters subgroup will be included. If bit 37 is set, the fixed-function counters subgroup will be included. If bit 38 is set, the performance metrics subgroup will be included and reset.
- **LBR**, bits 41:40 – Identifies the number of Last Branch Records (LBRs) to record. The encodings of this field are as follows:
 - 00 – LBR group will not be recorded. The group is disabled.
 - 01 – 8 LBRs will be recorded.
 - 10 – 16 LBRs will be recorded.
 - 11 – The number of LBRs to be recorded is IA32_LBR_DEPTH.DEPTH.
- **XER**, bits 55:49 – When set, enables particular XSAVE-Enabled Registers to be included in the PEBS record. The encodings for the register subgroup are:
 - Bit 49 – SSER: Record XMM0-XMM15.
 - Bit 50 – YMMHIR: Record the upper 128 bits of YMM0-YMM15.
 - Bit 51 – EGPR: Record R16-R31.
 - Bit 53 – OPMASKR: Record K0-K7.
 - Bit 54 – ZMMHIR: Record the upper 256 bits of ZMM0-ZMM15.
 - Bit 55 – Hi16ZMMR: Record ZMM16-ZMM31.
- **GPR**, bit 61 – When set, this field enables the General-Purpose Registers group to be included in the PEBS record.
- **AUX**, bit 62 – When set, this field enables the Auxiliary group to be included in the PEBS record.
- **PEBS_EN**, bit 63 – When set, PEBS is enabled for this counter.

On reset, the MSR is initialized to all zeros.

22.10.3.4 Changes from Legacy PEBS

The following MSRs are removed and generate a #GP exception on any attempt to read or write them:

- IA32_PEBS_ENABLE
- MSR_PEBS_DATA_CFG

Architectural PEBS does not use the IA32_DS_AREA MSR, but it is maintained for BTS use.

Bits 32, 36, 40, and 44 of IA32_PMC_FXm_CFG.C.ADAPTIVE_RECORD_CTRm will report 0 on a RDMSR and will generate a #GP exception on an attempt to set them. The same results occur with IA32_PMC_GPn_CFG.Adaptive PEBS for all general-purpose counters. See Section 22.9.2.1 for more information.

The IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_STATUS_RESET, and IA32_PERF_GLOBAL_STATUS_SET MSRs are modified by adding a bit, ARCH_PEBS[54]:

- The processor clears IA32_PERF_GLOBAL_STATUS[54]=0 on a software write of 1 to IA32_PERF_GLOBAL_STATUS_RESET[54].
- The processor sets IA32_PERF_GLOBAL_STATUS[54]=1 on a software write of 1 to IA32_PERF_GLOBAL_STATUS_SET[54].

See Section 22.2.4.2 for more information.

22.10.4 Records and Groups

All records begin with a 16-byte header, shown in Figure 22-77, which includes the 8-byte PEBS record format. The record format contains the following:

- The bit vector indicating the groups in the record
- The record size, which may include end padding (see Section 22.10.4.7)
- Other indicator bits

The remaining 8 bytes in the header are reserved.

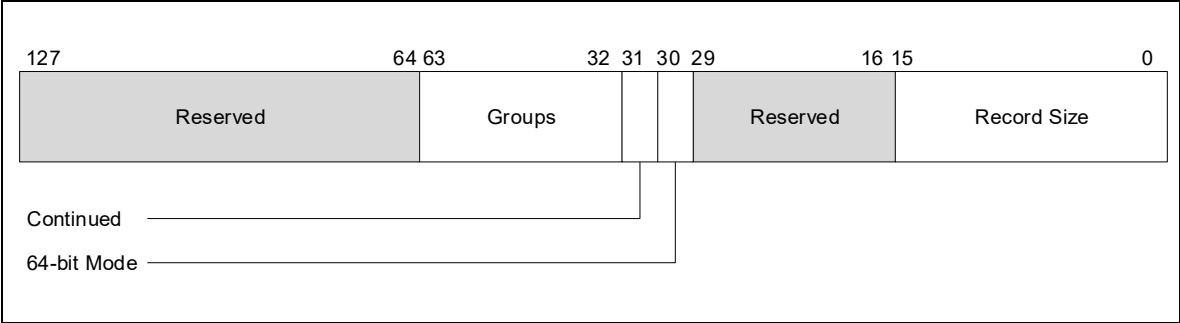


Figure 22-77. PEBS Record Format

The record size is a 16-bit field containing the size of the generated record (the header plus the groups) in bytes, with a maximum size of 64 kilobytes. A record always starts at a 16-byte aligned address. The size may include padding at the end for alignment reasons. The padding bytes would be stale and not written by the processor.

The Continued bit indicates that the record has additional groups in subsequent record fragments. See Section 22.10.4.7 for an example.

A record continues in the architecturally defined group order: Basic group, Memory Auxiliary group, General-Purpose Registers group, XSAVE-Enabled Registers group, Last Branch Records group, and finally, Performance-Monitoring Counters group. The record is also compacted to place only those groups indicated in the bit vector contiguously.

All groups are aligned along 16-byte boundaries and multiples of 16 bytes. Some groups have reserved fields and subfields to preserve alignment requirements. Most groups have a predetermined size, though some are variable.

A processor may dump groups/fields in any temporal order, not necessarily per their offset.

22.10.4.1 Basic Group

The Basic group is present in all complete PEBS records (see Section 22.10.4.7). The group size is 48 bytes and consists of the Instruction Pointer, Applicable Counter, Time-Stamp Counter, and Retire Latency fields. The final two fields in the group, each consisting of 8 bytes, are reserved for future growth.

Table 22-111 lists the Basic group's fields and the offset from the start of the Basic group. Each field is 64 bits wide.

Table 22-111. Basic Group Fields

Field Name	Byte Offset
Instruction Pointer	00H
Applicable Counter	08H
Time-Stamp Counter	10H
Retire Latency	18H
Reserved	20H
Reserved	28H

22.10.4.1.1 Instruction Pointer

The Instruction Pointer field holds the RIP register value¹ of the instruction that retired immediately before emitting the PEBS record. This field is sometimes historically referred to as “EventingIP.”

For precise events, this is the instruction that triggered PEBS record generation. For non-precise events, this is some instruction when the counter overflows without necessarily stopping at the causing instruction.

If a record was generated immediately after execution just became “in context” for a given counter, then the Instruction Pointer of that record is considered “out of context.” A -1 value denotes an out-of-context Instruction Pointer. This behavior avoids leaking information across contexts.

Note that the Instruction Pointer field differs from the RIP field in the General-Purpose Registers group, which records the instruction pointer of the next instruction to be executed after record generation. Also, when not in 64-bit mode, the upper 32 bits will be zero.

22.10.4.1.2 Applicable Counter

This field indicates which counter(s) triggered the generation of the PEBS record.

Figure 22-78 shows the mapping of the various one-hot counter bits in the PEBS record. IA32_PMC[i-1:0] is the bit vector of general purpose counters, where IA32_PMC0 starts at bit 0, and i is the maximum general purpose counter ID supported. IA32_FIXED_CTR[32+j-1:32] is the bit vector of fixed counters, where IA32_FIXED_CTR0 starts at bit 32, and j is the maximum fixed counter ID supported. Bits that do not correspond to counters identified in Section 22.2.9 are reserved.

1. RIP register without the CS register base.

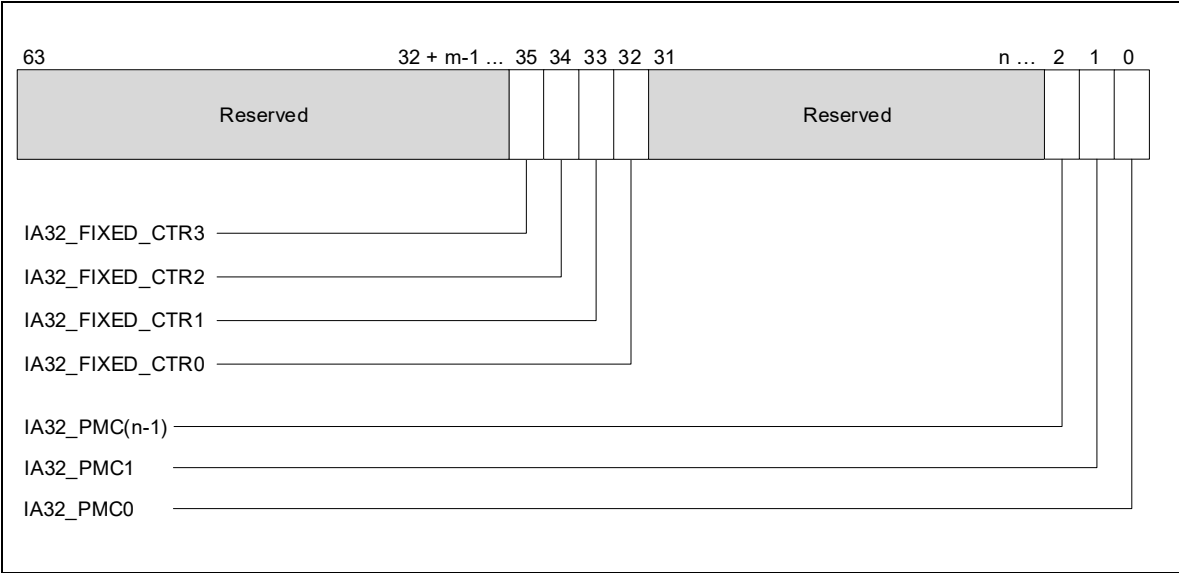


Figure 22-78. Bit Mapping of Applicable Counters in the PEBS Record

22.10.4.1.3 Time-Stamp Counter

The time-stamp counter is a 64-bit counter set to 0 following a processor reset. After a reset, the counter increments even when the HLT instruction or the external STPCLK# pin halts the processor.

The time-stamp counter will reflect the value of the counter when the record is generated, and it will include the offset and scale adjustments even if RDTSC Exiting is enabled (i.e., no exit taken). It is the same as Intel Processor Trace.

For more details, see Section 20.17, “Time-Stamp Counter,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B.

22.10.4.1.4 Retire Latency

The Retire Latency field in the record will consist of:

- **Retire Latency field**, bits 15:0 – Indicates the elapsed cycles between the retirement of the architecturally-visible instruction that caused PEBS and the prior instruction retirement. The measurement reflects core unhalted cycles (at the pace of Fixed-Function Counter 1) and is reported for any PEBS, regardless of whether precise or non-precise events are programmed. The count saturates at 216-1.
- **Retire Latency Valid flag**, bit 16 – A valid bit for the Retire Latency field. When set, the Retire Latency field holds a valid value.
- Reserved field, bits 63:17 – Reserved for future use.

22.10.4.2 Auxiliary Group

The Auxiliary (AUX) group is home to custom fields, often without ties to the state of a particular processor architectural feature and without architectural registers. The AUX group size is 64 bytes, consisting of 8 fields of 8 bytes each. The first 8-byte field is the Memory Access Address, followed by seven 8-byte reserved fields, as shown in Table 22-112. A named reserved field may have processor-dependent implementation.

Table 22-112. Auxiliary Group Fields

Field Name	Byte Offset
Memory Access Address	00H
Reserved	08H
Reserved	10H
Reserved	18H
Reserved	20H
Reserved (Memory Auxiliary Information)	28H
Reserved (Memory Access Latency)	30H
Reserved (TSX Auxiliary Information)	38H

If CPUID.23H.04H:EBX.AUX[30] = 1, then the Auxiliary group is available. If AUX[62] in the counter's Event Select Extended register is set, then the Auxiliary group will be included when that particular counter generates a PEBS record.

If PEBS Record Format.AUX[62] = 1 in the record itself, then the Auxiliary group is included.

All fields are populated in a model- and event-specific manner. None of the architectural events are architecturally specified to populate these fields, including the Memory Access Address field.

If the Memory Access Address population is not indicated in the model-specific event list, the field contents are reserved.

22.10.4.2.1 Memory Access Address

The Memory Access Address (MAA) is the 64-bit canonical linear address¹ of memory accessed by a load or store access of an instruction. The location of the MAA field is architectural, but which events update it and the address saved is model-specific. Unlike legacy PEBS, Architectural PEBS enables one instruction to generate multiple records with different MAA fields. For example, MOVDIR* instructions have distinct load and store addresses.

The MAA will be the address of a linear access performed by the instruction immediately preceding the PEBS event (at EventingIP, or macrofused² to it), or zero. The MAA will never expose access from any of these cases:

- A filtered out CPL, even for ring transitions
- Before any counters are enabled, e.g., a VMM address after a VM entry enables counters
- Inside a production/opt-out enclave (no PEBS records are taken during a production enclave, and PEBS records taken after EEXIT/AEX will have MAA=0)

For instructions with multiple linear load accesses, or multiple linear store accesses, which one (if any) is recorded is model-specific. Consult the online event list for more details on the MAA behavior for a particular processor: <https://perfmon-events.intel.com/>.

NOTE

For MSR flows (VMX MSR list, WRMSRLIST) that can disable and re-enable counters, it is possible for an MAA recorded before the disable to persist. Intel advises against such uses.

22.10.4.3 General-Purpose Register Group

The General-Purpose Register group contains 20 fields, each being 64 bits wide, for a total of 160 bytes. The group contains RFLAGS, RIP, RAX, RCX, etc., as well as the Shadow Stack Pointer (SSP). The SSP will be zero on implementations that do not support CET or when CET is not in use.

1. No Linear Address Masking (LAM) bits are included.
2. Macrofusion merges two instructions to a single micro-op. See Section 3.4.2.2, "Optimizing for Macrofusion," in the Intel® 64 and IA-32 Architectures Optimization Reference Manual: Volume 1.

If CPUID.23H.04H:EBX.GPR[29] = 1, then the General-Purpose Register group is available. If IA32_PMC_GPn_CFG_C.GPR[61]=1 or IA32_PMC_FXm_CFG_C.GPR[61] = 1, then the record will include the General-Purpose Register group when the counter generates a PEBS record. In the PEBS record, if PEBS Record Format.GPR[61] = 1, the General-Purpose Register group is included.

Table 22-113. General-Purpose Register Group Fields

Field Name	Byte Offset
RFLAGS	00H
RIP	08H
RAX	10H
RCX	18H
RDX	20H
RBX	28H
RSP	30H
RBP	38H
RSI	40H
RDI	48H
R8	50H
...	...
R15	88H
SSP	90H
Reserved	98H

The fields are sized to support 64-bit mode. Modes are checked at the time of PEBS record generation.

When not in 64-bit mode, the upper 32 bits of the RAX through RDI fields will be unmodified. Furthermore, fields for registers that “do not exist” are left unmodified, similar to how XSAVE handles vector registers. Space is not compacted away. Registers R8-R15 will be unmodified. This is consistent with model-specific PEBS behavior.

When not in 64-bit mode, fields for registers that do not exist are left unmodified. Core legacy model-specific PEBS wrote zero.

PEBS enable is not impacted by transitions into/out of 64-bit mode. Some legacy PEBS versions used to disable it. The indicator bit in PEBS Record Format.64BIT_MODE[30] will be set to 1 for 64-bit mode and 0 for other modes.

22.10.4.4 XSAVE-Enabled Registers Group

The XSAVE-Enabled Registers (XER) group is a super-group for user-level state with XSAVE architectural support, i.e., those supported by the XSAVEC instruction. Five groups (or sub-groups) are available initially (see Table 22-114). Additional groups are likely to be supported in future generations. The supported groups depend on the ISA support of the processor for SSER, YMMHIR, EGPR, OPMASKR, ZMMHIR, and Hi16ZMMR.

CPUID.23H.04H:EBX.XER[23:17] indicate which components of the XER super-group are available. The bits IA32_PMC_GPn_CFG_C.XER[55:49] and IA32_PMC_FXm_CFG_C.XER[55:49] select which groups of XER to include when the counter generates a PEBS record. The select bit in the *CFG_C MSRs is writeable if that group is available in CPUID.

The bits PEBS Record Format.XER[55:49] indicate which components of the XER group are included in the PEBS record. This field is set to 0 if CR0.TS = 1 or CR4.OSXSAVE = 0. Otherwise, a bit is set only if a group is selected in the CFG MSR and the XCR0 bit corresponding to the group is set. Group sizes are listed in Table 22-114.

Table 22-114. XSAVE-Enabled Registers (XER) Group Fields and Sizes

Field Name	Registers Used	Size
XSTATE_BV	XINUSE for groups included in PEBS Record Format.XER[55:49]	8 B
Reserved	Reserved	8 B
SSER	XMM0-XMM15	16 regs * 16 B = 256 B
YMMHIR	Upper 128 bits of YMM0-YMM15	16 regs * 16 B = 256 B
EGPR	R16-R31	16 regs * 8 B = 128 B
OPMASKR	K0-K7	8 regs * 8 B = 64 B
ZMMHIR	Upper 256 bits of ZMM0-ZMM15	16 regs * 32 B = 512 B
Hi16ZMMR	ZMM16-ZMM31	16 regs * 64 B = 1024 B

The first eight bytes include the XSAVES instruction's XSTATE_BV bit vector (reflecting INIT optimization). This field is in XCR0 format.

Memory space in the output buffer is allocated for the groups that are included in PEBS Record Format.XER[55:49]. The memory space is allocated but not written if that group is either not in use (that is, in the INIT state) or is disabled by XFD (eXtended Feature Disable). (In this case the corresponding bit in XSTATE_BV bitmap is set to 0.) The memory space is allocated and written when all conditions are met (that is, the group is selected, enabled by XSAVE, and is in use).

Outside 64-bit mode, vector registers beyond XMM7/YMM7/ZMM7 are not accessible. Thus, the memory space is allocated but not written for those registers.

Processors that do not support a particular group, e.g., ZMMHIR, will have the corresponding bit clear in CPUID, and attempting to set the select bit will cause a #GP fault.

22.10.4.5 Last Branch Record Group

Last Branch Records (LBRs) enable the recording of the software path history by logging taken branches and other control flow transfers within processor registers. See Section 20.5, "Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for information about last branch records.

Table 22-115 lists the LBR fields.

Architectural PEBS can include a subset of the architectural LBRs available. Software may, independent to PEBS, reduce the number of active LBR records from the maximum using the IA32_LBR_DEPTH MSR.

The bits IA32_PMC_GpN_CFG_C.LBR[41:40] and IA32_PMC_Fxm_CFG_C.LBR[41:40] indicate the number of LBR records to include in the Last Branch Record group. The encodings of this field are as follows:

- 00 – LBR group will not be recorded. The group is disabled.
- 01 – 8 LBRs will be recorded.
- 10 – 16 LBRs will be recorded.
- 11 – Variable size. The number of LBRs to be recorded is defined by IA32_LBR_DEPTH.DEPTH.

The bits PEBS Record Format.LBR[41:40] indicate the number of LBR records in the group.

Table 22-115. Last Branch Record Group Fields

Field Name	Byte Offset
Reserved	00H
IA32_LBR_CTL	08H
IA32_LBR_DEPTH	10H
IA32_LER_FROM_IP	18H

Table 22-115. Last Branch Record Group Fields

Field Name	Byte Offset
IA32_LER_TO_IP	20H
IA32_LER_INFO	28H
IA32_LBR_0_FROM_IP	30H
IA32_LBR_0_TO_IP	38H
IA32_LBR_0_INFO	40H
...	...
IA32_LBR_[Num_LBR-1]_FROM_IP	30H+[Num_LBR-1]*24
IA32_LBR_[Num_LBR-1]_TO_IP	38H+[Num_LBR-1]*24
IA32_LBR_[Num_LBR-1]_INFO	40H+[Num_LBR-1]*24

The group size is 6*8 bytes + Num_LBR*24 bytes. Software must look at IA32_LBR_DEPTH recorded in the group. If a fixed size was requested in the IA32_PMC_GPn_CFG_C or IA32_PMC_FXm_CFG_C register, IA32_LBR_DEPTH recorded in the group will indicate how many LBR entries in the group are valid (meaningful). If a variable size was requested in the IA32_PMC_GPn_CFG_C or IA32_PMC_FXm_CFG_C register, IA32_LBR_DEPTH recorded in the group will indicate the size of the variable portion of the LBR group.

The LBR array is preceded in the record by five additional LBR- and LER-related fields. Should a fixed number of Num_LBRs be requested, but LBR_DEPTH is set lower, entries beyond LBR_DEPTH will be unmodified. IA32_LBR_DEPTH is included in the LBR group in a specific location.

It is possible for LBRs to be in system wide mode (IA32_LBR_CTL[OS=1, USR=1]) while PEBS is in user mode. PEBS will not attempt to filter out any non-user data from the LBRs, including LERs. It is up to the software to ensure that if it allows LBRs in PEBS, that the filtering conditions are appropriate.

The size of a single LBR entry is 3 fields * 64 bits, or 24 bytes. The contents of each field is determined by the architectural LBR feature.

22.10.4.6 Performance Counters Group

Performance Counters enable performance monitoring events to be counted during processor execution. The Counters group (CNTR) enables users to snapshot the current values of select performance monitoring counters into the PEBS record, including counters other than the one that triggered the generation of the PEBS record. General purpose (GP), fixed function (FIXED), and performance metrics (METRICS) are covered by the Counters group. Table 22-116 provides the Counter group details.

The bits CPUID.23H.04H:EBX.CNTR[7:4] indicate that the Counter group component subgroups are available.

IA32_PMC_GPn_CFG_C.CNTR[39:36] and IA32_PMC_FXm_CFG_C.CNTR[39:36] indicate whether to include the Counter group subgroups when this counter generates a PEBS record. Any set bit will cause the Counter group header to be included in the record.

The bits PEBS Record Format.CNTR[39:36] indicate which counter group subgroups were included in the record. If any of these bits are set, the PEBS record will include the 16-byte group header. The group header contains four 32-bit vectors, one vector per subgroup, indicating which counters are included in each subgroup. Individual counters must opt in.

CPUID.23H.04H:EBX.ALLOW_IN_RECORD[3] indicates the IA32_PMC_GPn_CFG_C.ALLOW_IN_RECORD and IA32_PMC_FXm_CFG_C.ALLOW_IN_RECORD bits are available. Software should set IA32_PMC_GPn_CFG_C.ALLOW_IN_RECORD[35] and IA32_PMC_FXm_CFG_C.ALLOW_IN_RECORD[35] to include the counter's value in the PEBS record when a record is generated that includes the associated subgroup.

The ALLOW_IN_RECORD, METRICS, FIXED, and GP bits within the extension registers allow software to control which counters actually appear in the PEBS record.

Note that the bit PEBS Record Format.ALLOW_IN_RECORD[35] is reserved.

Table 22-116. Counter Group Details

Bit	Reserved	METRICS Subgroup	FIXED Subgroup	GP Subgroup	ALLOW_IN_RECORD
CPUID.23H.04H:EBX	7	6	5	4	3
IA32_PMC_GPn_CFG_C	39	38	37	36	35
IA32_PMC_FXm_CFG_C					
PEBS Record Format	39	38	37	36	35
Group Header Bits	127:96	95:94	63:32	31:0	N/A

Consider the following example using general-purpose counters, where:

- IA32_PMC_GP0_CFG_C.{ALLOW=1, GP=1}
- IA32_PMC_GP1_CFG_C.ALLOW=1
- IA32_PMC_GP2_CFG_C.ALLOW=0

If general-purpose counter 0 generates a PEBS record, it includes general-purpose counter 0 and general-purpose counter 1 (but not general-purpose counter 2 because its ALLOW bit was clear).

If general-purpose counter 1 or general-purpose counter 2 generates a PEBS record, it does not include any counters because all METRICS, FIXED, and GP bits were clear for general-purpose counter 1 and general-purpose counter 2.

22.10.4.7 Fragmented Records

A PEBS record can be split into *n* fragments that comprise the complete record. Fragmented records can help mitigate the contiguous buffer when PEBS2GPA is used. See Section 22.10.5.1 for information about PEBS2GPA and Chapter 31, “VMX Support for Address Translation” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 3C, for EPT details.

Each fragment of the complete record has a header and a subset of the groups. The first *n*-1 fragments will have their Continued[31] bit set to 1 to indicate that the record has additional groups in subsequent record fragments. The last fragment will have its Continued bit clear.

If there is empty space between record fragments, then either the Record Size field may indicate padding (i.e., having a value larger than the required size as described in Section 22.10.4) or a NULL record will be inserted. A NULL record is an empty record where only the Record Size field is non-zero (group indicators, the Continued bit, and other fields are clear).

For example, suppose that a complete record consists of two fragments, record 4 and record 5, which cross an EPT page boundary as shown in Figure 22-79. In the top portion of Figure 22-79, record 4 includes some of the requested groups and padding to abut the end of the page, and it has its Continued bit set to 1. In the lower part of Figure 22-79, record 4 includes some of the requested groups with no padding (it has its Continued bit set to 1) and the Null Record is at the end of the page. Record 5 contains the remaining groups in both instances, with the Continued bit clear.

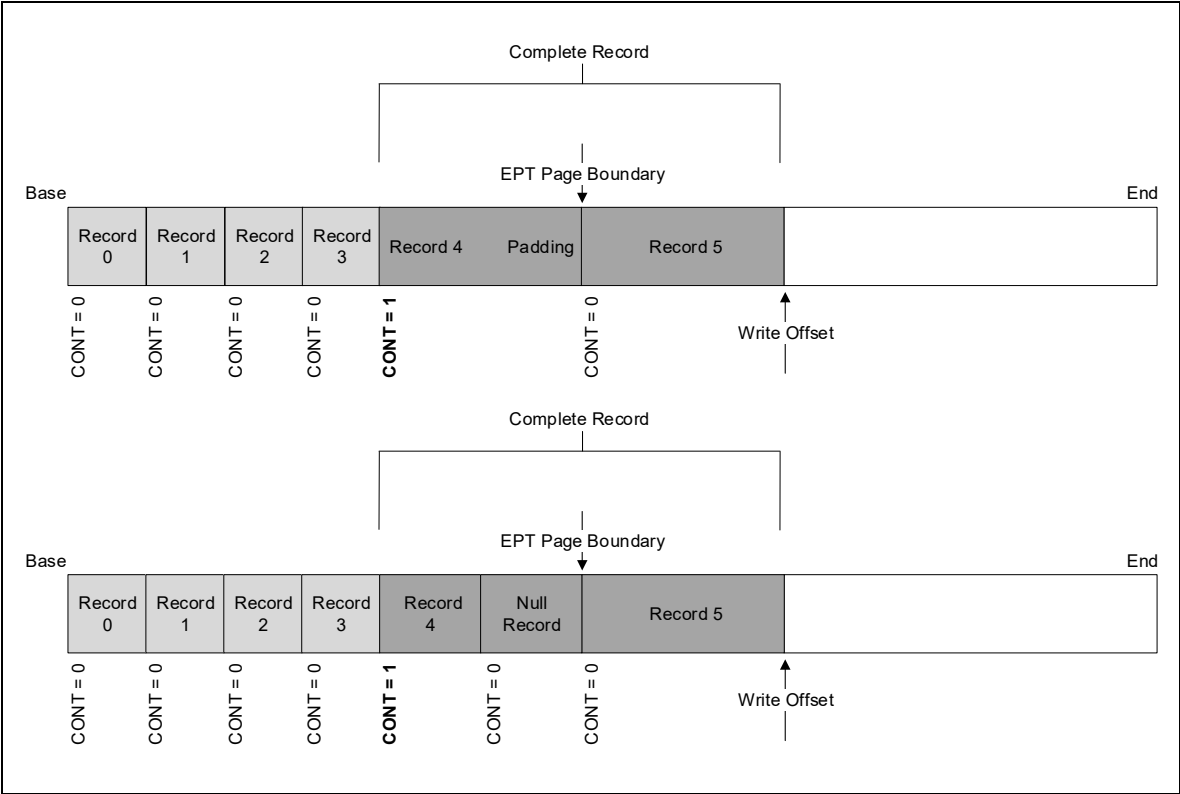


Figure 22-79. Example of a Fragmented Record

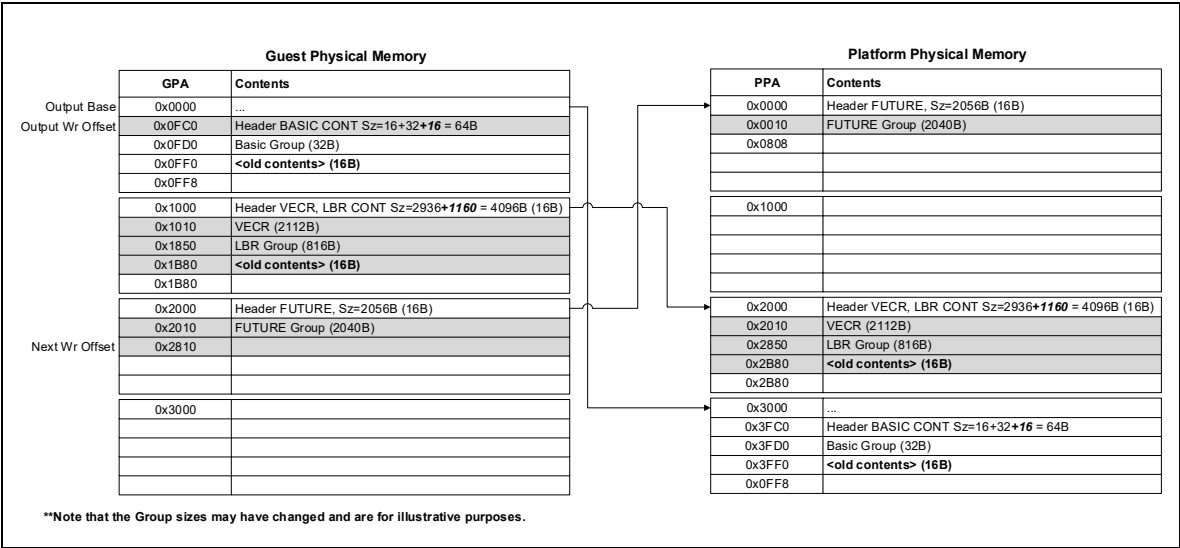


Figure 22-80. EPT Page Boundary Example with Included Padding

22.10.5 Interactions with Other Processor Features

22.10.5.1 Virtualization

In the context of virtualization, PEBS has dedicated support for either of two usages: system-wide use and guest-only use.

With system-wide use, PEBS operation can persist across VMX transitions. A VMM would allocate the PEBS buffer using a physical base address, and PEBS would use the resulting addresses in both VMX root operation and VMX non-root operation. Counters configured for PEBS would always be enabled (by suitable management of the IA32_PERF_GLOBAL_CTRL MSR), possibly hidden from guests. As a result, the relevant counters should continue counting across VMX transitions. This ensures that the counters and the PEBS record information are system-wide.

In this use case, the host owns counter/PEBS configuration, manages the PEBS output buffer, and assures the counter/PEBS state persists across guests, as illustrated by Figure 22-81.

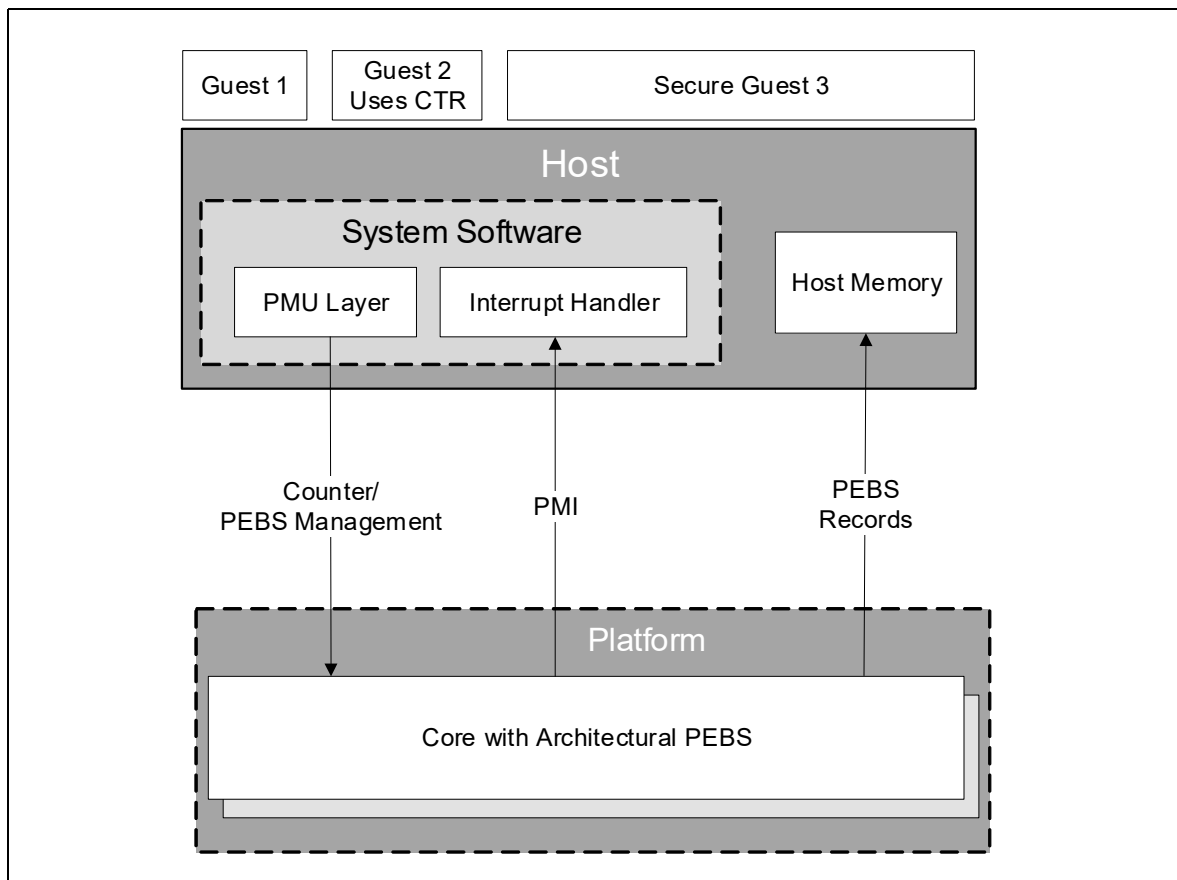


Figure 22-81. System-Wide PEBS: Diagram of Interactions Across Entities

To support the guest-only use case, the CPU defines a VM-execution control called "PEBS uses guest physical addresses." When this control is set, the addresses used by PEBS in VMX non-root operation are treated as guest-physical addresses and are translated through EPT (see Section 28.5.5, "Translation of Guest-Physical Addresses Used by Architectural PEBS"). A VMM setting this control can allow guest software to manage the IA32_PEBS_BASE MSR, as the MSR would be interpreted as having a guest-physical address.

To enable the guest-only PEBS use case, a VMM should set this VM-execution control and allocate the PEBS buffer using guest-physical addresses. Additionally, counters should be enabled only in VMX non-root operation for that guest (by suitable management of the IA32_PERF_GLOBAL_CTRL MSR). In other words, on VM exit from the

guest, the counters should be stopped, and on VM entry back to the guest, the counters should be resumed. This ensures that the counters and the PEBS record information belong to that guest.

In this use case, the guest owns counter/PEBS configuration, manages the PEBS output buffer, and switches the counter/PEBS state across applications inside that guest. The host can simply pass through the PEBS and counter management handling to that guest as illustrated by Figure 22-82.

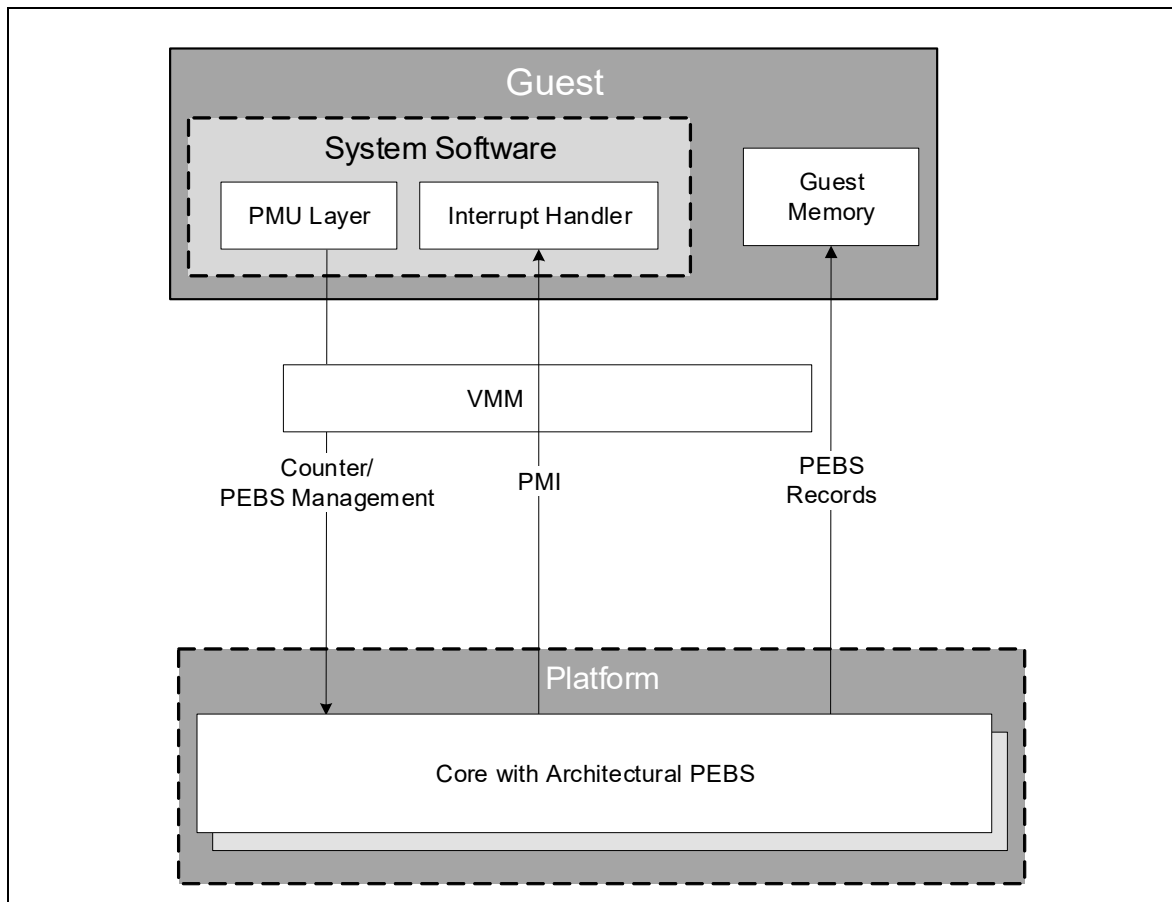


Figure 22-82. Guest-Only PEBS: Diagram of Interactions Across Entities

Other use cases, e.g., host-only, are possible but must be managed by the VMM.

22.10.5.2 Intel® Secure Guard Extensions (Intel® SGX)

For production (opt-out) enclaves, an enclave entry freezes counters, except fixed-function counters 1 and 2 (unhalted core cycles and unhalted reference cycles, respectively). Additionally, handling any PEBS event that may occur during an enclave execution is deferred until after the enclave exit, i.e., the PEBS event would skid to the end of EEXIT/AEX. Specifically, the instruction pointer field of the PEBS record would have the EENTER/ERESUME address and the Memory Access Address (MAA) field would be zero.

For non-production (opt-in) enclaves, all counters continue to count. Any PEBS event that may occur during enclave execution would be handled normally. Specifically, fields of the PEBS record would expose the non-production enclave state.

Architectural PEBS will write nothing to memory in the case where the PEBS buffer is mapped to the Processor Reserved Memory Range Register (PRMRR). Writes are directed to the abort page.

22.10.5.3 Intel® Trust Domain Extensions (Intel® TDX)

Intel TDX supports two modes of monitoring:

- TD Profiling – The trusted domain (TD) uses performance-monitoring/PEBS to profile itself.
- System Profiling – The VMM uses performance-monitoring/PEBS to profile itself plus any debuggable TDs.

For TD profiling mode, a TD may use performance-monitoring by setting the TDCS.ATTRIBUTES[PerfMon] bit. This enables a performance-monitoring-enabled TD to access all performance-monitoring MSRs. Architectural PEBS will be enumerated to the TD and the PEBS2GPA flag is set. The TD is expected to set up the PEBS buffer in TD private pages.

Architectural PEBS will not be available for TDs that are not performance-monitoring-enabled. Specifically, Leaf 23H Subleaves 4 and 5 will be cleared, and accesses to the IA32_PEBS_* and IA32_PMC_*_CFG_C MSRs will result in a #GP exception.

22.10.5.4 System Management Mode (SMM)

Performance-monitoring counters count through system management mode (SMM), unless IA32_DE-BUGCTL.FREEZE_WHILE_SMM_EN = 1.

PEBS will be inhibited on opt-out #SMI, inhibit cleared on RSM. This ensures any PEBS pended during SMM will skid to end of RSM and precludes any SMM use of PEBS.

22.10.5.5 SMM-Transfer Monitor (STM)

No STM-specific support is added with Architectural PEBS. However, the STM should configure the SMM-transfer VMCS as follows:

- The “save IA32_PERF_GLOBAL_CTRL” and “load IA32_PERF_GLOBAL_CTRL” VM-exit controls are set to 1.
- The host IA32_PERF_GLOBAL_CTRL value is zero.
- The “load IA32_PERF_GLOBAL_CTRL” VM-entry control is set to 1.

These ensure that non-SMM counters are disabled on an SMM VM exit, that any PEBS pended during such a VM exit are deferred, and that non-SMM counters are re-enabled by a VM entry that returns from SMM.

An STM is free to use PEBS itself, but it should ensure that non-SMM counter configuration state is restored before a VM entry that returns from SMM.

22.11 AUTO COUNTER RELOAD

Auto Counter Reload (ACR) provides a means for software to specify that, for each supported counter, the hardware should automatically reload the counter to a specified initial value upon overflow of chosen counters. This mechanism enables software to sample based on the relative rate of two (or more) events, such that a sample (PMI or PEBS) is taken only if the rate of one event exceeds some threshold relative to the rate of another event. Taking a PMI or PEBS only when the relative rate of performance-monitoring events crosses a threshold can have significantly less performance overhead than other techniques (e.g., taking a PMI every 1000 instructions in order to check the number of mispredicts since the last PMI).

22.11.1 Discovery and Interface

CPUID.23H.02H:EAX indicates general-purpose counters [n:0] that can be reloaded. CPUID.23H.02H:EBX indicates fixed-function counters [m:0] that can be reloaded. CPUID.23H.02H:ECX indicates general-purpose counters [n:0] that can cause a reload of reloadable counters. CPUID.23H.02H:EDX indicates fixed-function counters [m:0] that can cause a reload of reloadable counters. If a counter can be reloaded, its associated reload configuration MSR (*_CFG_B) and its reload value MSR (*_CFG_C) are supported.

See Chapter 2 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for details about the following MSRs: IA32_PMC_GPn_CFG_B, IA32_PMC_GPn_CFG_C, IA32_PMC_FXm_CFG_B, and IA32_PMC_FXm_CFG_C.

22.11.2 Configuration and Behavior

For a given counter `IA32_PMC_GPn_CTR`, bit fields in the `IA32_PMC_GPn_CFG_B` MSR indicate which counter(s) can cause a reload of that counter:

- If the general-purpose counter 'n' is configured to do a reload when general-purpose counter 'x' overflows (`IA32_PMC_GPn_CFG_B.PMC[x] = 1`), then that general-purpose counter 'n' will be written with its reload value (in `IA32_PMC_GPn_CFG_C[31:0]`) when counter 'x' (`IA32_PMC_GPx_CTR`) overflows.
- If general-purpose counter 'n' is configured to do a reload when fixed-function counter 'x' overflows (`IA32_PMC_GPn_CFG_B.FIXED_CTR[x] = 1`), then that general-purpose counter 'n' will be written with its reload value (in `IA32_PMC_GPn_CFG_C[31:0]`) when fixed-function counter 'x' (`IA32_PMC_FXx_CTR`) overflows.

ACR will not reload `IA32_PMC_GPn_CTR` if counters are frozen (`IA32_PERF_GLOBAL_STATUS.COUNTERS_FROZEN = 1`) or if `IA32_PMC_GPn_CTR` has already overflowed (`IA32_PERF_GLOBAL_STATUS.PMCn_OVF = 1`). If a PMI or PEBS is taken due to a counter overflow, the PMI ISR or PEBS record can record the unmodified counter value before reloading the counter. In race conditions, where `IA32_PMC_GPn_CTR` overflows in the same cycle as a counter configured to reload the `IA32_PMC_GPn_CTR` on overflow, `IA32_PMC_GPn_CTR` will not be reloaded, and `IA32_PERF_GLOBAL_STATUS.PMCn_OVF` will be set.

For counters that reload themselves (i.e., `IA32_PMC_GPn_CFG_B.PMCn = 1`), the overflow bit (`IA32_PERF_GLOBAL_STATUS.PMCn_OVF`) will never be set. Instead, upon overflow, the counter will be immediately reloaded; thus, it is never in an overflowed state. There is an exception associated with PEBS; see Section 22.11.2.2.

The behavior is similar for reloading of fixed-function counters. For `IA32_PMC_FXm_CTR`, the reload value is stored in `IA32_PMC_FXm_CFG_C[31:0]`, and which counters cause reload of `IA32_PMC_FXm_CTR` is configured in `IA32_PMC_FXm_CFG_B`.

22.11.2.1 Reload Precision

ACR reload is not guaranteed to be precise; in some cases, a small number of events may be lost during the time between counter overflow and counter reload. However, when the reload happens, hardware will reload all configured counters simultaneously.

22.11.2.2 PEBS Interaction

If a counter is configured to reload other counters with ACR and to take PEBS on overflow, the counter reload actions will be taken only after the PEBS record has been written. This ensures that any counter values captured in the PEBS record reflect the value before the reload occurs. Because the reload actions are taken after the PEBS records are written, reloaded counter value will not account for the events that occurred during the process of writing the PEBS record.

For a counter configured to reload itself and to take PEBS on overflow, the overflow bit associated with the counter (in `IA32_PERF_GLOBAL_STATUS`) will be set from the time the counter overflows to the time the PEBS record is written. This is required to ensure the PEBS record is not lost due to a VM exit taken during record generation. Once the record is written, the overflow bit will be cleared, and the counter reloaded.

22.11.2.3 Precise Distribution (PDIST) Interaction

Precise distribution of PEBS events (PDIR) is not supported when such a counter is reloaded by ACR. For details on PDIST, see Section 22.9.6.

11. Updates to Chapter 27, Volume 3C

Change bars and violet 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:

- Added control bit 12 for PEBS uses guest physical addresses to Table 27-8, "Definitions of Tertiary Processor-Based VM-Execution Controls" (in Section 27.6.2).

27.1 OVERVIEW

A logical processor uses **virtual-machine control data structures (VMCSs)** while it is in VMX operation. These manage transitions into and out of VMX non-root operation (VM entries and VM exits) as well as processor behavior in VMX non-root operation. This structure is manipulated by the new instructions VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.

A VMM can use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM can use a different VMCS for each virtual processor.

A logical processor associates a region in memory with each VMCS. This region is called the **VMCS region**.¹ Software references a specific VMCS using the 64-bit physical address of the region (a **VMCS pointer**). VMCS pointers must be aligned on a 4-KByte boundary (bits 11:0 must be zero). These pointers must not set bits beyond the processor's physical-address width.^{2,3}

A logical processor may maintain a number of VMCSs that are **active**. The processor may optimize VMX operation by maintaining the state of an active VMCS in memory, on the processor, or both. At any given time, at most one of the active VMCSs is the **current** VMCS. (This document frequently uses the term "the VMCS" to refer to the current VMCS.) The VMLAUNCH, VMREAD, VMRESUME, and VMWRITE instructions operate only on the current VMCS.

The following items describe how a logical processor determines which VMCSs are active and which is current:

- The memory operand of the VMPTRLD instruction is the address of a VMCS. After execution of the instruction, that VMCS is both active and current on the logical processor. Any other VMCS that had been active remains so, but no other VMCS is current.
- The VMCS link pointer field in the current VMCS (see Section 27.4.2) 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".

1. The amount of memory required for a VMCS region is at most 4 KBytes. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).

2. CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. This width is used to determine the value of MAXPHYADDR, which constrains VMCS pointers. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

3. If IA32_VMX_BASIC[48] is read as 1, these pointers must not set any bits in the range 63:32; see Appendix A.1.

- The memory operand of the VMCLEAR instruction is the address of a VMCS. After execution of the instruction, the launch state of that VMCS is “clear”.
- There are no other ways to modify the launch state of a VMCS (it cannot be modified using VMWRITE) and there is no direct way to discover it (it cannot be read using VMREAD).

Figure 27-1 illustrates the different states of a VMCS. It uses “X” to refer to the VMCS and “Y” to refer to any other VMCS. Thus: “VMPTRLD X” always makes X current and active; “VMPTRLD Y” always makes X not current (because it makes Y current); VMLAUNCH makes the launch state of X “launched” if X was current and its launch state was “clear”; and VMCLEAR X always makes X inactive and not current and makes its launch state “clear”.

The figure does not illustrate operations that do not modify the VMCS state relative to these parameters (e.g., execution of VMPTRLD X when X is already current). Note that VMCLEAR X makes X “inactive, not current, and clear,” even if X’s current state is not defined (e.g., even if X has not yet been initialized). See Section 27.11.3.

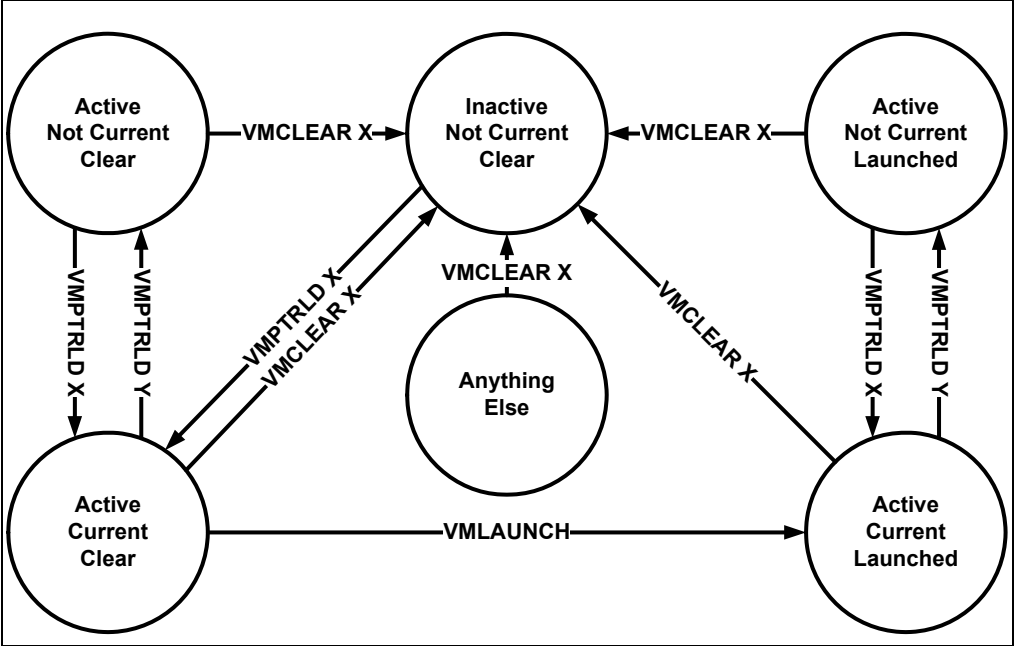


Figure 27-1. States of VMCS X

Because a shadow VMCS (see Section 27.10) cannot be used for VM entry, the launch state of a shadow VMCS is not meaningful. Figure 27-1 does not illustrate all the ways in which a shadow VMCS may be made active.

27.2 FORMAT OF THE VMCS REGION

A VMCS region comprises up to 4-KBytes.¹ The format of a VMCS region is given in Table 27-1.

Table 27-1. Format of the VMCS Region

Byte Offset	Contents
0	Bits 30:0: VMCS revision identifier Bit 31: shadow-VMCS indicator (see Section 27.10)
4	VMX-abort indicator

1. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).

Table 27-1. Format of the VMCS Region (Contd.)

Byte Offset	Contents
8	VMCS data (implementation-specific format)

The first 4 bytes of the VMCS region contain the **VMCS revision identifier** at bits 30:0.¹ Processors that maintain VMCS data in different formats (see below) use different VMCS revision identifiers. These identifiers enable software to avoid using a VMCS region formatted for one processor on a processor that uses a different format.² Bit 31 of this 4-byte region indicates whether the VMCS is a shadow VMCS (see Section 27.10).

Software should write the VMCS revision identifier to the VMCS region before using that region for a VMCS. The VMCS revision identifier is never written by the processor; VMPTRLD fails if its operand references a VMCS region whose VMCS revision identifier differs from that used by the processor. (VMPTRLD also fails if the shadow-VMCS indicator is 1 and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control; see Section 27.6.2) Software can discover the VMCS revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

Software should clear or set the shadow-VMCS indicator depending on whether the VMCS is to be an ordinary VMCS or a shadow VMCS (see Section 27.10). VMPTRLD fails if the shadow-VMCS indicator is set and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control. Software can discover support for this setting by reading the VMX capability MSR IA32_VMX_PROCBASED_CTL2 (see Appendix A.3.3).

The next 4 bytes of the VMCS region are used for the **VMX-abort indicator**. The contents of these bits do not control processor operation in any way. A logical processor writes a non-zero value into these bits if a VMX abort occurs (see Section 30.7). Software may also write into this field.

The remainder of the VMCS region is used for **VMCS data** (those parts of the VMCS that control VMX non-root operation and the VMX transitions). The format of these data is implementation-specific. VMCS data are discussed in Section 27.3 through Section 27.9. To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 27.11.4) in writeback cacheable memory. Future implementations may allow or require a different memory type³. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

27.3 ORGANIZATION OF VMCS DATA

The VMCS data are organized into six logical groups:

- **Guest-state area.** Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
- **Host-state area.** Processor state is loaded from the host-state area on VM exits.
- **VM-execution control fields.** These fields control processor behavior in VMX non-root operation. They determine in part the causes of VM exits.
- **VM-exit control fields.** These fields control VM exits.
- **VM-entry control fields.** These fields control VM entries.
- **VM-exit information fields.** These fields receive information on VM exits and describe the cause and the nature of VM exits. On some processors, these fields are read-only.⁴

1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.

2. Logical processors that use the same VMCS revision identifier use the same size for VMCS regions.

3. Alternatively, software may map any of these regions or structures with the UC memory type. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.

4. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

The VM-execution control fields, the VM-exit control fields, and the VM-entry control fields are sometimes referred to collectively as VMX controls.

27.4 GUEST-STATE AREA

This section describes fields contained in the guest-state area of the VMCS. VM entries load processor state from these fields and VM exits store processor state into these fields. See Section 29.3.2 and Section 30.3 for details.

27.4.1 Guest Register State

The following fields in the guest-state area correspond to processor registers:

- Control registers CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Debug register DR7 (64 bits; 32 bits on processors that do not support Intel 64 architecture).
- RSP, RIP, and RFLAGS (64 bits each; 32 bits on processors that do not support Intel 64 architecture).¹
- The following fields for each of the registers CS, SS, DS, ES, FS, GS, LDTR, and TR:
 - Selector (16 bits).
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture). The base-address fields for CS, SS, DS, and ES have only 32 architecturally-defined bits; nevertheless, the corresponding VMCS fields have 64 bits on processors that support Intel 64 architecture.
 - Segment limit (32 bits). The limit field is always a measure in bytes.
 - Access rights (32 bits). The format of this field is given in Table 27-2 and detailed as follows:
 - The low 16 bits correspond to bits 23:8 of the upper 32 bits of a 64-bit segment descriptor. While bits 19:16 of code-segment and data-segment descriptors correspond to the upper 4 bits of the segment limit, the corresponding bits (bits 11:8) are reserved in this VMCS field.
 - Bit 16 indicates an **unusable segment**. Attempts to use such a segment fault except in 64-bit mode. In general, a segment register is unusable if it has been loaded with a null selector.²
 - Bits 31:17 are reserved.

Table 27-2. Format of Access Rights

Bit Position(s)	Field
3:0	Segment type
4	S — Descriptor type (0 = system; 1 = code or data)
6:5	DPL — Descriptor privilege level
7	P — Segment present

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.
2. There are a few exceptions to this statement. For example, a segment with a non-null selector may be unusable following a task switch that fails after its commit point; see “Interrupt 10—Invalid TSS Exception (#TS)” in Section 7.14, “Exception and Interrupt Handling in 64-bit Mode,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. In contrast, the TR register is usable after processor reset despite having a null selector; see Table 13-1 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Table 27-2. Format of Access Rights (Contd.)

Bit Position(s)	Field
11:8	Reserved
12	AVL — Available for use by system software
13	Reserved (except for CS) L — 64-bit mode active (for CS only)
14	D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
15	G — Granularity
16	Segment unusable (0 = usable; 1 = unusable)
31:17	Reserved

The base address, segment limit, and access rights compose the “hidden” part (or “descriptor cache”) of each segment register. These data are included in the VMCS because it is possible for a segment register’s descriptor cache to be inconsistent with the segment descriptor in memory (in the GDT or the LDT) referenced by the segment register’s selector.

The value of the DPL field for SS is always equal to the logical processor’s current privilege level (CPL).¹

On some processors, executions of VMWRITE ignore attempts to write non-zero values to any of bits 11:8 or bits 31:17. On such processors, VMREAD always returns 0 for those bits, and VM entry treats those bits as if they were all 0 (see Section 29.3.1.2).

- The following fields for each of the registers GDTR and IDTR:
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - Limit (32 bits). The limit fields contain 32 bits even though these fields are specified as only 16 bits in the architecture.
- The following MSRs:
 - IA32_DEBUGCTL (64 bits)
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture)
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-entry control.
 - IA32_PAT (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_PAT” VM-entry control or that of the “save IA32_PAT” VM-exit control.
 - IA32_EFER (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_EFER” VM-entry control or that of the “save IA32_EFER” VM-exit control.
 - IA32_BNDCFGS (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control.
 - IA32_RTIT_CTL (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control.
 - IA32_LBR_CTL (64 bits). This field is supported only on processors that support either the 1-setting of the “load guest IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control.
 - IA32_S_CET (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.

1. In protected mode, CPL is also associated with the RPL field in the CS selector. However, the RPL fields are not meaningful in real-address mode or in virtual-8086 mode.

- IA32_INTERRUPT_SSP_TABLE_ADDR (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.
- IA32_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the “load PKRS” VM-entry control.
- The following MSRs on processors that support either the 1-setting of the “load FRED” VM-entry control or that of the “save FRED” VM-exit control (each is 64 bits):
 - IA32_FRED_CONFIG
 - IA32_FRED_RSP1
 - IA32_FRED_RSP2
 - IA32_FRED_RSP3
 - IA32_FRED_STKLVLS
 - IA32_FRED_SSP1
 - IA32_FRED_SSP2
 - IA32_FRED_SSP3
- The shadow-stack pointer register SSP (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.
- The register SMBASE (32 bits). This register contains the base address of the logical processor’s SMRAM image.

27.4.2 Guest Non-Register State

In addition to the register state described in Section 27.4.1, the guest-state area includes the following fields that characterize guest state but which do not correspond to processor registers:

- **Activity state** (32 bits). This field identifies the logical processor’s activity state. When a logical processor is executing instructions normally, it is in the **active state**. Execution of certain instructions and the occurrence of certain events may cause a logical processor to transition to an **inactive state** in which it ceases to execute instructions.

The following activity states are defined:¹

- 0: **Active**. The logical processor is executing instructions normally.
- 1: **HLT**. The logical processor is inactive because it executed the HLT instruction.
- 2: **Shutdown**. The logical processor is inactive because it incurred a **triple fault**² or some other serious error.
- 3: **Wait-for-SIPI**. The logical processor is inactive because it is waiting for a startup-IPI (SIPI).

Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.

- **Interruptibility state** (32 bits). The IA-32 architecture includes features that permit certain events to be blocked for a period of time. This field contains information about such blocking. Details and the format of this field are given in Table 27-3.
- **Pending debug exceptions** (64 bits; 32 bits on processors that do not support Intel 64 architecture). IA-32 processors may recognize one or more debug exceptions without immediately delivering them.³ This field contains information about such exceptions. This field is described in Table 27-4.

1. Execution of the MWAIT instruction may put a logical processor into an inactive state. However, this VMCS field never reflects this state. See Section 30.1.

2. A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

Table 27-3. Format of Interruptibility State

Bit Position(s)	Bit Name	Notes
0	Blocking by STI	See the “STI—Set Interrupt Flag” section in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B. Execution of STI with RFLAGS.IF = 0 blocks maskable interrupts on the instruction boundary following its execution. ¹ Setting this bit indicates that this blocking is in effect.
1	Blocking by MOV SS	See Section 7.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. Execution of a MOV to SS or a POP to SS blocks or suppresses certain debug exceptions as well as interrupts (maskable and nonmaskable) on the instruction boundary following its execution. Setting this bit indicates that this blocking is in effect. ² This document uses the term “blocking by MOV SS,” but it applies equally to POP SS.
2	Blocking by SMI	See Section 34.2, “System Management Interrupt (SMI).” System-management interrupts (SMIs) are disabled while the processor is in system-management mode (SMM). Setting this bit indicates that blocking of SMIs is in effect.
3	Blocking by NMI	See Section 7.7.1, “Handling Multiple NMIs,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A and Section 34.8, “NMI Handling While in SMM.” Delivery of a non-maskable interrupt (NMI) or a system-management interrupt (SMI) blocks subsequent NMIs until the next execution of IRET. See Section 28.3 for how this behavior of IRET may change in VMX non-root operation. Setting this bit indicates that blocking of NMIs is in effect. Clearing this bit does not imply that NMIs are not (temporarily) blocked for other reasons. If the “virtual NMIs” VM-execution control (see Section 27.6.1) is 1, this bit does not control the blocking of NMIs. Instead, it refers to “virtual-NMI blocking” (the fact that guest software is not ready for an NMI).
4	Enclave interruption	Set to 1 if the VM exit occurred while the logical processor was in enclave mode. Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode. A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field sets this bit; or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.
31:5	Reserved	VM entry will fail if these bits are not 0. See Section 29.3.1.5.

NOTES:

1. Nonmaskable interrupts and system-management interrupts may also be inhibited on the instruction boundary following such an execution of STI.
 2. System-management interrupts may also be inhibited on the instruction boundary following such an execution of MOV or POP.
- **VMCS link pointer** (64 bits). If the “VMCS shadowing” VM-execution control is 1, the VMREAD and VMWRITE instructions access the VMCS referenced by this pointer (see Section 27.10). Otherwise, software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 29.3.1.5).
-
3. For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 7.8.3 of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. In addition, certain events incident to an instruction (for example, an INIT signal) may take priority over debug traps generated by that instruction. See Table 7-2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Table 27-4. Format of Pending-Debug-Exceptions

Bit Position(s)	Bit Name	Notes
3:0	B3 – B0	When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if the corresponding enabling bit in DR7 is not set.
10:4	Reserved	VM entry fails if these bits are not 0. See Section 29.3.1.5.
11	BLD	When set, this bit indicates that a bus lock was asserted while OS bus-lock detection was enabled and CPL > 0 (see Section 20.3.1.6, “OS Bus-Lock Detection”). ¹
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7; the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled; or a bus lock was asserted while CPL > 0 and OS bus-lock detection had been enabled.
13	Reserved	VM entry fails if this bit is not 0. See Section 29.3.1.5.
14	BS	When set, this bit indicates that a debug exception would have been triggered by single-step execution mode.
15	Reserved	VM entry fails if this bit is not 0. See Section 29.3.1.5.
16	RTM	When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). ²
63:17	Reserved	VM entry fails if these bits are not 0. See Section 29.3.1.5. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 11 to indicate detection of a bus lock, while this field **sets** the bit to indicate that condition.
2. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- **VMX-preemption timer value** (32 bits). This field is supported only on processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control. This field contains the value that the VMX-preemption timer will use following the next VM entry with that setting. See Section 28.5.1 and Section 29.7.4.
- **Page-directory-pointer-table entries** (PDPTes; 64 bits each). These four (4) fields (PDPTe0, PDPTe1, PDPTe2, and PDPTe3) are supported only on processors that support the 1-setting of the “enable EPT” VM-execution control. They correspond to the PDPTes referenced by CR3 when PAE paging is in use (see Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). They are used only if the “enable EPT” VM-execution control is 1.
- **Guest interrupt status** (16 bits). This field is supported only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control. It characterizes part of the guest’s virtual-APIC state and does not correspond to any processor or APIC registers. It comprises two 8-bit subfields:
 - **Requesting virtual interrupt (RVI)**. This is the low byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is requesting service. (The value 0 implies that there is no such interrupt.)
 - **Servicing virtual interrupt (SVI)**. This is the high byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is in service. (The value 0 implies that there is no such interrupt.)

See Chapter 32 for more information on the use of this field.
- **PML index** (16 bits). This field is supported only on processors that support the 1-setting of the “enable PML” VM-execution control. It contains the logical index of the next entry in the page-modification log. Because the

page-modification log comprises 512 entries, the PML index is typically a value in the range 0–511. Details of the page-modification log and use of the PML index are given in Section 31.3.6.

27.5 HOST-STATE AREA

This section describes fields contained in the host-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM exit (see Section 30.5).

All fields in the host-state area correspond to processor registers:

- CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- RSP and RIP (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Selector fields (16 bits each) for the segment registers CS, SS, DS, ES, FS, GS, and TR. There is no field in the host-state area for the LDTR selector.
- Base-address fields for FS, GS, TR, GDTR, and IDTR (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- The following MSRs:
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
 - IA32_PAT (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PAT” VM-exit control.
 - IA32_EFER (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_EFER” VM-exit control.
 - IA32_S_CET (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.
 - IA32_INTERRUPT_SSP_TABLE_ADDR (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.
 - IA32_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the “load PKRS” VM-exit control.
 - The following MSRs on processors that support the 1-setting of the “load FRED” VM-exit control (each is 64 bits):
 - IA32_FRED_CONFIG
 - IA32_FRED_RSP1
 - IA32_FRED_RSP2
 - IA32_FRED_RSP3
 - IA32_FRED_STKLVL
 - IA32_FRED_SSP1
 - IA32_FRED_SSP2
 - IA32_FRED_SSP3
- The shadow-stack pointer register SSP (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.

In addition to the state identified here, some processor state components are loaded with fixed values on every VM exit; there are no fields corresponding to these components in the host-state area. See Section 30.5 for details of how state is loaded on VM exits.

27.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 27.6.1 through Section 27.6.8.

27.6.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts).¹ Table 27-5 lists the controls. See Chapter 29 for how these controls affect processor behavior in VMX non-root operation.

Table 27-5. Definitions of Pin-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	External-interrupt exiting	If this control is 1, external interrupts cause VM exits. Otherwise, they are delivered normally. If this control is 1, the value of RFLAGS.IF does not affect interrupt blocking.
3	NMI exiting	If this control is 1, non-maskable interrupts (NMIs) cause VM exits. Otherwise, they are delivered normally using vector 2. This control also determines interactions between IRET and blocking by NMI (see Section 28.3).
5	Virtual NMIs	If this control is 1, NMIs are never blocked and the “blocking by NMI” bit (bit 3) in the interruptibility-state field indicates “virtual-NMI blocking” (see Table 27-3). This control also interacts with the “NMI-window exiting” VM-execution control (see Section 27.6.2).
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 28.5.1. A VM exit occurs when the timer counts down to zero; see Section 28.2.
7	Process posted interrupts	If this control is 1, the processor treats interrupts with the posted-interrupt notification vector (see Section 27.6.8) specially, updating the virtual-APIC page with posted-interrupt requests (see Section 32.6).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs `IA32_VMX_PINBASED_CTL`s and `IA32_VMX_TRUE_PINBASED_CTL`s (see Appendix A.3.1) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 29.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 2, and 4. The VMX capability MSR `IA32_VMX_PINBASED_CTL`s 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_CTL`s MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

27.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls constitute three vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions.² These are the **primary processor-based VM-execution controls** (32 bits), the **secondary processor-based VM-execution controls** (32 bits), and the tertiary **VM-execution controls** (64 bits).

1. Some asynchronous events cause VM exits regardless of the settings of the pin-based VM-execution controls (see Section 28.2).
2. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 28.1.2), as do task switches (see Section 28.2).

Table 27-6 lists the primary processor-based VM-execution controls. See Chapter 27 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 27-6. Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there are no other blocking of interrupts (see Section 27.4.2).
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 27.6.5 and Section 28.3).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG and INVPCID cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC cause VM exits.
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP cause VM exits.
15	CR3-load exiting	In conjunction with the CR3-target controls (see Section 27.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 28.1.3. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
16	CR3-store exiting	This control determines whether executions of MOV from CR3 cause VM exits. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
17	Activate tertiary controls	This control determines whether the tertiary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the tertiary processor-based VM-execution controls were also 0.
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits.
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits.
21	Use TPR shadow	Setting this control to 1 enables TPR virtualization and other APIC-virtualization features. See Chapter 32.
22	NMI-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if there is no virtual-NMI blocking (see Section 27.4.2).
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 27.6.4 and Section 28.1.3). For this control, “0” means “do not use I/O bitmaps” and “1” means “use I/O bitmaps.” If the I/O bitmaps are used, the setting of the “unconditional I/O exiting” control is ignored.
27	Monitor trap flag	If this control is 1, the monitor trap flag debugging feature is enabled. See Section 28.5.2.
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 27.6.9 and Section 28.1.3). For this control, “0” means “do not use MSR bitmaps” and “1” means “use MSR bitmaps.” If the MSR bitmaps are not used, all executions of the RDMSR and WRMSR instructions cause VM exits.
29	MONITOR exiting	This control determines whether executions of MONITOR cause VM exits.
30	PAUSE exiting	This control determines whether executions of PAUSE cause VM exits.
31	Activate secondary controls	This control determines whether the secondary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the secondary processor-based VM-execution controls were also 0.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs `IA32_VMX_PROCBASED_CTLS` and `IA32_VMX_TRUE_PROCBASED_CTLS` (see Appendix A.3.2) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 29.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 4–6, 8, 13–16, and 26. The VMX capability MSR `IA32_VMX_PROCBASED_CTLS` will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR `IA32_VMX_TRUE_PROCBASED_CTLS` MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-execution controls determines whether the secondary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the secondary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 31 of the primary processor-based VM-execution controls do not support the secondary processor-based VM-execution controls.

Table 27-7 lists the secondary processor-based VM-execution controls. See Chapter 27 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 27-7. Definitions of Secondary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	Virtualize APIC accesses	If this control is 1, the logical processor treats specially accesses to the page with the APIC-access address. See Section 32.4.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Section 31.3.
2	Descriptor-table exiting	This control determines whether executions of <code>LGDT</code> , <code>LIDT</code> , <code>LLDT</code> , <code>LTR</code> , <code>SGDT</code> , <code>SIDT</code> , <code>SLDT</code> , and <code>STR</code> cause VM exits.
3	Enable RDTSCP	If this control is 0, any execution of <code>RDTSCP</code> causes an invalid-opcode exception (<code>#UD</code>).
4	Virtualize x2APIC mode	If this control is 1, the logical processor treats specially <code>RDMSR</code> and <code>WRMSR</code> to APIC MSRs (in the range 800H–8FFH). See Section 32.5.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 31.1.
6	WBINVD exiting	This control determines whether executions of <code>WBINVD</code> and <code>WBNOINVD</code> cause VM exits.
7	Unrestricted guest	This control determines whether guest software may run in unpaged protected mode or in real-address mode.
8	APIC-register virtualization	If this control is 1, the logical processor virtualizes certain APIC accesses. See Section 32.4 and Section 32.5.
9	Virtual-interrupt delivery	This controls enables the evaluation and delivery of pending virtual interrupts as well as the emulation of writes to the APIC registers that control interrupt prioritization.
10	PAUSE-loop exiting	This control determines whether a series of executions of <code>PAUSE</code> can cause a VM exit (see Section 27.6.13 and Section 28.1.3).
11	RDRAND exiting	This control determines whether executions of <code>RDRAND</code> cause VM exits.
12	Enable INVPCID	If this control is 0, any execution of <code>INVPCID</code> causes a <code>#UD</code> .
13	Enable VM functions	Setting this control to 1 enables use of the <code>VMFUNC</code> instruction in VMX non-root operation. See Section 28.5.7.
14	VMCS shadowing	If this control is 1, executions of <code>VMREAD</code> and <code>VMWRITE</code> in VMX non-root operation may access a shadow VMCS (instead of causing VM exits). See Section 27.10 and Section 33.3.
15	Enable ENCLS exiting	If this control is 1, executions of <code>ENCLS</code> consult the <code>ENCLS</code> -exiting bitmap to determine whether the instruction causes a VM exit. See Section 27.6.16 and Section 28.1.3.
16	RDSEED exiting	This control determines whether executions of <code>RDSEED</code> cause VM exits.
17	Enable PML	If this control is 1, an access to a guest-physical address that sets an EPT dirty bit first adds an entry to the page-modification log. See Section 31.3.6.

Table 27-7. Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
18	EPT-violation #VE	If this control is 1, EPT violations may cause virtualization exceptions (#VE) instead of VM exits. See Section 28.5.8.
19	Conceal VMX from PT	If this control is 1, Intel Processor Trace suppresses from PIPs an indication that the processor was in VMX non-root operation and omits a VMCS packet from any PSB+ produced in VMX non-root operation (see Chapter 36).
20	Enable XSAVES/XRSTORS	If this control is 0, any execution of XSAVES or XRSTORS causes a #UD.
21	PASID translation	If this control is 1, PASID translation is performed for executions of ENQCMD and ENQCMLS. See Section 28.5.9.
22	Mode-based execute control for EPT	If this control is 1, EPT execute permissions are based on whether the linear address being accessed is supervisor mode or user mode. See Chapter 31.
23	Sub-page write permissions for EPT	If this control is 1, EPT write permissions may be specified at the granularity of 128 bytes. See Section 31.3.4.
24	Intel PT uses guest physical addresses	If this control is 1, all output addresses used by Intel Processor Trace are treated as guest-physical addresses and translated using EPT. See Section 28.5.4.
25	Use TSC scaling	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC multiplier field (see Section 27.6.5 and Section 28.3).
26	Enable user wait and pause	If this control is 0, any execution of TPAUSE, UMONITOR, or UMWAIT causes a #UD.
27	Enable PCONFIG	If this control is 0, any execution of PCONFIG causes a #UD.
30	VMM bus-lock detection	This control determines whether assertion of a bus lock causes a VM exit. See Section 28.2.
31	Instruction timeout	If this control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within a specified amount of time. See Section 27.6.24 and Section 28.2.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROC-BASED_CTLSS2 (see Appendix A.3.3) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 29.2.1.1).

Bit 17 of the primary processor-based VM-execution controls determines whether the tertiary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the tertiary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 17 of the primary processor-based VM-execution controls do not support the tertiary processor-based VM-execution controls.

Table 27-8 lists the tertiary processor-based VM-execution controls. See Chapter 27 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 27-8. Definitions of Tertiary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	LOADIWKEY exiting	This control determines whether executions of LOADIWKEY cause VM exits.
1	Enable HLAT	This control enables hypervisor-managed linear-address translation. See Section 5.5.1.
2	EPT paging-write control	If this control is 1, EPT permissions can be specified to allow writes only for paging-related updates. See Section 31.3.3.2.
3	Guest-paging verification	If this control is 1, EPT permissions can be specified to prevent accesses using linear addresses whose translation has certain properties. See Section 31.3.3.2.
4	IPI virtualization	If this control is 1, virtualization of interprocessor interrupts (IPIs) is enabled. See Section 32.1.6.

Table 27-8. Definitions of Tertiary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
5	SEAM guest-physical address width	This control determines the operation of EPT in SEAM non-root operation. See Section 35.3.2.
6	Enable MSR-list instructions	If this control is 0, any execution of RDMSRLIST or WRMSRLIST causes a #UD.
7	Virtualize IA32_SPEC_CTRL	If this control is 1, the operation of the RDMSR and WRMSR instructions is changed when accessing the IA32_SPEC_CTRL MSR. See Section 26.3.
9	Enable PBNDKB	If this control is 0, any execution of the PBBNDKB causes a #UD.
12	PEBS uses guest physical addresses	If this control is 1, all output addresses used by architectural PEBS are treated as guest-physical addresses and translated using EPT. See Section 28.5.5.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROC-BASED_CTLSS3 (see Appendix A.3.4) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 29.2.1.1).

27.6.3 Exception Bitmap

The **exception bitmap** is a 32-bit field that contains one bit for each exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is 1, the exception causes a VM exit. If the bit is 0, the exception is delivered normally using the exception's vector.

Whether a page fault (exception with vector 14) causes a VM exit is determined by bit 14 in the exception bitmap as well as the error code produced by the page fault and two 32-bit fields in the VMCS (the **page-fault error-code mask** and **page-fault error-code match**). See Section 28.2 for details.

27.6.4 I/O-Bitmap Addresses

The VM-execution control fields include the 64-bit physical addresses of **I/O bitmaps** A and B (each of which are 4 KBytes in size). I/O bitmap A contains one bit for each I/O port in the range 0000H through 7FFFH; I/O bitmap B contains bits for ports in the range 8000H through FFFFH.

A logical processor uses these bitmaps if and only if the "use I/O bitmaps" control is 1. If the bitmaps are used, execution of an I/O instruction causes a VM exit if any bit in the I/O bitmaps corresponding to a port it accesses is 1. See Section 28.1.3 for details. If the bitmaps are used, their addresses must be 4-KByte aligned.

27.6.5 Time-Stamp Counter Offset and Multiplier

The VM-execution control fields include a 64-bit **TSC-offset** field. If the "RDTSC exiting" control is 0 and the "use TSC offsetting" control is 1, this field controls executions of the RDTSC and RDTSCP instructions. It also controls executions of the RDMSR instruction that read from the IA32_TIME_STAMP_COUNTER MSR. For all of these, the value of the TSC offset is added to the value of the time-stamp counter, and the sum is returned to guest software in EDX:EAX.

Processors that support the 1-setting of the "use TSC scaling" control also support a 64-bit **TSC-multiplier** field. If this control is 1 (and the "RDTSC exiting" control is 0 and the "use TSC offsetting" control is 1), this field also affects the executions of the RDTSC, RDTSCP, and RDMSR instructions identified above. Specifically, the contents of the time-stamp counter is first multiplied by the TSC multiplier before adding the TSC offset.

See Chapter 27 for a detailed treatment of the behavior of RDTSC, RDTSCP, and RDMSR in VMX non-root operation.

27.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4

VM-execution control fields include **guest/host masks** and **read shadows** for the CR0 and CR4 registers. These fields control executions of instructions that access those registers (including CLTS, LMSW, MOV CR, and SMSW). They are 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

In general, bits set to 1 in a guest/host mask correspond to bits “owned” by the host:

- Guest attempts to set them (using CLTS, LMSW, or MOV to CR) to values differing from the corresponding bits in the corresponding read shadow cause VM exits.
- Guest reads (using MOV from CR or SMSW) return values for these bits from the corresponding read shadow.

Bits cleared to 0 correspond to bits “owned” by the guest; guest attempts to modify them succeed and guest reads return values for these bits from the control register itself.

See Chapter 29 for details regarding how these fields affect VMX non-root operation.

27.6.7 CR3-Target Controls

The VM-execution control fields include a set of 4 **CR3-target values** and a **CR3-target count**. The CR3-target values each have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not. The CR3-target count has 32 bits on all processors.

An execution of MOV to CR3 in VMX non-root operation does not cause a VM exit if its source operand matches one of these values. If the CR3-target count is n , only the first n CR3-target values are considered; if the CR3-target count is 0, MOV to CR3 always causes a VM exit.

There are no limitations on the values that can be written for the CR3-target values. VM entry fails (see Section 29.2) if the CR3-target count is greater than 4.

Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine the number of values supported.

27.6.8 Controls for APIC Virtualization

There are three mechanisms by which software accesses registers of the logical processor’s local APIC:

- If the local APIC is in xAPIC mode, it can perform memory-mapped accesses to addresses in the 4-KByte page referenced by the physical address in the IA32_APIC_BASE MSR (see Section 13.4.4, “Local APIC Status and Location,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, and the Intel® 64 Architecture Processor Topology Enumeration Technical Paper).¹
- If the local APIC is in x2APIC mode, it can access the local APIC’s registers using the RDMSR and WRMSR instructions (see the Intel® 64 Architecture Processor Topology Enumeration Technical Paper).
- In 64-bit mode, it can access the local APIC’s task-priority register (TPR) using the MOV CR8 instruction.

Several processor-based VM-execution controls (see Section 27.6.2) control such accesses. These are “use TPR shadow”, “virtualize APIC accesses”, “virtualize x2APIC mode”, “virtual-interrupt delivery”, “APIC-register virtualization”, and “IPI virtualization”. These controls interact with the following fields:

- **APIC-access address** (64 bits). This field contains the physical address of the 4-KByte **APIC-access page**. If the “virtualize APIC accesses” VM-execution control is 1, access to this page may cause VM exits or be virtualized by the processor. See Section 32.4.
The APIC-access address exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.
- **Virtual-APIC address** (64 bits). This field contains the physical address of the 4-KByte **virtual-APIC page**. The processor uses the virtual-APIC page to virtualize certain accesses to APIC registers and to manage virtual interrupts; see Chapter 32.

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

Depending on the setting of the controls indicated earlier, the virtual-APIC page may be accessed by the following operations:

- The MOV CR8 instructions (see Section 32.3).
- Accesses to the APIC-access page if, in addition, the “virtualize APIC accesses” VM-execution control is 1 (see Section 32.4).
- The RDMSR and WRMSR instructions if, in addition, the value of ECX is in the range 800H–8FFH (indicating an APIC MSR) and the “virtualize x2APIC mode” VM-execution control is 1 (see Section 32.5).

If the “use TPR shadow” VM-execution control is 1, VM entry ensures that the virtual-APIC address is 4-KByte aligned. The virtual-APIC address exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

- **TPR threshold** (32 bits). Bits 3:0 of this field determine the threshold below which bits 7:4 of VTPR (see Section 32.1.1) cannot fall. If the “virtual-interrupt delivery” VM-execution control is 0, a VM exit occurs after an operation (e.g., an execution of MOV to CR8) that reduces the value of those bits below the TPR threshold. See Section 32.1.2.

The TPR threshold exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

- **EOI-exit bitmap** (4 fields; 64 bits each). These fields are supported only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control. They are used to determine which virtualized writes to the APIC’s EOI register cause VM exits:
 - EOI_EXIT0 contains bits for vectors from 0 (bit 0) to 63 (bit 63).
 - EOI_EXIT1 contains bits for vectors from 64 (bit 0) to 127 (bit 63).
 - EOI_EXIT2 contains bits for vectors from 128 (bit 0) to 191 (bit 63).
 - EOI_EXIT3 contains bits for vectors from 192 (bit 0) to 255 (bit 63).

See Section 32.1.4 for more information on the use of this field.

- **Posted-interrupt notification vector** (16 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. Its low 8 bits contain the interrupt vector that is used to notify a logical processor that virtual interrupts have been posted. See Section 32.6 for more information on the use of this field.
- **Posted-interrupt descriptor address** (64 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. It is the physical address of a 64-byte aligned posted interrupt descriptor. See Section 32.6 for more information on the use of this field.
- **PID-pointer table address** (64 bits). This field contains the physical address of the **PID-pointer table**. If the “IPI virtualization” VM-execution control is 1, the logical processor uses entries in this table to virtualize IPIs. See Section 32.1.6.
- **Last PID-pointer index** (16 bits). This field contains the index of the last entry in the PID-pointer table.

27.6.9 MSR-Bitmap Address

On processors that support the 1-setting of the “use MSR bitmaps” VM-execution control, the VM-execution control fields include the 64-bit physical address of four contiguous **MSR bitmaps**, which are each 1-KByte in size. This field does not exist on processors that do not support the 1-setting of that control. The four bitmaps are:

- **Read bitmap for low MSRs** (located at the MSR-bitmap address). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Read bitmap for high MSRs** (located at the MSR-bitmap address plus 1024). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Write bitmap for low MSRs** (located at the MSR-bitmap address plus 2048). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.

- **Write bitmap for high MSRs** (located at the MSR-bitmap address plus 3072). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.

A logical processor uses these bitmaps if and only if the “use MSR bitmaps” control is 1. If the bitmaps are used, an execution of RDMSR or WRMSR causes a VM exit if the value of RCX is in neither of the ranges covered by the bitmaps or if the appropriate bit in the MSR bitmaps (corresponding to the instruction and the RCX value) is 1. See Section 28.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

27.6.10 Executive-VMCS Pointer

The executive-VMCS pointer is a 64-bit field used in the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). SMM VM exits save this field as described in Section 34.15.2. VM entries that return from SMM use this field as described in Section 34.15.4.

27.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of EPT paging structure that is the root of the hierarchy of structures that control the translation of guest-physical addresses (see Section 31.3.2), as well as other EPT configuration information. The format of this field is shown in Table 27-9.

Table 27-9. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 31.3.7): 0 = Uncacheable (UC) 6 = Write-back (WB) Other values are reserved. ¹
5:3	This value is 1 less than the EPT page-walk length (see Section 31.3.2)
6	Setting this control to 1 enables accessed and dirty flags for EPT (see Section 31.3.5) ²
7	Setting this control to 1 enables enforcement of access rights for supervisor shadow-stack pages (see Section 31.3.3.2) ³
11:8	Reserved
M-1:12	Bits M-1:12 of the physical address of the 4-KByte aligned EPT paging-structure (an EPT PML4 table with 4-level EPT and an EPT PML5 table with 5-level EPT) ⁴
63:M	Reserved

NOTES:

1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT paging-structure memory types are supported.
2. Not all processors support accessed and dirty flags for EPT. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
3. Not all processors enforce access rights for shadow-stack pages. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
4. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 27-1.

The EPTP exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

27.6.12 Virtual-Processor Identifier (VPID)

The **virtual-processor identifier** (VPID) is a 16-bit field. It exists only on processors that support the 1-setting of the “enable VPID” VM-execution control. See Section 31.1 for details regarding the use of this field.

27.6.13 Controls for PAUSE-Loop Exiting

On processors that support the 1-setting of the “PAUSE-loop exiting” VM-execution control, the VM-execution control fields include the following 32-bit fields:

- **PLE_Gap**. Software can configure this field as an upper bound on the amount of time between two successive executions of PAUSE in a loop.
- **PLE_Window**. Software can configure this field as an upper bound on the amount of time a guest is allowed to execute in a PAUSE loop.

These fields measure time based on a counter that runs at the same rate as the timestamp counter (TSC). See Section 28.1.3 for more details regarding PAUSE-loop exiting.

27.6.14 VM-Function Controls

The **VM-function controls** constitute a 64-bit vector that governs use of the VMFUNC instruction in VMX non-root operation. This field is supported only on processors that support the 1-settings of both the “activate secondary controls” primary processor-based VM-execution control and the “enable VM functions” secondary processor-based VM-execution control.

Table 27-10 lists the VM-function controls. See Section 28.5.7 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 27-10. Definitions of VM-Function Controls

Bit Position(s)	Name	Description
0	EPTP switching	The EPTP-switching VM function changes the EPT pointer to a value chosen from the EPTP list. See Section 28.5.7.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_VMFUNC (see Appendix A.11) to determine which bits are reserved. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 29.2.1.1).

Processors that support the 1-setting of the “EPTP switching” VM-function control also support a 64-bit field called the **EPTP-list address**. This field contains the physical address of the 4-KByte **EPTP list**. The EPTP list comprises 512 8-Byte entries (each an EPTP value) and is used by the EPTP-switching VM function (see Section 28.5.7.3).

27.6.15 VMCS Shadowing Bitmap Addresses

On processors that support the 1-setting of the “VMCS shadowing” VM-execution control, the VM-execution control fields include the 64-bit physical addresses of the **VMREAD bitmap** and the **VMWRITE bitmap**. Each bitmap is 4 KBytes in size and thus contains 32 KBits. The addresses are the **VMREAD-bitmap address** and the **VMWRITE-bitmap address**.

If the “VMCS shadowing” VM-execution control is 1, executions of VMREAD and VMWRITE may consult these bitmaps (see Section 27.10 and Section 33.3).

27.6.16 ENCLS-Exiting Bitmap

The **ENCLS-exiting bitmap** is a 64-bit field. If the “enable ENCLS exiting” VM-execution control is 1, execution of ENCLS causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the bit is 0, the instruction executes normally. See Section 28.1.3 for more information.

27.6.17 PCONFIG-Exiting Bitmap

The **PCONFIG-exiting bitmap** is a 64-bit field. If the “enable PCONFIG” VM-execution control is 1, execution of PCONFIG causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the control is 0, any execution of PCONFIG causes a #UD. See Section 28.1.3 for more information.

27.6.18 Control Field for Page-Modification Logging

The **PML address** is a 64-bit field. It is the 4-KByte aligned address of the **page-modification log**. The page-modification log consists of 512 64-bit entries. It is used for the page-modification logging feature. Details of the page-modification logging are given in Section 31.3.6.

If the “enable PML” VM-execution control is 1, VM entry ensures that the PML address is 4-KByte aligned. The PML address exists only on processors that support the 1-setting of the “enable PML” VM-execution control.

27.6.19 Controls for Virtualization Exceptions

On processors that support the 1-setting of the “EPT-violation #VE” VM-execution control, the VM-execution control fields include the following:

- **Virtualization-exception information address** (64 bits). This field contains the physical address of the **virtualization-exception information area**. When a logical processor encounters a virtualization exception, it saves virtualization-exception information at the virtualization-exception information address; see Section 28.5.8.2.
- **EPTP index** (16 bits). When an EPT violation causes a virtualization exception, the processor writes the value of this field to the virtualization-exception information area. The EPTP-switching VM function updates this field (see Section 28.5.7.3).

27.6.20 XSS-Exiting Bitmap

On processors that support the 1-setting of the “enable XSAVES/XRSTORS” VM-execution control, the VM-execution control fields include a 64-bit **XSS-exiting bitmap**. If the “enable XSAVES/XRSTORS” VM-execution control is 1, executions of XSAVES and XRSTORS may consult this bitmap (see Section 28.1.3 and Section 28.3).

27.6.21 Sub-Page-Permission-Table Pointer (SPPTP)

If the sub-page write-permission feature of EPT is enabled, EPT write permissions may be determined at a 128-byte granularity (see Section 31.3.4). These permissions are determined using a hierarchy of sub-page-permission structures in memory.

The root of this hierarchy is referenced by a VM-execution control field called the **sub-page-permission-table pointer** (SPPTP). The SPPTP contains the address of the base of the root SPP table (see Section 31.3.4.2). The format of this field is shown in Table 27-9.

Table 27-11. Format of Sub-Page-Permission-Table Pointer

Bit Position(s)	Field
11:0	Reserved
M-1:12	Bits N-1:12 of the physical address of the 4-KByte aligned root SPP table ¹
63:M	Reserved

NOTES:

1. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 27-1.

The SPPTP exists only on processors that support the 1-setting of the “sub-page write permissions for EPT” VM-execution control.

27.6.22 Fields Related to Hypervisor-Managed Linear-Address Translation

Two fields are used when the “enable HLAT” VM-execution control is 1, enabling HLAT paging:

- The **hypervisor-managed linear-address translation pointer** (HLAT pointer or HLATP) is used by HLAT paging to locate and access the first paging structure used for linear-address translation (see Section 5.5). The format of this field is shown in Table 27-12.

Table 27-12. Format of Hypervisor-Managed Linear-Address Translation Pointer

Bit Position(s)	Field
2:0	Reserved
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the first HLAT paging structure during linear-address translation.
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the first HLAT paging structure during linear-address translation.
11:5	Reserved
M-1:12	Guest-physical address (4KB-aligned) of the first HLAT paging structure during linear-address translation. ¹
63:M	Reserved

NOTES:

1. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 27-1.

- The HLAT prefix size. The value of this field determines which linear address are subject to HLAT paging. See Section 5.5.1.

These fields exist only on processors that support the 1-setting of the “enable HLAT” VM-execution control.

27.6.23 Fields Related to PASID Translation

Two 64-bit VM-execution control fields are used when the “PASID translation” VM-execution control is 1, enabling translation of PASIDs for executions of ENQCMD and ENQCMLS: the **low PASID directory address** and the **high PASID directory address**. These are the physical addresses of the low PASID directory and the high PASID directory, respectively. These fields exist only on processors that support the 1-setting of the “PASID translation” VM-execution control.

See Section 28.5.9 for information on the PASID-translation process for ENQCMD and ENQCMLS.

27.6.24 Instruction-Timeout Control

On processors that support the 1-setting of the “instruction timeout” VM-execution control, the VM-execution control fields include a 32-bit **instruction-timeout control**. The processor interprets the value of this field as an amount of time as measured in units of crystal clock cycles.¹ If the “instruction timeout” VM-execution control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within this amount of time.

1. CPUID.15H:ECX enumerates the nominal frequency of the core crystal clock in Hz.

27.6.25 Fields Controlling Virtualization of the IA32_SPEC_CTRL MSR

On processors that support the 1-setting of the “virtualize IA32_SPEC_CTRL” VM-execution control, the VM-execution control fields include the following 64-bit fields:

- **IA32_SPEC_CTRL mask.** Setting a bit in this field prevents guest software from modifying the corresponding bit in the IA32_SPEC_CTRL MSR.
- **IA32_SPEC_CTRL shadow.** This field contains the value that guest software expects to be in the IA32_SPEC_CTRL MSR.

Section 28.3 discusses how these fields are used in VMX non-root operation.

27.6.26 Fields Controlling SEAM Non-Root Operation

On processors that support the 1-setting of the “SEAM guest-physical address width” VM-execution control, the VM-execution control fields include the following fields:

- **SEAM-guest KeyID.** This 32-bit field contains the key identifier associated with the key used to encrypt SEAM private memory for the current SEAM guest.
- **SEAM shared EPT pointer.** This 64-bit field contains the address of the EPT paging structure that is the root of the hierarchy of structures that control the translation of shared guest-physical addresses. The format of this field is shown in Table 27-13.

Table 27-13. Format of SEAM Shared EPT Pointer

Bit Position(s)	Field
11:0	Reserved
M-1:12	Bits M-1:12 of the physical address of the 4-KByte aligned EPT paging-structure (an EPT PML4 table with 4-level EPT and an EPT PML5 table with 5-level EPT) ¹
63:M	Reserved

NOTES:

1. M is the value enumerated by CPUID.80000008H:EAX[7:0].

These fields control SEAM non-root operation. They are ignored outside of SEAM.

27.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 27.7.1 and Section 27.7.2.

27.7.1 VM-Exit Controls

The VM-exit controls constitute two vectors that govern the basic operation of VM exits. These are the **primary VM-exit controls** (32 bits) and the **secondary VM-exits controls** (64 bits).

Table 27-14 lists the primary VM-exit controls. See Chapter 29 for complete details of how these controls affect VM exits.

Table 27-14. Definitions of Primary VM-Exit Controls

Bit Position(s)	Name	Description
2	Save debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are saved on VM exit. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	Host address-space size	On processors that support Intel 64 architecture, this control determines whether a logical processor is in 64-bit mode after the next VM exit. Its value is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
12	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM exit.
15	Acknowledge interrupt on exit	This control affects VM exits due to external interrupts: ▪ If such a VM exit occurs and this control is 1, the logical processor acknowledges the interrupt controller, acquiring the interrupt's vector. The vector is stored in the exiting-event identification field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the exiting-event identification field is marked invalid.
18	Save IA32_PAT	This control determines whether the IA32_PAT MSR is saved on VM exit.
19	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM exit.
20	Save IA32_EFER	This control determines whether the IA32_EFER MSR is saved on VM exit.
21	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM exit.
22	Save VMX-preemption timer value	This control determines whether the value of the VMX-preemption timer is saved on VM exit.
23	Clear IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is cleared on VM exit.
24	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM exit or a VMCS packet on an SMM VM exit (see Chapter 36).
25	Clear IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is cleared on VM exit.
26	Clear IA32_LBR_CTL	This control determines whether the IA32_LBR_CTL MSR is cleared on VM exit.
27	Clear UINV	This control determines whether UINV is cleared on VM exit.
28	Load CET state	This control determines whether CET-related MSRs and SSP are loaded on VM exit.
29	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded on VM exit.
30	Save IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is saved on VM exit.
31	Activate secondary controls	This control determines whether the secondary VM-exit controls are used. If this control is 0, the logical processor operates as if all the secondary VM-exit controls were also 0.

NOTES:

1. Since the Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CR0.PG and IA32_EFER.LME, and since CR0.PG is always 1 in VMX root operation, IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX root operation.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_EXIT_CTLS and IA32_VMX_TRUE_EXIT_CTLS (see Appendix A.4) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 29.2.1.2).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8, 10, 11, 13, 14, 16, and 17. The VMX capability MSR IA32_VMX_EXIT_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX-

_TRUE_EXIT_CTLX MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-exit controls determines whether the secondary VM-exit controls are used. If that bit is 0, VM entries and VM exits function as if all the secondary VM-exit controls were 0. Processors that support only the 0-setting of bit 31 of the primary VM-exit controls do not support the secondary VM-exit controls.

Table 27-15 lists the secondary VM-exit controls. See Chapter 29 for more details of how these controls affect VM exits.

Table 27-15. Definitions of Secondary VM-Exit Controls

Bit Position(s)	Name	Description
0	Save FRED	This control determines whether the FRED MSRs are saved on VM exit.
1	Load FRED	This control determines whether the FRED MSRs are loaded on VM exit.
3	Prematurely busy shadow stack	If this control is 1, VM exits that cause a shadow stack to become prematurely busy (see Section 18.2.3, “Supervisor Shadow Stack Token,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1) indicate this fact and save additional information into the VMCS.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_EXIT_CTLX2 (see Appendix A.4.2) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 29.2.1.2).

27.7.2 VM-Exit Controls for MSRs

A VMM may specify lists of MSRs to be stored and loaded on VM exits. The following VM-exit control fields determine how MSRs are stored on VM exits:

- **VM-exit MSR-store count** (32 bits). This field specifies the number of MSRs to be stored on VM exit. It is recommended that this count not exceed 512.¹ Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.
- **VM-exit MSR-store address** (64 bits). This field contains the physical address of the VM-exit MSR-store area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-store count. The format of each entry is given in Table 27-16. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 27-16. Format of an MSR Entry

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

See Section 30.4 for how this area is used on VM exits.

The following VM-exit control fields determine how MSRs are loaded on VM exits:

- **VM-exit MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM exit. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.²

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).

- **VM-exit MSR-load address** (64 bits). This field contains the physical address of the VM-exit MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-load count (see Table 27-16). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 30.6 for how this area is used on VM exits.

27.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Sections 27.8.1 through 27.8.3.

27.8.1 VM-Entry Controls

The **VM-entry controls** constitute a 32-bit vector that governs the basic operation of VM entries. Table 27-17 lists the controls supported. See Chapter 27 for how these controls affect VM entries.

Table 27-17. Definitions of VM-Entry Controls

Bit Position(s)	Name	Description
2	Load debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are loaded on VM entry. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	IA-32e mode guest	On processors that support Intel 64 architecture, this control determines whether the logical processor is in IA-32e mode after VM entry. Its value is loaded into IA32_EFER.LMA as part of VM entry. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
10	Entry to SMM	This control determines whether the logical processor is in system-management mode (SMM) after VM entry. This control must be 0 for any VM entry from outside SMM.
11	Deactivate dual-monitor treatment	If set to 1, the default treatment of SMI and SMM is in effect after the VM entry (see Section 34.15.7). This control must be 0 for any VM entry from outside SMM.
13	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM entry.
14	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM entry.
15	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM entry.
16	Load IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is loaded on VM entry.
17	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM entry or a VMCS packet on a VM entry that returns from SMM (see Chapter 36).
18	Load IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is loaded on VM entry.
19	Load UINV	This control determines whether UINV is loaded on VM entry.
20	Load CET state	This control determines whether CET-related MSRs and SSP are loaded on VM entry.
21	Load guest IA32_LBR_CTL	This control determines whether the IA32_LBR_CTL MSR is loaded on VM entry.
22	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded on VM entry.
23	Load FRED	This control determines whether the FRED MSRs are loaded on VM entry.
25	Allow SEAM-guest telemetry	If set to 1, core out-of-band telemetry can be enabled upon VM entry to SEAM non-root operation. See Section 20.21.1.

NOTES:

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control. If it is read as 1, every VM exit stores the value of IA32_EFER.LMA into the “IA-32e mode guest” VM-entry control (see Section 30.2).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_ENTRY_CTLS and IA32_VMX_TRUE_ENTRY_CTLS (see Appendix A.5) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 29.2.1.3).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8 and 12. The VMX capability MSR IA32_VMX_ENTRY_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_ENTRY_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

27.8.2 VM-Entry Controls for MSRs

A VMM may specify a list of MSRs to be loaded on VM entries. The following VM-entry control fields manage this functionality:

- **VM-entry MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM entry. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM entry.¹
- **VM-entry MSR-load address** (64 bits). This field contains the physical address of the VM-entry MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-entry MSR-load count. The format of entries is described in Table 27-16. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 29.4 for details of how this area is used on VM entries.

27.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event (after all guest state and MSRs have been loaded). This process is called **event injection** and is controlled by the following three VM-entry control fields:

- **Injected-event identification field** (32 bits). This field provides details about the event to be injected. Table 27-18 describes the field.²

Table 27-18. Format of the Injected-Event Identification Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Event type: ¹ 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception (e.g., #PF) 4: Software interrupt (INT <i>n</i>) 5: Privileged software exception (INT1) 6: Software exception (INT3 or INTO) 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)

1. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

2. Older versions of this document called this field VM-entry interruption information.

Table 27-18. Format of the Injected-Event Identification Field (Contd.)

Bit Position(s)	Content
12	Reserved
13	Nested exception (may be set only if IA32_VMX_BASIC[58] = 1)
30:14	Reserved
31	Valid

NOTES:

1. Older versions of this document called this sub-field interruption type.

- The **vector** (bits 7:0) identifies the interrupt or exception or which other event is injected.
- The **event type** (bits 10:8) determines details of how the injection is performed. In general, a VMM should use the type hardware exception for all exceptions **other than** the following:
 - breakpoint exceptions (#BP; a VMM should use the type software exception);
 - overflow exceptions (#OF a VMM should use the use type software exception); and
 - those debug exceptions (#DB) that are generated by INT1 (a VMM should use the use type privileged software exception).¹

The type **other event** is used for injection of events that are not delivered through the IDT.²

- For exceptions, the **deliver-error-code bit** (bit 11) determines whether delivery pushes an error code on the guest stack.
- For exceptions, the **nested-exception bit** (bit 13) indicates that VM entry should treat the injected event as an exception nested on delivery of another event. This bit may be set only if IA32_VMX_BASIC[58] = 1.
- VM entry injects an event if and only if the **valid bit** (bit 31) is 1. The valid bit in this field is cleared on every VM exit (see Section 30.2).
- **Injected-event exception error code** (32 bits). This field is used if and only if the valid bit (bit 31) and the deliver-error-code bit (bit 11) are both set in the injected-event identification field.³
- **Injected-event data** (64 bits). This field is used if and only if the valid bit is set in the injected-event identification field and FRED transitions will be enabled following VM entry.
- **VM-entry instruction length** (32 bits). For injection of events whose type is software interrupt, software exception, or privileged software exception, this field is used to determine the value of RIP that is pushed on the stack. It is used also for injection of SYSCALL and SYSEXIT when FRED transitions would be enabled.

See Section 29.6 for details regarding the mechanics of event injection, including the use of the event type, the nested-exception bit, and the VM-entry instruction length.

VM exits clear the valid bit (bit 31) in the injected-event identification field.

27.9 VM-EXIT INFORMATION FIELDS

The VMCS contains a section of fields that contain information about the most recent VM exit.

On some processors, attempts to write to these fields with VMWRITE fail (see “VMWRITE—Write Field to Virtual-Machine Control Structure” in Chapter 32).⁴

-
1. The type hardware exception should be used for all other debug exceptions.
 2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with values 1 or 3 for *n*.
 3. Older versions of this document called this field VM-entry exception error code.
 4. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

27.9.1 Basic VM-Exit Information

The following VM-exit information fields provide basic information about a VM exit:

- **Exit reason** (32 bits). This field encodes the reason for the VM exit and has the structure given in Table 27-19.

Table 27-19. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason.
16	Always cleared to 0.
24:17	Not currently defined.
25	A VM exit saves this bit as 1 to indicate that the VM exit caused a shadow stack to become prematurely busy.
26	A VM exit saves this bit as 1 to indicate that the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1.
27	A VM exit saves this bit as 1 to indicate that the VM exit was incident to enclave mode.
28	Pending MTF VM exit.
29	VM exit from VMX root operation.
30	Not currently defined.
31	VM-entry failure (0 = true VM exit; 1 = VM-entry failure)

- Bits 15:0 provide basic information about the cause of the VM exit (if bit 31 is clear) or of the VM-entry failure (if bit 31 is set). Appendix C enumerates the basic exit reasons.
- Bit 16 is always cleared to 0.
- Bit 25 is set to 1 if the “prematurely busy shadow stack” VM-exit control is 1 and the VM exit caused a shadow stack to become prematurely busy (see Section 28.4.3). Otherwise, the bit is cleared.
- Bit 26 is set to 1 if the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1. Such VM exits include those that occur due to the 1-setting of that control as well as others that might occur during execution of an instruction that asserted a bus lock.
- Bit 27 is set to 1 if the VM exit occurred while the logical processor was in enclave mode. See Section 30.2.1 for details.
- Bit 28 is set only by an SMM VM exit (see Section 34.15.2) that took priority over an MTF VM exit (see Section 28.5.2) that would have occurred had the SMM VM exit not occurred. See Section 34.15.2.3.
- Bit 29 is set if and only if the processor was in VMX root operation at the time the VM exit occurred. This can happen only for SMM VM exits. See Section 34.15.2.
- Because some VM-entry failures load processor state from the host-state area (see Section 29.8), software must be able to distinguish such cases from true VM exits. Bit 31 is used for that purpose.
- **Exit qualification** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field contains additional information about the cause of VM exits due to the following: debug exceptions; page-fault exceptions; start-up IPIs (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 30.2.1 for details.
- **Guest-linear address** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is used in the following cases:
 - VM exits due to attempts to execute LMSW with a memory operand.
 - VM exits due to attempts to execute INS or OUTS.

- VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions.
- Certain VM exits due to EPT violations

See Section 30.2.1 and Section 34.15.2.3 for details of when and how this field is used.

- **Guest-physical address** (64 bits). This field is used by VM exits due to EPT violations and EPT misconfigurations. See Section 30.2.1 for details of when and how this field is used.

27.9.2 Information for VM Exits Due to Vectored Events

Event-specific information is provided for VM exits due to the following vectored events: exceptions (including those generated by the instructions INT3, INTO, INT1, BOUND, UD0, UD1, UD2, and UDB); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs). This information is provided in the following fields:

- **Exiting-event identification** (32 bits). This field receives basic information associated with the event causing the VM exit. Table 27-20 describes this field.¹

Table 27-20. Format of the Exiting-Event Identification Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Event type: ¹ 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Not used 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	NMI unblocking due to IRET
13	Nested exception (during FRED event delivery)
30:14	Not currently defined
31	Valid

NOTES:

1. Older versions of this document called this sub-field interruption type.

- **Exiting-event error code** (32 bits). For VM exits caused by hardware exceptions that would have delivered an error code on the stack, this field receives that error code.²

Section 30.2.2 provides details of how these fields are saved on VM exits.

27.9.3 Information for VM Exits That Occur During Event Delivery

Additional information is provided for VM exits that occur during event delivery in VMX non-root operation.³ This information is provided in the following fields:

-
1. Older versions of this document called this field VM-exit interruption information.
 2. Older versions of this document called this field VM-exit interruption error code.
 3. This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 29.6.1.2.

- **Original-event identification** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 27-21 describes this field.¹

Table 27-21. Format of the Original-Event Identification Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Event type: ¹ 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	Not currently defined
13	Exception nested on FRED event delivery
30:14	Not currently defined
31	Valid

NOTES:

1. Older versions of this document called this sub-field interruption type.

- **Original-event error code** (32 bits). For VM exits that occur during delivery of hardware exceptions that would have delivered an error code on the stack, this field receives that error code.²
- **Original-event data** (64 bits). For VM exits that occur during FRED event delivery, this field receives the event data that that delivery would have saved on the stack.

See Section 30.2.4 provides details of how these fields are saved on VM exits.

27.9.4 Information for VM Exits Due to Instruction Execution

The following fields are used for VM exits caused by attempts to execute certain instructions in VMX non-root operation:

- **VM-exit instruction length** (32 bits). For VM exits resulting from instruction execution, this field receives the length in bytes of the instruction whose execution led to the VM exit.³ See Section 30.2.5 for details of when and how this field is used.
- **VM-exit instruction information** (32 bits). This field is used for VM exits due to attempts to execute `INS`, `INVEPT`, `INVVPID`, `LIDT`, `LGDT`, `LLDT`, `LTR`, `OUTS`, `SIDT`, `SGDT`, `SLDT`, `STR`, `VMCLEAR`, `VMPTRLD`, `VMPTRST`, `VMREAD`, `VMWRITE`, or `VMXON`.⁴ The format of the field depends on the cause of the VM exit. See Section 30.2.5 for details.

The following fields (64 bits each; 32 bits on processors that do not support Intel 64 architecture) are used only for VM exits due to SMIs that arrive immediately after retirement of I/O instructions. They provide information about that I/O instruction:

1. Older versions of this document called this field IDT-vectoring information.
2. Older versions of this document called this field IDT-vectoring error code.
3. This field is also used for VM exits that occur during the delivery of a software interrupt, software exception, and (if FRED transitions are enabled), `SYSCALL` and `SYSENTER`.
4. 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).

- **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).

An execution of WRMSRLIST causes a VM exit if it writes to an MSR that cannot be written due to the contents of the MSR bitmaps (see Section 28.1.3). Such VM exits save the data that would have been written to the MSR in a 64-bit field called **MSR data**. This field is supported only on processors that support the 1-setting of the “enable MSR-list instructions” VM-execution control.

27.9.5 VM-Instruction Error Field

The 32-bit **VM-instruction error field** does not provide information about the most recent VM exit. In fact, it is not modified on VM exits. Instead, it provides information about errors encountered by a non-faulting execution of one of the VMX instructions.

27.10 VMCS TYPES: ORDINARY AND SHADOW

Every VMCS is either an **ordinary VMCS** or a **shadow VMCS**. A VMCS’s type is determined by the shadow-VMCS indicator in the VMCS region (this is the value of bit 31 of the first 4 bytes of the VMCS region; see Table 27-1): 0 indicates an ordinary VMCS, while 1 indicates a shadow VMCS. Shadow VMCSs are supported only on processors that support the 1-setting of the “VMCS shadowing” VM-execution control (see Section 27.6.2).

A shadow VMCS differs from an ordinary VMCS in two ways:

- An ordinary VMCS can be used for VM entry but a shadow VMCS cannot. Attempts to perform VM entry when the current VMCS is a shadow VMCS fail (see Section 29.1).
- The VMREAD and VMWRITE instructions can be used in VMX non-root operation to access a shadow VMCS but not an ordinary VMCS. This fact results from the following:
 - If the “VMCS shadowing” VM-execution control is 0, execution of the VMREAD and VMWRITE instructions in VMX non-root operation always cause VM exits (see Section 28.1.3).
 - If the “VMCS shadowing” VM-execution control is 1, execution of the VMREAD and VMWRITE instructions in VMX non-root operation can access the VMCS referenced by the VMCS link pointer (see Section 33.3).
 - If the “VMCS shadowing” VM-execution control is 1, VM entry ensures that any VMCS referenced by the VMCS link pointer is a shadow VMCS (see Section 29.3.1.5).

In VMX root operation, both types of VMCSs can be accessed with the VMREAD and VMWRITE instructions.

Software should not modify the shadow-VMCS indicator in the VMCS region of a VMCS that is active. Doing so may cause the VMCS to become corrupted (see Section 27.11.1). Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

27.11 SOFTWARE USE OF THE VMCS AND RELATED STRUCTURES

This section details guidelines that software should observe when using a VMCS and related structures. It also provides descriptions of consequences for failing to follow guidelines.

27.11.1 Software Use of Virtual-Machine Control Structures

To ensure proper processor behavior, software should observe certain guidelines when using an active VMCS.

No VMCS should ever be active on more than one logical processor. If a VMCS is to be “migrated” from one logical processor to another, the first logical processor should execute VMCLEAR for the VMCS (to make it inactive on that logical processor and to ensure that all VMCS data are in memory) before the other logical processor executes

VMPTRLD for the VMCS (to make it active on the second logical processor).¹ A VMCS that is made active on more than one logical processor may become **corrupted** (see below).

Software should not modify the shadow-VMCS indicator (see Table 27-1) in the VMCS region of a VMCS that is active. Doing so may cause the VMCS to become corrupted. Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

Software should use the VMREAD and VMWRITE instructions to access the different fields in the current VMCS (see Section 27.11.2). Software should never access or modify the VMCS data of an active VMCS using ordinary memory operations, in part because the format used to store the VMCS data is implementation-specific and not architecturally defined, and also because a logical processor may maintain some VMCS data of an active VMCS on the processor and not in the VMCS region. The following items detail some of the hazards of accessing VMCS data using ordinary memory operations:

- Any data read from a VMCS with an ordinary memory read does not reliably reflect the state of the VMCS. Results may vary from time to time or from logical processor to logical processor.
- Writing to a VMCS with an ordinary memory write is not guaranteed to have a deterministic effect on the VMCS. Doing so may cause the VMCS to become corrupted (see below).

(Software can avoid these hazards by removing any linear-address mappings to a VMCS region before executing a VMPTRLD for that region and by not remapping it until after executing VMCLEAR for that region.)

If a logical processor leaves VMX operation, any VMCSs active on that logical processor may be corrupted (see below). To prevent such corruption of a VMCS that may be used either after a return to VMX operation or on another logical processor, software should execute VMCLEAR for that VMCS before executing the VMXOFF instruction or removing power from the processor (e.g., as part of a transition to the S3 and S4 power states).

This section has identified operations that may cause a VMCS to become corrupted. These operations may cause the VMCS's data to become undefined. Behavior may be unpredictable if that VMCS used subsequently on any logical processor. The following items detail some hazards of VMCS corruption:

- VM entries may fail for unexplained reasons or may load undesired processor state.
- The processor may not correctly support VMX non-root operation as documented in Chapter 27 and may generate unexpected VM exits.
- VM exits may load undesired processor state, save incorrect state into the VMCS, or cause the logical processor to transition to a shutdown state.

27.11.2 VMREAD, VMWRITE, and Encodings of VMCS Fields

Every field of the VMCS is associated with a 32-bit value that is its **encoding**. The encoding is provided in an operand to VMREAD and VMWRITE when software wishes to read or write that field. These instructions fail if given, in 64-bit mode, an operand that sets an encoding bit beyond bit 32. See Chapter 32 for a description of these instructions.

The structure of the 32-bit encodings of the VMCS components is determined principally by the width of the fields and their function in the VMCS. See Table 27-22.

Table 27-22. Structure of VMCS Component Encoding

Bit Position(s)	Contents
0	Access type (0 = full; 1 = high); must be full for 16-bit, 32-bit, and natural-width fields
9:1	Index

1. As noted in Section 27.1, execution of the VMPTRLD instruction makes a VMCS active. In addition, VM entry makes active any shadow VMCS referenced by the VMCS link pointer in the current VMCS. If a shadow VMCS is made active by VM entry, it is necessary to execute VMCLEAR for that VMCS before allowing that VMCS to become active on another logical processor.

Table 27-22. Structure of VMCS Component Encoding (Contd.)

Bit Position(s)	Contents
11:10	Type: 0: control 1: VM-exit information 2: guest state 3: host state
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).

27.11.3 Initializing a VMCS

Software should initialize fields in a VMCS (using VMWRITE) before using the VMCS for VM entry. Failure to do so may result in unpredictable behavior; for example, a VM entry may fail for unexplained reasons, or a successful transition (VM entry or VM exit) may load processor state with unexpected values.

It is not necessary to initialize fields that the logical processor will not use. (For example, it is not necessary to initialize the MSR-bitmap address if the “use MSR bitmaps” VM-execution control is 0.)

A processor maintains some VMCS information that cannot be modified with the VMWRITE instruction; this includes a VMCS’s launch state (see Section 27.1). Such information may be stored in the VMCS data portion of a VMCS region. Because the format of this information is implementation-specific, there is no way for software to know, when it first allocates a region of memory for use as a VMCS region, how the processor will determine this information from the contents of the memory region.

In addition to its other functions, the VMCLEAR instruction initializes any implementation-specific information in the VMCS region referenced by its operand. To avoid the uncertainties of implementation-specific behavior, software should execute VMCLEAR on a VMCS region before making the corresponding VMCS active with VMPTRLD for the first time. (Figure 27-1 illustrates how execution of VMCLEAR puts a VMCS into a well-defined state.)

The following software usage is consistent with these limitations:

- VMCLEAR should be executed for a VMCS before it is used for VM entry for the first time.
- VMLAUNCH should be used for the first VM entry using a VMCS after VMCLEAR has been executed for that VMCS.
- VMRESUME should be used for any subsequent VM entry using a VMCS (until the next execution of VMCLEAR for the VMCS).

It is expected that, in general, VMRESUME will have lower latency than VMLAUNCH. Since “migrating” a VMCS from one logical processor to another requires use of VMCLEAR (see Section 27.11.1), which sets the launch state of the VMCS to “clear”, such migration requires the next VM entry to be performed using VMLAUNCH. Software developers can avoid the performance cost of increased VM-entry latency by avoiding unnecessary migration of a VMCS from one logical processor to another.

27.11.4 Software Access to Related Structures

In addition to data in the VMCS region itself, VMX non-root operation can be controlled by data structures that are referenced by pointers in a VMCS (for example, the I/O bitmaps). While the pointers to these data structures are

parts of the VMCS, the data structures themselves are not. They are not accessible using VMREAD and VMWRITE but by ordinary memory writes.

Software should ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 27.11.1). Exceptions are made for the following data structures (subject to detailed discussion in the sections indicated): EPT paging structures and the data structures used to locate SPP vectors (Section 31.4.3); the virtual-APIC page (Section 32.1); the posted interrupt descriptor (Section 32.6); and the virtualization-exception information area (Section 28.5.8.2).

27.11.5 VMXON Region

Before executing VMXON, software allocates a region of memory (called the VMXON region)¹ that the logical processor uses to support VMX operation. The physical address of this region (the VMXON pointer) is provided in an operand to VMXON. The VMXON pointer is subject to the limitations that apply to VMCS pointers:

- The VMXON pointer must be 4-KByte aligned (bits 11:0 must be zero).
- The VMXON pointer must not set any bits beyond the processor's physical-address width.^{2,3}

Before executing VMXON, software should write the VMCS revision identifier (see Section 27.2) to the VMXON region. (Specifically, it should write the 31-bit VMCS revision identifier to bits 30:0 of the first 4 bytes of the VMXON region; bit 31 should be cleared to 0.) It need not initialize the VMXON region in any other way. Software should use a separate region for each logical processor and should not access or modify the VMXON region of a logical processor between execution of VMXON and VMXOFF on that logical processor. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 27.11.1).

-
1. The amount of memory required for the VMXON region is the same as that required for a VMCS region. This size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).
 2. CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. It is used to determine the value of and MAXPHYADDR, which constrains the VMXON pointer. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36]. (This is different in secure arbitration mode — SEAM — but VMXON cannot be used in SEAM.) IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode SEAM-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for KeyIDs generally.
 3. If IA32_VMX_BASIC[48] is read as 1, the VMXON pointer must not set any bits in the range 63:32; see Appendix A.1.

12. Updates to Chapter 28, Volume 3B

Change bars and violet 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:

- Inserted Section 28.5.5, "Translation of Guest-Physical Addresses Used by Architectural PEBS," was inserted and subsequent sections were re-numbered.
- PEBS was added to Section 28.5.8.1, "Convertible EPT Violations."

In a virtualized environment using VMX, the guest software stack typically runs on a logical processor in VMX non-root operation. This mode of operation is similar to that of ordinary processor operation outside of the virtualized environment. This chapter describes the differences between VMX non-root operation and ordinary processor operation with special attention to causes of VM exits (which bring a logical processor from VMX non-root operation to root operation). The differences between VMX non-root operation and ordinary processor operation are described in the following sections:

- Section 28.1, “Instructions That Cause VM Exits.”
- Section 28.2, “Other Causes of VM Exits.”
- Section 28.3, “Changes to Instruction Behavior in VMX Non-Root Operation.”
- Section 28.4, “Other Changes in VMX Non-Root Operation.”
- Section 28.5, “Features Specific to VMX Non-Root Operation.”
- Section 28.6, “Unrestricted Guests.”

Chapter 28, “VMX Non-Root Operation,” describes the data control structures that govern VMX non-root operation. Chapter 28, “VMX Non-Root Operation,” describes the operation of VM entries by which the processor transitions from VMX root operation to VMX non-root operation. Chapter 28, “VMX Non-Root Operation,” describes the operation of VM exits by which the processor transitions from VMX non-root operation to VMX root operation.

Chapter 31, “VMX Support for Address Translation,” describes two features that support address translation in VMX non-root operation. Chapter 32, “APIC Virtualization and Virtual Interrupts,” describes features that support virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC) in VMX non-root operation.

28.1 INSTRUCTIONS THAT CAUSE VM EXITS

Certain instructions may cause VM exits if executed in VMX non-root operation. Unless otherwise specified, such VM exits are “fault-like,” meaning that the instruction causing the VM exit does not execute and no processor state is updated by the instruction. Section 30.1 details architectural state in the context of a VM exit.

Section 28.1.1 defines the prioritization between faults and VM exits for instructions subject to both. Section 28.1.2 identifies instructions that cause VM exits whenever they are executed in VMX non-root operation (and thus can never be executed in VMX non-root operation). Section 28.1.3 identifies instructions that cause VM exits depending on the settings of certain VM-execution control fields (see Section 27.6).

28.1.1 Relative Priority of Faults and VM Exits

The following principles describe the ordering between existing faults and VM exits:

- Certain exceptions have priority over VM exits. These include invalid-opcode exceptions, faults based on privilege level,¹ and general-protection exceptions that are based on checking I/O permission bits in the task-state segment (TSS). For example, execution of RDMSR with CPL = 3 generates a general-protection exception and not a VM exit.²
- Faults incurred while fetching instruction operands have priority over VM exits that are conditioned based on the contents of those operands (see LMSW in Section 28.1.3).
- VM exits caused by execution of the INS and OUTS instructions (resulting either because the “unconditional I/O exiting” VM-execution control is 1 or because the “use I/O bitmaps control is 1”) have priority over the following faults:

1. These include faults generated by attempts to execute, in virtual-8086 mode, privileged instructions that are not recognized in that mode.
2. MOV DR is an exception to this rule; see Section 28.1.3.

- A general-protection fault due to the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) being unusable
- A general-protection fault due to an offset beyond the limit of the relevant segment
- An alignment-check exception
- Fault-like VM exits have priority over exceptions other than those mentioned above. For example, RDMSR of a non-existent MSR with CPL = 0 generates a VM exit and not a general-protection exception.

When Section 28.1.2 or Section 28.1.3 (below) identify an instruction execution that may lead to a VM exit, it is assumed that the instruction does not incur a fault that takes priority over a VM exit.

28.1.2 Instructions That Cause VM Exits Unconditionally

The following instructions cause VM exits when they are executed in VMX non-root operation: CPUID, GETSEC,¹ INVD, and XSETBV. This is also true of instructions introduced with VMX: INVEPT, INVVPID, SEAMCALL, TDCALL, VMCALL,² VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMRESUME, VMXOFF, and VMXON.

28.1.3 Instructions That Cause VM Exits Conditionally

Certain instructions cause VM exits in VMX non-root operation depending on the setting of the VM-execution controls. The following instructions can cause “fault-like” VM exits based on the conditions described:³

- **CLTS.** The CLTS instruction causes a VM exit if the bits in position 3 (corresponding to CR0.TS) are set in both the CR0 guest/host mask and the CR0 read shadow.
- **ENCLS.** The ENCLS instruction causes a VM exit if the “enable ENCLS exiting” VM-execution control is 1 and one of the following is true:
 - The value of EAX is less than 63 and the corresponding bit in the ENCLS-exiting bitmap is 1 (see Section 27.6.16).
 - The value of EAX is greater than or equal to 63 and bit 63 in the ENCLS-exiting bitmap is 1.
- **ENQCMD, ENQCMDS.** The behavior of each of these instructions is determined by the setting of the “PASID translation” VM-execution control. If that control is 0, the instruction executes normally. If the control is 1, instruction behavior is modified and may cause a VM exit. See Section 28.5.9.
- **HLT.** The HLT instruction causes a VM exit if the “HLT exiting” VM-execution control is 1.
- **IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD.** The behavior of each of these instructions is determined by the settings of the “unconditional I/O exiting” and “use I/O bitmaps” VM-execution controls:
 - If both controls are 0, the instruction executes normally.
 - If the “unconditional I/O exiting” VM-execution control is 1 and the “use I/O bitmaps” VM-execution control is 0, the instruction causes a VM exit.
 - If the “use I/O bitmaps” VM-execution control is 1, the instruction causes a VM exit if it attempts to access an I/O port corresponding to a bit set to 1 in the appropriate I/O bitmap (see Section 27.6.4). If an I/O operation “wraps around” the 16-bit I/O-port space (accesses ports FFFFH and 0000H), the I/O instruction causes a VM exit (the “unconditional I/O exiting” VM-execution control is ignored if the “use I/O bitmaps” VM-execution control is 1).

1. An execution of GETSEC in VMX non-root operation causes a VM exit if CR4.SMXE[Bit 14] = 1 regardless of the value of CPL or RAX. An execution of GETSEC causes an invalid-opcode exception (#UD) if CR4.SMXE[Bit 14] = 0.

2. Under the dual-monitor treatment of SMIs and SMM, executions of VMCALL cause SMM VM exits in VMX root operation outside SMM. See Section 34.15.2.

3. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 27.6.2.

See Section 28.1.1 for information regarding the priority of VM exits relative to faults that may be caused by the INS and OUTS instructions.

- **INVLPG.** The INVLPG instruction causes a VM exit if the “INVLPG exiting” VM-execution control is 1.
- **INVPCID.** The INVPCID instruction causes a VM exit if the “INVLPG exiting” and “enable INVPCID” VM-execution controls are both 1.
- **LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR.** These instructions cause VM exits if the “descriptor-table exiting” VM-execution control is 1.
- **LMSW.** In general, the LMSW instruction causes a VM exit if it would write, for any bit set in the low 4 bits of the CR0 guest/host mask, a value different than the corresponding bit in the CR0 read shadow. LMSW never clears bit 0 of CR0 (CR0.PE); thus, LMSW causes a VM exit if either of the following are true:
 - The bits in position 0 (corresponding to CR0.PE) are set in both the CR0 guest/host mask and the source operand, and the bit in position 0 is clear in the CR0 read shadow.
 - For any bit position in the range 3:1, the bit in that position is set in the CR0 guest/host mask and the values of the corresponding bits in the source operand and the CR0 read shadow differ.
- **LOADIWKEY.** The LOADIWKEY instruction causes a VM exit if the “LOADIWKEY exiting” VM-execution control is 1.
- **MONITOR.** The MONITOR instruction causes a VM exit if the “MONITOR exiting” VM-execution control is 1.
- **MOV from CR3.** The MOV from CR3 instruction causes a VM exit if the “CR3-store exiting” VM-execution control is 1. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
- **MOV from CR8.** The MOV from CR8 instruction causes a VM exit if the “CR8-store exiting” VM-execution control is 1.
- **MOV to CR0.** The MOV to CR0 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR0 guest/host mask, the corresponding bit in the CR0 read shadow. (If every bit is clear in the CR0 guest/host mask, MOV to CR0 cannot cause a VM exit.)
- **MOV to CR3.** The MOV to CR3 instruction causes a VM exit unless the “CR3-load exiting” VM-execution control is 0 or the value of its source operand is equal to one of the CR3-target values specified in the VMCS. Only the first n CR3-target values are considered, where n is the CR3-target count. If the “CR3-load exiting” VM-execution control is 1 and the CR3-target count is 0, MOV to CR3 always causes a VM exit.

The first processors to support the virtual-machine extensions supported only the 1-setting of the “CR3-load exiting” VM-execution control. These processors always consult the CR3-target controls to determine whether an execution of MOV to CR3 causes a VM exit.

- **MOV to CR4.** The MOV to CR4 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR4 guest/host mask, the corresponding bit in the CR4 read shadow.
- **MOV to CR8.** The MOV to CR8 instruction causes a VM exit if the “CR8-load exiting” VM-execution control is 1.
- **MOV DR.** The MOV DR instruction causes a VM exit if the “MOV-DR exiting” VM-execution control is 1. Such VM exits represent an exception to the principles identified in Section 28.1.1 in that they take priority over the following: general-protection exceptions based on privilege level; and invalid-opcode exceptions that occur because CR4.DE=1 and the instruction specified access to DR4 or DR5.
- **MWAIT.** The MWAIT instruction causes a VM exit if the “MWAIT exiting” VM-execution control is 1. If this control is 0, the behavior of the MWAIT instruction may be modified (see Section 28.3).
- **PAUSE.** The behavior of each of this instruction depends on CPL and the settings of the “PAUSE exiting” and “PAUSE-loop exiting” VM-execution controls:
 - CPL = 0.
 - If the “PAUSE exiting” and “PAUSE-loop exiting” VM-execution controls are both 0, the PAUSE instruction executes normally.
 - If the “PAUSE exiting” VM-execution control is 1, the PAUSE instruction causes a VM exit (the “PAUSE-loop exiting” VM-execution control is ignored if CPL = 0 and the “PAUSE exiting” VM-execution control is 1).

- If the “PAUSE exiting” VM-execution control is 0 and the “PAUSE-loop exiting” VM-execution control is 1, the following treatment applies.

The processor determines the amount of time between this execution of PAUSE and the previous execution of PAUSE at CPL 0. If this amount of time exceeds the value of the VM-execution control field PLE_Gap, the processor considers this execution to be the first execution of PAUSE in a loop. (It also does so for the first execution of PAUSE at CPL 0 after VM entry.)

Otherwise, the processor determines the amount of time since the most recent execution of PAUSE that was considered to be the first in a loop. If this amount of time exceeds the value of the VM-execution control field PLE_Window, a VM exit occurs.

For purposes of these computations, time is measured based on a counter that runs at the same rate as the timestamp counter (TSC).

— CPL > 0.

- If the “PAUSE exiting” VM-execution control is 0, the PAUSE instruction executes normally.
- If the “PAUSE exiting” VM-execution control is 1, the PAUSE instruction causes a VM exit.

The “PAUSE-loop exiting” VM-execution control is ignored if CPL > 0.

- **PCONFIG.** The PCONFIG instruction causes a VM exit if the “enable PCONFIG” VM-execution control is 1 and one of the following is true:

- The value of EAX is less than 63 and the corresponding bit in the PCONFIG-exiting bitmap is 1 (see Section 27.6.17).
- The value of EAX is greater than or equal to 63 and bit 63 in the PCONFIG-exiting bitmap is 1.

If the “enable PCONFIG” VM-execution control is 1 and neither of the previous items hold, the PCONFIG instruction executes normally.

- **RDMSR.** The RDMSR instruction causes a VM exit if any of the following are true:
 - The “use MSR bitmaps” VM-execution control is 0.
 - The value of ECX is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - The value of ECX is in the range 00000000H – 00001FFFH and bit *n* in read bitmap for low MSRs is 1, where *n* is the value of ECX.
 - The value of ECX is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of ECX & 00001FFFH.

See Section 27.6.9 for details regarding how these bitmaps are identified.

- **RDMSRLIST.** The RDMSRLIST instruction causes a VM exit if the “enable MSR-list instructions” VM-execution control is 1 and the “use MSR bitmaps” VM-execution control is 0. If both controls are 1, the instruction reads one MSR at a time normally, storing the value read to memory and clearing the corresponding bit in RCX. An attempt to read MSR *X* causes a VM exit if any of the following are true:
 - *X* is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - *X* is in the range 00000000H – 00001FFFH and bit *X* in read bitmap for low MSRs is 1.
 - *X* is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of *X* & 00001FFFH.

If an attempt to read an MSR causes a VM exit, the corresponding bit in RCX is not cleared, the MSR is not read, and no value is stored to memory.

- **RDPMC.** The RDPMC instruction causes a VM exit if the “RDPMC exiting” VM-execution control is 1.
- **RDRAND.** The RDRAND instruction causes a VM exit if the “RDRAND exiting” VM-execution control is 1.
- **RDSEED.** The RDSEED instruction causes a VM exit if the “RDSEED exiting” VM-execution control is 1.
- **RDTSC.** The RDTSC instruction causes a VM exit if the “RDTSC exiting” VM-execution control is 1.
- **RDTSCP.** The RDTSCP instruction causes a VM exit if the “RDTSC exiting” and “enable RDTSCP” VM-execution controls are both 1.
- **RSM.** The RSM instruction causes a VM exit if executed in system-management mode (SMM).¹

- **TPAUSE.** The TPAUSE instruction causes a VM exit if the “RDTSC exiting” and “enable user wait and pause” VM-execution controls are both 1.
- **UMWAIT.** The UMWAIT instruction causes a VM exit if the “RDTSC exiting” and “enable user wait and pause” VM-execution controls are both 1.
- **VMREAD.** The VMREAD instruction causes a VM exit if any of the following are true:
 - The “VMCS shadowing” VM-execution control is 0.
 - Bits 63:15 (bits 31:15 outside 64-bit mode) of the register source operand are not all 0.
 - Bit n in VMREAD bitmap is 1, where n is the value of bits 14:0 of the register source operand. See Section 27.6.15 for details regarding how the VMREAD bitmap is identified.

If the VMREAD instruction does not cause a VM exit, it reads from the VMCS referenced by the VMCS link pointer. See Chapter 33, “VMREAD—Read Field from Virtual-Machine Control Structure” for details of the operation of the VMREAD instruction.

- **VMWRITE.** The VMWRITE instruction causes a VM exit if any of the following are true:
 - The “VMCS shadowing” VM-execution control is 0.
 - Bits 63:15 (bits 31:15 outside 64-bit mode) of the register source operand are not all 0.
 - Bit n in VMWRITE bitmap is 1, where n is the value of bits 14:0 of the register source operand. See Section 27.6.15 for details regarding how the VMWRITE bitmap is identified.

If the VMWRITE instruction does not cause a VM exit, it writes to the VMCS referenced by the VMCS link pointer. See Chapter 33, “VMWRITE—Write Field to Virtual-Machine Control Structure” for details of the operation of the VMWRITE instruction.

- **WBINVD.** The WBINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.
- **WBNOINVD.** The WBNOINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.
- **WRMSR, WRMSRNS.** Execution of one of these instructions causes a VM exit if any of the following are true:
 - The “use MSR bitmaps” VM-execution control is 0.
 - The value of ECX is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - The value of ECX is in the range 00000000H – 00001FFFH and bit n in write bitmap for low MSRs is 1, where n is the value of ECX.
 - The value of ECX is in the range C0000000H – C0001FFFH and bit n in write bitmap for high MSRs is 1, where n is the value of ECX & 00001FFFH.

See Section 27.6.9 for details regarding how these bitmaps are identified.

- **WRMSRLIST.** The WRMSRLIST instruction causes a VM exit if the “enable MSR-list instructions” VM-execution control is 1 and the “use MSR bitmaps” VM-execution control is 0. In this case, the register operands of the instructions are not used, memory is not read, and no MSRs are modified.

If both controls are 1, the instruction writes one MSR at a time normally, using a value read from memory and clearing the corresponding bit in RCX. An attempt to write MSR X causes a VM exit if any of the following are true:

- X is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
- X is in the range 00000000H – 00001FFFH and bit X in write bitmap for low MSRs is 1.
- X is in the range C0000000H – C0001FFFH and bit n in write bitmap for high MSRs is 1, where n is the value of X & 00001FFFH.

Such a VM exit occurs after the data that would be written to the MSR is read from memory, but the corresponding bit in RCX is not cleared and the MSR is not written. The data that would have been written to the MSR is saved into the MSR-data field of the VMCS (see Section 30.2.5).

1. Execution of the RSM instruction outside SMM causes an invalid-opcode exception regardless of whether the processor is in VMX operation. It also does so in VMX root operation in SMM; see Section 34.15.3.

- **XRSTORS.** The XRSTORS instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 27.6.20).
- **XSAVES.** The XSAVES instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 27.6.20).

28.2 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:¹

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 27.6.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally. This use of the exception bitmap applies also to exceptions generated by the instructions INT1, INT3, INTO, BOUND, UD0, UD1, and UD2, and UDB.²

Page faults (exceptions with vector 14) are specially treated. When a page fault occurs, a processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault [PFEC]; (3) the page-fault error-code mask field [PFEC_MASK]; and (4) the page-fault error-code match field [PFEC_MATCH]. It checks if PFEC & PFEC_MASK = PFEC_MATCH. If there is equality, the specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

Thus, if software desires VM exits on all page faults, it can set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If software desires VM exits on no page faults, it can set bit 14 in the exception bitmap to 1, the page-fault error-code mask field to 00000000H, and the page-fault error-code match field to FFFFFFFFH.

- **Triple fault.** A VM exit occurs if the logical processor encounters an exception while attempting to call the double-fault handler and that exception itself does not cause a VM exit due to the exception bitmap. This applies to the case in which the double-fault exception was generated within VMX non-root operation, the case in which the double-fault exception was generated during event injection by VM entry, and to the case in which VM entry is injecting a double-fault exception.
- **External interrupts.** An external interrupt causes a VM exit if the “external-interrupt exiting” VM-execution control is 1 (see Section 32.6 for an exception.) Otherwise, the processor handles the interrupt normally.³ (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The processor does handle the interrupt and no VM exit occurs.)
- **Non-maskable interrupts (NMIs).** An NMI causes a VM exit if the “NMI exiting” VM-execution control is 1. Otherwise, it is delivered using vector 2. (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered and no VM exit occurs.)
- **INIT signals.** INIT signals cause VM exits. A logical processor performs none of the operations normally associated with these events. Such exits do not modify register state or clear pending events as they would outside of VMX operation. (If a logical processor is in the wait-for-SIPI state, INIT signals are blocked. They do not cause VM exits in this case.)
- **Start-up IPIs (SIPIs). SIPIs cause VM exits.** If a logical processor is not in the wait-for-SIPI activity state when a SIPI arrives, no VM exit occurs and the SIPI is discarded. VM exits due to SIPIs do not perform any of the normal operations associated with those events: they do not modify register state as they would outside of

1. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 27.6.2.

2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

3. Normal handling usually means delivery to software, but it could also mean treatment of the interrupt as a user-interrupt notification.

VMX operation. (If a logical processor is not in the wait-for-SIPI state, SIPIs are blocked. They do not cause VM exits in this case.)

- **Task switches.** Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. See Section 28.4.2.
- **System-management interrupts (SMIs).** If the logical processor is using the dual-monitor treatment of SMIs and system-management mode (SMM), SMIs cause SMM VM exits. See Section 34.15.2.¹
- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 28.5.1 for details of operation of the VMX-preemption timer.

Debug-trap exceptions and higher priority events take priority over VM exits caused by the VMX-preemption timer. VM exits caused by the VMX-preemption timer take priority over VM exits caused by the “NMI-window exiting” VM-execution control and lower priority events.

These VM exits wake a logical processor from the same inactive states as would a non-maskable interrupt. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

- **Bus locks.** Assertion of a bus lock (see Section 11.1.2) causes a VM exit if the “VMM bus-lock detection” VM-execution control is 1. Such a VM exit is trap-like because it is generated after execution of an instruction that asserts a bus lock. The VM exit thus does not prevent assertion of the bus lock. These VM exits take priority over system-management interrupts (SMIs), INIT signals, and lower priority events.
- **Instruction timeout.** If the “instruction timeout” VM-execution control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within the amount of time specified by the instruction-timeout control VM-execution control field (see Section 27.6.24).

In addition, there are controls that cause VM exits based on the readiness of guest software to receive interrupts:

- If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if RFLAGS.IF = 1 and there is no blocking of events by STI or by MOV SS (see Table 27-3).

Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.

- If the “NMI-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if there is no virtual-NMI blocking and there is no blocking of events by MOV SS and no blocking of events by STI (see Table 27-3).

VM exits caused by the VMX-preemption timer and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an NMI. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

Conditions necessary for some of the VM exits identified in this section may hold immediately after VM entry. If they do, a corresponding VM exit occurs at that time.

28.3 CHANGES TO INSTRUCTION BEHAVIOR IN VMX NON-ROOT OPERATION

The behavior of some instructions is changed in VMX non-root operation. Some of these changes are determined by the settings of certain VM-execution control fields. The following items detail such changes:²

- **CLTS.** Behavior of the CLTS instruction is determined by the bits in position 3 (corresponding to CR0.TS) in the CR0 guest/host mask and the CR0 read shadow:

1. Under the dual-monitor treatment of SMIs and SMM, SMIs also cause SMM VM exits if they occur in VMX root operation outside SMM. If the processor is using the default treatment of SMIs and SMM, SMIs are delivered as described in Section 34.14.1.

- If bit 3 in the CR0 guest/host mask is 0, CLTS clears CR0.TS normally (the value of bit 3 in the CR0 read shadow is irrelevant in this case), unless CR0.TS is fixed to 1 in VMX operation (see Section 26.8), in which case CLTS causes a general-protection exception.
- If bit 3 in the CR0 guest/host mask is 1 and bit 3 in the CR0 read shadow is 0, CLTS completes but does not change the contents of CR0.TS.
- If the bits in position 3 in the CR0 guest/host mask and the CR0 read shadow are both 1, CLTS causes a VM exit.
- **ENQCMD, ENQCMLS.** Each of these instructions performs a 64-byte enqueue store that includes a PASID value in bits 19:0. For ENQCMD, the PASID is normally the value of IA32_PASID[19:0], while for ENQCMLS, the PASID is normally read from memory.

The behavior of each of these instructions (and in particular the PASID value used for the enqueue store) is determined by the setting of the “PASID translation” VM-execution control:

- If the “PASID translation” VM-execution control is 0, the instruction operates normally.
- If the “PASID translation” VM-execution control is 1, the PASID value used for the enqueue store is determined by the PASID-translation process described in Section 28.5.9. (Note the PASID translation may result in a VM exit, in which case the enqueue store is not performed.)

An execution of ENQCMD or ENQCMLS performs PASID translation only after checking for conditions that may result in general-protection exception (the check of IA32_PASID.Valid for ENQCMD; the privilege-level check for ENQCMLS), after loading the instruction's source operand from memory, and thus after any faults or VM exits that the loading may cause (e.g., page faults or EPT violations). PASID translation occurs before the actual enqueue store and thus before any faults or VM exits that it may cause.

- **ERETS, ERETU.** Each of these instructions unblocks NMIs if bit 18 is set in the augmented SS in the return state on the stack. The following items detail how this behavior may be changed in VMX non-root operation, depending on the settings of certain VM-execution controls:
 - If the “NMI exiting” VM-execution control is 0, this behavior of ERETU and ERETU is not modified (they unblock NMIs as indicated above). (If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 29.2.1.1.)
 - If the “NMI exiting” VM-execution control is 1, ERETU and ERETU do not unblock physical NMIs, regardless of the return state on the stack. If, in addition, the “virtual NMIs” VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, ERETU and ERETU each unblocks virtual NMIs if bit 18 is set in the augmented SS in the return state.

If the “NMI exiting” VM-execution control is 1 and the “virtual NMIs” VM-execution control is 0, ERETU and ERETU ignore bit 18 of the augmented SS.

- **INVPCID.** Behavior of the INVPCID instruction is determined first by the setting of the “enable INVPCID” VM-execution control:
 - If the “enable INVPCID” VM-execution control is 0, INVPCID causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable INVPCID” VM-execution control is 1, treatment is based on the setting of the “INVLPG exiting” VM-execution control:
 - If the “INVLPG exiting” VM-execution control is 0, INVPCID operates normally.
 - If the “INVLPG exiting” VM-execution control is 1, INVPCID causes a VM exit.
- **IRET.** Behavior of IRET with regard to NMI blocking (see Table 27-3) is determined by the settings of the “NMI exiting” and “virtual NMIs” VM-execution controls:
 - If the “NMI exiting” VM-execution control is 0, IRET operates normally and unblocks NMIs. (If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 29.2.1.1.)

-
2. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 27.6.2.

- If the “NMI exiting” VM-execution control is 1, IRET does not affect blocking of NMIs. If, in addition, the “virtual NMIs” VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, IRET removes any virtual-NMI blocking.

The unblocking of NMIs or virtual NMIs specified above occurs even if IRET causes a fault.

- **LMSW.** Outside of VMX non-root operation, LMSW loads its source operand into CR0[3:0], but it does not clear CR0.PE if that bit is set. In VMX non-root operation, an execution of LMSW that does not cause a VM exit (see Section 28.1.3) leaves unmodified any bit in CR0[3:0] corresponding to a bit set in the CR0 guest/host mask. An attempt to set any other bit in CR0[3:0] to a value not supported in VMX operation (see Section 26.8) causes a general-protection exception. Attempts to clear CR0.PE are ignored without fault.
- **MOV from CR0.** The behavior of MOV from CR0 is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0 reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow.
Depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.
- **MOV from CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV from CR3 does not cause a VM exit (see Section 28.1.3), the value loaded from CR3 is a guest-physical address; see Section 31.3.1.
- **MOV from CR4.** The behavior of MOV from CR4 is determined by the CR4 guest/host mask and the CR4 read shadow. For each position corresponding to a bit clear in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR4. For each position corresponding to a bit set in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR4 read shadow. Thus, if every bit is cleared in the CR4 guest/host mask, MOV from CR4 reads normally from CR4; if every bit is set in the CR4 guest/host mask, MOV from CR4 returns the value of the CR4 read shadow.
Depending on the contents of the CR4 guest/host mask and the CR4 read shadow, bits may be set in the destination that would never be set when reading directly from CR4.
- **MOV from CR8.** If the MOV from CR8 instruction does not cause a VM exit (see Section 28.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 32.3.
- **MOV to CR0.** An execution of MOV to CR0 that does not cause a VM exit (see Section 28.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask. Treatment of attempts to modify other bits in CR0 depends on the setting of the “unrestricted guest” VM-execution control:
 - If the control is 0, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 to a value not supported in VMX operation (see Section 26.8).
 - If the control is 1, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 other than bit 0 (PE) or bit 31 (PG) to a value not supported in VMX operation. It remains the case, however, that MOV to CR0 causes a general-protection exception if it would result in CR0.PE = 0 and CR0.PG = 1 or if it would result in CR0.PG = 1, CR4.PAE = 0, and IA32_EFER.LME = 1.
- **MOV to CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV to CR3 does not cause a VM exit (see Section 28.1.3), the value loaded into CR3 is treated as a guest-physical address; see Section 31.3.1. The following details apply:
 - An attempt to set a reserved bit in CR3[63:M] results in #GP(0), where M is the value enumerated in CPUID.80000008H:EAX[7:0]. (Unlike other cases, the value M is not reduced outside SEAM.)
 - If PAE paging is not being used, the instruction does not use the guest-physical address to access memory and it does not cause it to be translated through EPT.¹
 - If PAE paging is being used, the instruction translates the guest-physical address through EPT and uses the result to load the four (4) page-directory-pointer-table entries (PDPTes). The instruction does not use the guest-physical addresses the PDPTes to access memory and it does not cause them to be translated

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

through EPT. An attempt to set a PDPTE bit in positions 63:M results in #GP(0), where M is defined above (M is not reduced outside SEAM).

- **MOV to CR4.** An execution of MOV to CR4 that does not cause a VM exit (see Section 28.1.3) leaves unmodified any bit in CR4 corresponding to a bit set in the CR4 guest/host mask. Such an execution causes a general-protection exception if it attempts to set any bit in CR4 (not corresponding to a bit set in the CR4 guest/host mask) to a value not supported in VMX operation (see Section 26.8).
- **MOV to CR8.** If the MOV to CR8 instruction does not cause a VM exit (see Section 28.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 32.3.
- **MWAIT.** Behavior of the MWAIT instruction (which always causes an invalid-opcode exception—#UD—if CPL > 0) is determined by the setting of the “MWAIT exiting” VM-execution control:
 - If the “MWAIT exiting” VM-execution control is 1, MWAIT causes a VM exit.
 - If the “MWAIT exiting” VM-execution control is 0, MWAIT operates normally if one of the following are true: (1) ECX[0] is 0; (2) RFLAGS.IF = 1; or both of the following are true: (a) the “interrupt-window exiting” VM-execution control is 0; and (b) the logical processor has not recognized a pending virtual interrupt (see Section 29.2.1).
 - If the “MWAIT exiting” VM-execution control is 0, ECX[0] = 1, and RFLAGS.IF = 0, MWAIT does not cause the processor to enter an implementation-dependent optimized state if either the “interrupt-window exiting” VM-execution control is 1 or the logical processor has recognized a pending virtual interrupt; instead, control passes to the instruction following the MWAIT instruction.
- **PBNDKB.** Behavior of the PBNDKB instruction is determined by the setting of the “enable PBNDKB” VM-execution control:
 - If the “enable PBNDKB” VM-execution control is 0, PBNDKB causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable PBNDKB” VM-execution control is 1, PBNDKB operates normally.
- **PCONFIG.** Behavior of the PCONFIG instruction is determined by the setting of the “enable PCONFIG” VM-execution control:
 - If the “enable PCONFIG” VM-execution control is 0, PCONFIG causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable PCONFIG” VM-execution control is 1, PCONFIG may cause a VM exit as specified in Section 28.1.3; if it does not cause such a VM exit, it operates normally.
- **RDMSR.** Section 28.1.3 identifies when executions of the RDMSR instruction cause VM exits. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:
 - If ECX contains 10H (indicating the IA32_TIME_STAMP_COUNTER MSR), the value returned by the instruction is determined by the setting of the “use TSC offsetting” VM-execution control:
 - If the control is 0, RDMSR operates normally, loading EAX:EDX with the value of the IA32_TIME_STAMP_COUNTER MSR.
 - If the control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDMSR loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDMSR first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

The 1-setting of the “use TSC offsetting” VM-execution control does not affect executions of RDMSR if ECX contains 6E0H (indicating the IA32_TSC_DEADLINE MSR). Such executions return the APIC-timer deadline relative to the actual timestamp counter without regard to the TSC offset.

 - If ECX contains 48H (indicating the IA32_SPEC_CTRL MSR), the value returned by the instruction is determined by the setting of the “virtualize IA32_SPEC_CTRL” VM-execution control:

- If the control is 0, RDMSR operates normally, loading EAX:EDX with the value of the IA32_SPEC_CTRL MSR.
- If the control is 1, the value returned is that of the IA32_SPEC_CTRL shadow field in the VMCS.
- If ECX is in the range 800H–8FFH (indicating an APIC MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 32.5.
- **RDMSRLIST.** Behavior of the RDMSRLIST instruction is determined first by the setting of the “enable MSR-list instructions” VM-execution control:
 - If the “enable MSR-list instructions” VM-execution control is 0, RDMSRLIST causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable MSR-list instructions” VM-execution control is 1, the instruction causes a general-protection exception (#GP) normally if CPL > 0. Otherwise, its operation depends on the setting of the “use MSR bitmaps” VM-execution control:
 - If the control is 0, the instruction causes a VM exit.
 - If the control is 1, the instruction commences normally, reading one MSR at a time. Reads of certain MSRs are treated specially as described above for RDMSR. In addition, attempts to access specific MSRs may cause VM exits; see Section 28.1.3 for details.
- **RDPID.** Behavior of the RDPID instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:
 - If the “enable RDTSCP” VM-execution control is 0, RDPID causes an invalid-opcode exception (#UD).
 - If the “enable RDTSCP” VM-execution control is 1, RDPID operates normally.
- **RDTS.** Behavior of the RDTS instruction is determined by the settings of the “RDTS exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTS operates normally.
 - If the “RDTS exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDTS loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDTS first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.
 - If the “RDTS exiting” VM-execution control is 1, RDTS causes a VM exit.
- **RDTSCP.** Behavior of the RDTSCP instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:
 - If the “enable RDTSCP” VM-execution control is 0, RDTSCP causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable RDTSCP” VM-execution control is 1, treatment is based on the settings of the “RDTS exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTSCP operates normally.
 - If the “RDTS exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDTSCP loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDTSCP first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

In either case, RDTSCP also loads ECX with the value of bits 31:0 of the IA32_TSC_AUX MSR.

- If the “RDTS exiting” VM-execution control is 1, RDTSCP causes a VM exit.

- **SMSW.** The behavior of SMSW is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, SMSW reads normally from CR0; if every bit is set in the CR0 guest/host mask, SMSW returns the value of the CR0 read shadow.

Note the following: (1) for any memory destination or for a 16-bit register destination, only the low 16 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:16 of a register destination are left unchanged); (2) for a 32-bit register destination, only the low 32 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:32 of the destination are cleared); and (3) depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.

- **TPAUSE.** Behavior of the TPAUSE instruction is determined first by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, TPAUSE causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable user wait and pause” VM-execution control is 1, treatment is based on the setting of the “RDTSC exiting” VM-execution control:
 - If the “RDTSC exiting” VM-execution control is 0, the instruction delays for an amount of time called here the **physical delay**. The physical delay is first computed by determining the **virtual delay** (the time to delay relative to the guest’s timestamp counter).
 If IA32_UWAIT_CONTROL[31:2] is zero, the virtual delay is the value in EDX:EAX minus the value that RDTSC would return (see above); if IA32_UWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UWAIT_CONTROL, FFFFFFFCH).
 The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:
 - If either control is 0, the physical delay is the virtual delay.
 - If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.
 - If the “RDTSC exiting” VM-execution control is 1, TPAUSE causes a VM exit.

- **UMONITOR.** Behavior of the UMONITOR instruction is determined by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, UMONITOR causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable user wait and pause” VM-execution control is 1, UMONITOR operates normally.
- **UWAIT.** Behavior of the UWAIT instruction is determined first by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, UWAIT causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable user wait and pause” VM-execution control is 1, treatment is based on the setting of the “RDTSC exiting” VM-execution control:
 - If the “RDTSC exiting” VM-execution control is 0, and if the instruction causes a delay, the amount of time delayed is called here the **physical delay**. The physical delay is first computed by determining the **virtual delay** (the time to delay relative to the guest’s timestamp counter).
 If IA32_UWAIT_CONTROL[31:2] is zero, the virtual delay is the value in EDX:EAX minus the value that RDTSC would return (see above); if IA32_UWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UWAIT_CONTROL, FFFFFFFCH).
 The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:

- If either control is 0, the physical delay is the virtual delay.
- If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.
- If the “RDTSC exiting” VM-execution control is 1, UMWAIT causes a VM exit.
- **WRMSR, WRMSRNS.** Section 28.1.3 identifies when an execution of WRMSR or WRMSRNS would cause a VM exit. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:
 - If ECX contains 48H (indicating the IA32_SPEC_CTRL MSR), instruction behavior depends on the setting of the “virtualize IA32_SPEC_CTRL” VM-execution control:
 - If the control is 0, WRMSR and WRMSRNS operate normally, loading the IA32_SPEC_CTRL MSR with the value in EAX:EDX.
 - If the control is 1, the instruction will attempt to write the IA32_SPEC_CTRL MSR using the instruction’s source operand, but it will attempt to modify only those bits in positions corresponding to bits cleared in the IA32_SPEC_CTRL mask field in the VMCS.

Specifically, the instruction attempts to write the MSR with the following value:

$$(\text{MSR\_VAL} \& \text{ISC\_MASK}) \text{ OR } (\text{SRC} \& \text{NOT ISC\_MASK}),$$

where MSR_VAL is the original value of the MSR, ISC_MASK is the IA32_SPEC_CTRL mask, and SRC is the instruction’s source operand.

Any fault that would result from writing that value to the MSR (e.g., due to a reserved-bit violation) occurs normally. Otherwise, the value is written to the MSR.

Such a write to the MSR will have any side effects that would occur normally had the MSR been written with the value indicated above (including any side effects that may result from writing unchanged values to the masked bits).

If the write completes without a fault, the unmodified value of the source operand is written to the IA32_SPEC_CTRL shadow field in the VMCS.
 - If ECX contains 79H (indicating IA32_BIOS_UPDT_TRIG MSR), no microcode update is loaded, and control passes to the next instruction. This implies that microcode updates cannot be loaded in VMX non-root operation.
 - On processors that support Intel PT but which do not allow it to be used in VMX operation, if ECX contains 570H (indicating the IA32_RTIT_CTL MSR), the instruction causes a general-protection exception.¹
 - If ECX contains 808H (indicating the TPR MSR), 80BH (the EOI MSR), 830H (the ICR MSR), or 83FH (the self-IPI MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 32.5.
- **WRMSRLIST.** Behavior of the WRMSRLIST instruction is determined first by the setting of the “enable MSR-list instructions” VM-execution control:
 - If the “enable MSR-list instructions” VM-execution control is 0, WRMSRLIST causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable MSR-list instructions” VM-execution control is 1, the instruction causes a general-protection exception (#GP) normally if CPL > 0. Otherwise, its operation depends on the setting of the “use MSR bitmaps” VM-execution control:
 - If the control is 0, the instruction causes a VM exit.
 - If the control is 1, the instruction commences normally, writing one MSR at a time. Writes to certain MSRs are treated specially as described above for WRMSR and WRMSRNS. In addition, attempts to access specific MSRs may cause VM exits; see Section 28.1.3 for details.

1. Software should read the VMX capability MSR IA32_VMX_MISC to determine whether the processor allows Intel PT to be used in VMX operation (see Appendix A.6).

- **XRSTORS.** Behavior of the XRSTORS instruction is determined first by the setting of the “enable XSAVES/XRSTORS” VM-execution control:
 - If the “enable XSAVES/XRSTORS” VM-execution control is 0, XRSTORS causes an invalid-opcode exception (#UD).
 - If the “enable XSAVES/XRSTORS” VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 27.6.20):
 - XRSTORS causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XRSTORS operates normally.
- **XSAVES.** Behavior of the XSAVES instruction is determined first by the setting of the “enable XSAVES/XRSTORS” VM-execution control:
 - If the “enable XSAVES/XRSTORS” VM-execution control is 0, XSAVES causes an invalid-opcode exception (#UD).
 - If the “enable XSAVES/XRSTORS” VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 27.6.20):
 - XSAVES causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XSAVES operates normally.

28.4 OTHER CHANGES IN VMX NON-ROOT OPERATION

Treatments of event blocking, task switches, and certain shadow-stack updates may differ in VMX non-root operation as described in the following sections.

28.4.1 Event Blocking

Event blocking is modified in VMX non-root operation as follows:

- If the “external-interrupt exiting” VM-execution control is 1, RFLAGS.IF does not control the blocking of external interrupts. In this case, an external interrupt that is not blocked for other reasons causes a VM exit (even if RFLAGS.IF = 0).
- If the “external-interrupt exiting” VM-execution control is 1, external interrupts may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).
- If the “NMI exiting” VM-execution control is 1, non-maskable interrupts (NMIs) may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).

28.4.2 Treatment of Task Switches

Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. However, the following checks are performed (in the order indicated), possibly resulting in a fault, before there is any possibility of a VM exit due to task switch:

1. If a task gate is being used, appropriate checks are made on its P bit and on the proper values of the relevant privilege fields. The following cases detail the privilege checks performed:
 - a. If CALL, INT *n*, INT1, INT3, INTO, or JMP accesses a task gate in IA-32e mode, a general-protection exception occurs.
 - b. If CALL, INT *n*, INT3, INTO, or JMP accesses a task gate outside IA-32e mode, privilege-levels checks are performed on the task gate but, if they pass, privilege levels are not checked on the referenced task-state segment (TSS) descriptor.
 - c. If CALL or JMP accesses a TSS descriptor directly in IA-32e mode, a general-protection exception occurs.

- d. If CALL or JMP accesses a TSS descriptor directly outside IA-32e mode, privilege levels are checked on the TSS descriptor.
 - e. If a non-maskable interrupt (NMI), an exception, or an external interrupt accesses a task gate in the IDT in IA-32e mode, a general-protection exception occurs.
 - f. If a non-maskable interrupt (NMI), an exception other than breakpoint exceptions (#BP) and overflow exceptions (#OF), or an external interrupt accesses a task gate in the IDT outside IA-32e mode, no privilege checks are performed.
 - g. If IRET is executed with RFLAGS.NT = 1 in IA-32e mode, a general-protection exception occurs.
 - h. If IRET is executed with RFLAGS.NT = 1 outside IA-32e mode, a TSS descriptor is accessed directly and no privilege checks are made.
2. Checks are made on the new TSS selector (for example, that is within GDT limits).
 3. The new TSS descriptor is read. (A page fault results if a relevant GDT page is not present).
 4. The TSS descriptor is checked for proper values of type (depends on type of task switch), P bit, S bit, and limit.

Only if checks 1–4 all pass (do not generate faults) might a VM exit occur. However, the ordering between a VM exit due to a task switch and a page fault resulting from accessing the old TSS or the new TSS is implementation-specific. Some processors may generate a page fault (instead of a VM exit due to a task switch) if accessing either TSS would cause a page fault. Other processors may generate a VM exit due to a task switch even if accessing either TSS would cause a page fault.

If an attempt at a task switch through a task gate in the IDT causes an exception (before generating a VM exit due to the task switch) and that exception causes a VM exit, information about the event whose delivery that accessed the task gate is recorded in the original-event fields and information about the exception that caused the VM exit is recorded in the exiting-event fields. See Section 30.2. The fact that a task gate was being accessed is not recorded in the VMCS.

If an attempt at a task switch through a task gate in the IDT causes VM exit due to the task switch, information about the event whose delivery accessed the task gate is recorded in the original-event fields of the VMCS. Since the cause of such a VM exit is a task switch and not some other event, the valid bit for the exiting-event identification field is 0. See Section 30.2.

28.4.3 Shadow-Stack Updates

As noted in Section 18.2.3, “Supervisor Shadow Stack Token,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, a switch of shadow stack may occur as part of IDT event delivery or an execution of far CALL that changes the CPL, or as part of IDT event delivery that uses the interrupt stack table (IST).

As part of the shadow-stack switch, the processor gains exclusive access to the new stack through manipulation of the **supervisor shadow stack token** located at the base of the new shadow stack. The processor reads the token and, among other things, confirms that bit 0 of the token (its **busy bit**) is 0. If the busy bit is already 1, the transition (event delivery or far CALL) cause a general-protection fault and does not complete. If the busy bit is 0, the transition sets the busy bit by writing to the token in memory. (The update is atomic with the original read of the token.)

If the transition commenced with CPL < 3, it will follow the token update by pushing three items on the new shadow stack (for the old values of the CS selector, instruction pointer, and shadow-stack pointer). As noted in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, if any of the pushes causes a VM exit, the processor will revert to the old shadow stack and the busy bit in the new shadow stack’s token remains set. The new shadow stack is said to be prematurely busy.

If the “prematurely busy shadow stack” VM-exit control is 1, a VM exit that results in a shadow stack becoming prematurely busy will indicate that fact through information saved in the VMCS. See Section 30.2.1.

28.5 FEATURES SPECIFIC TO VMX NON-ROOT OPERATION

Some VM-execution controls support features that are specific to VMX non-root operation. These are the VMX-preemption timer (Section 28.5.1) and the monitor trap flag (Section 28.5.2), translation of guest-physical addresses (Section 28.5.3 and Section 28.5.4), APIC virtualization (Section 28.5.6), VM functions (Section 28.5.7), and virtualization exceptions (Section 28.5.8).

28.5.1 VMX-Preemption Timer

If the last VM entry was performed with the 1-setting of “activate VMX-preemption timer” VM-execution control, the **VMX-preemption timer** counts down (from the value loaded by VM entry; see Section 29.7.4) in VMX non-root operation. When the timer counts down to zero, it stops counting down and a VM exit occurs (see Section 28.2).

The VMX-preemption timer counts down at rate proportional to that of the timestamp counter (TSC). Specifically, the timer counts down by 1 every time bit X in the TSC changes due to a TSC increment. The value of X is in the range 0–31 and can be determined by consulting the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

The VMX-preemption timer operates in the C-states C0, C1, and C2; it also operates in the shutdown and wait-for-SIPI states. If the timer counts down to zero in any state other than the wait-for SIPI state, the logical processor transitions to the C0 C-state and causes a VM exit; the timer does not cause a VM exit if it counts down to zero in the wait-for-SIPI state. The timer is not decremented in C-states deeper than C2.

Treatment of the timer in the case of system management interrupts (SMIs) and system-management mode (SMM) depends on whether the treatment of SMIs and SMM:

- If the default treatment of SMIs and SMM (see Section 34.14) is active, the VMX-preemption timer counts across an SMI to VMX non-root operation, subsequent execution in SMM, and the return from SMM via the RSM instruction. However, the timer can cause a VM exit only from VMX non-root operation. If the timer expires during SMI, in SMM, or during RSM, a timer-induced VM exit occurs immediately after RSM with its normal priority unless it is blocked based on activity state (Section 28.2).
- If the dual-monitor treatment of SMIs and SMM (see Section 34.15) is active, transitions into and out of SMM are VM exits and VM entries, respectively. The treatment of the VMX-preemption timer by those transitions is mostly the same as for ordinary VM exits and VM entries; Section 34.15.2 and Section 34.15.4 detail some differences.

28.5.2 Monitor Trap Flag

The **monitor trap flag** is a debugging feature that causes VM exits to occur on certain instruction boundaries in VMX non-root operation. Such VM exits are called **MTF VM exits**. An MTF VM exit may occur on an instruction boundary in VMX non-root operation as follows:

- If the “monitor trap flag” VM-execution control is 1 and VM entry is injecting a vectored event (see Section 29.6.1), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry.
- If VM entry is injecting a pending MTF VM exit (see Section 29.6.2), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0.
- If the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and a pending event (e.g., debug exception or interrupt) is delivered before an instruction can execute, an MTF VM exit is pending on the instruction boundary following delivery of the event (or any nested exception).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is a REP-prefixed string instruction:
 - If the first iteration of the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).
 - If the first iteration of the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary after that iteration.

- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is the XBEGIN instruction. In this case, an MTF VM exit is pending at the fallback instruction address of the XBEGIN instruction. This behavior applies regardless of whether advanced debugging of RTM transactional regions has been enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is neither a REP-prefixed string instruction or the XBEGIN instruction:
 - If the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).¹
 - If the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary following execution of that instruction. If the instruction is INT1, INT3, or INTO, this boundary follows delivery of any software exception. If the instruction is INT *n*, this boundary follows delivery of a software interrupt. If the instruction is HLT, the MTF VM exit will be from the HLT activity state.

No MTF VM exit occurs if another VM exit occurs before reaching the instruction boundary on which an MTF VM exit would be pending (e.g., due to an exception or triple fault).

An MTF VM exit occurs on the instruction boundary on which it is pending unless a higher priority event takes precedence or the MTF VM exit is blocked due to the activity state:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over MTF VM exits. MTF VM exits take priority over debug-trap exceptions and lower priority events.
- No MTF VM exit occurs if the processor is in either the shutdown activity state or wait-for-SIPI activity state. If a non-maskable interrupt subsequently takes the logical processor out of the shutdown activity state without causing a VM exit, an MTF VM exit is pending after delivery of that interrupt.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 43.2 for details.

28.5.3 Translation of Guest-Physical Addresses Using EPT

The extended page-table mechanism (EPT) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain physical addresses are treated as guest-physical addresses and are not used to access memory directly. Instead, guest-physical addresses are translated by traversing a set of EPT paging structures to produce physical addresses that are used to access memory.

Details of the EPT mechanism are given in Section 31.3.

28.5.4 Translation of Guest-Physical Addresses Used by Intel Processor Trace

As described in Chapter 36, Intel® Processor Trace (Intel PT) captures information about software execution using dedicated hardware facilities.

Intel PT can be configured so that the trace output is written to memory using physical addresses. For example, when the ToPA (table of physical addresses) output mechanism is used, the IA32_RTIT_OUTPUT_BASE MSR contains the physical address of the base of the current ToPA. Each entry in that table contains the physical address of an output region in memory. When an output region becomes full, the ToPA output mechanism directs subsequent trace output to the next output region as indicated in the ToPA.

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the logical processor treats the addresses used by Intel PT (the output addresses as well as those used to discover the output addresses) as guest-physical addresses, translating to physical addresses using EPT before trace output is written to memory.

Translating these addresses through EPT implies that the trace-output mechanism may cause EPT violations and VM exits; details are provided in Section 28.5.4.1. Section 28.5.4.2 describes a mechanism that ensures that these VM exits do not cause loss of trace data.

1. This item includes the cases of an invalid opcode exception—#UD—generated by the UD0, UD1, UD2, and UDB instructions and a BOUND-range exceeded exception—#BR—generated by the BOUND instruction.

28.5.4.1 Guest-Physical Address Translation for Intel PT: Details

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the addresses used by Intel PT are treated as guest-physical addresses and translated using EPT. These addresses include the addresses of the output regions as well as the addresses of the ToPA entries that contain the output-region addresses.

Translation of accesses by the trace-output process may result in EPT violations or EPT misconfigurations (Section 31.3.3), resulting in VM exits. EPT violations resulting for the trace-output process always cause VM exits and are never converted to virtualization exceptions (Section 28.5.8.1).

If no EPT violation or EPT misconfiguration occurs and if page-modification logging (Section 31.3.6) is enabled, the address of an output region may be added to the page-modification log. If the log is full, a page-modification log-full event occurs, resulting in a VM exit.

If the “virtualize APIC accesses” VM-execution control is 1, a guest-physical address used by the trace-output process may be translated to an address on the APIC-access page. In this case, the access by the trace-output process causes an APIC-access VM exit as discussed in Section 32.4.6.1.

28.5.4.2 Trace-Address Pre-Translation (TAPT)

Because it buffers trace data produced by Intel PT before it is written to memory, the processor ensures that buffered data is not lost when a VM exit disables Intel PT. Specifically, the processor ensures that there is sufficient space left in the current output page for the buffered data. If this were not done, buffered trace data could be lost and the resulting trace corrupted.

To prevent the loss of buffered trace data, the processor uses a mechanism called **trace-address pre-translation (TAPT)**. With TAPT, the processor translates using EPT the guest-physical address of the current output region before that address would be used to write buffered trace data to memory.

Because of TAPT, no translation (and thus no EPT violation) occurs at the time output is written to memory; the writes to memory use translations that were cached as part of TAPT. (The details given in Section 28.5.4.1 apply to TAPT.) TAPT ensures that, if a write to the output region would cause an EPT violation, the resulting VM exit is delivered at the time of TAPT, before the region would be used. This allows software to resolve the EPT violation at that time and ensures that, when it is necessary to write buffered trace data to memory, that data will not be lost due to an EPT violation.

TAPT (and resulting VM exits) may occur at any of the following times:

- When software in VMX non-root operation enables tracing by loading the IA32_RTIT_CTL MSR to set the TraceEn bit, using the WRMSR instruction or the XRSTORS instruction.
Any VM exit resulting from TAPT in this case is trap-like: the WRMSR or XRSTORS completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).
- At an instruction boundary when one output region becomes full and Intel PT transitions to the next output region.
VM exits resulting from TAPT in this case take priority over any pending debug exceptions. Such a VM exit will save information about such exceptions in the guest-state area of the VMCS.
- As part of a VM entry that enables Intel PT. See Section 29.5 for details.

TAPT may translate not only the guest-physical address of the current output region but those of subsequent output regions as well. (Doing so may provide better protection of trace data.) This implies that any VM exits resulting from TAPT may result from the translation of output-region addresses other than that of the current output region.

28.5.5 Translation of Guest-Physical Addresses Used by Architectural PEBS

As described in Section 22.10, “Architectural PEBS,” architectural PEBS records sampling information in response to the overflow of performance-monitoring counters.

Architectural PEBS is normally configured so that records are written to memory using physical addresses. The IA32_PEBS_BASE MSR contains the physical address of the base of the PEBS buffer. However, when the “PEBS uses

guest physical addresses” VM-execution control is 1, the logical processor treats the addresses used by PEBS as guest-physical addresses, translating to physical addresses using EPT before PEBS records are written to memory.

Translating these addresses through EPT implies that the writing of PEBS records may result in EPT violations or EPT misconfigurations (Section 31.3.3), causing VM exits. EPT violations resulting from PEBS always cause VM exits and are never converted to virtualization exceptions (Section 28.5.8.1).

If no EPT violation or EPT misconfiguration occurs and if page-modification logging (Section 31.3.6) is enabled, the address of a PEBS buffer may be added to the page-modification log. If the log is full, a page-modification log-full event occurs, resulting in a VM exit.

If the “virtualize APIC accesses” VM-execution control is 1, a guest-physical address used by PEBS may be translated to an address on the APIC-access page. In this case, the access by PEBS causes an APIC-access VM exit as discussed in Section 32.4.6.1.

The following items specify details of PEBS processing for when a memory access for PEBS causes one of the VM exits identified above:

- The overflow bits for any performance-monitoring counters that generated PEBS records are not reset.
- The performance-monitoring counters that generated PEBS records are not reloaded.
- The `WR_OFFSET` field in the `IA32_PEBS_INDEX` MSR is not incremented to indicate the writing of PEBS records.

28.5.6 APIC Virtualization

APIC virtualization is a collection of features that can be used to support the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC). When APIC virtualization is enabled, the processor emulates many accesses to the APIC, tracks the state of the virtual APIC, and delivers virtual interrupts — all in VMX non-root operation without a VM exit.

Details of the APIC virtualization are given in Chapter 32.

28.5.7 VM Functions

A **VM function** is an operation provided by the processor that can be invoked from VMX non-root operation without a VM exit. VM functions are enabled and configured by the settings of different fields in the VMCS. Software in VMX non-root operation invokes a VM function with the **VMFUNC** instruction; the value of EAX selects the specific VM function being invoked.

Section 28.5.7.1 explains how VM functions are enabled. Section 28.5.7.2 specifies the behavior of the **VMFUNC** instruction. Section 28.5.7.3 describes a specific VM function called **EPTP switching**.

28.5.7.1 Enabling VM Functions

Software enables VM functions generally by setting the “enable VM functions” VM-execution control. A specific VM function is enabled by setting the corresponding VM-function control.

Suppose, for example, that software wants to enable EPTP switching (VM function 0; see Section 27.6.14). To do so, it must set the “activate secondary controls” VM-execution control (bit 31 of the primary processor-based VM-execution controls), the “enable VM functions” VM-execution control (bit 13 of the secondary processor-based VM-execution controls) and the “EPTP switching” VM-function control (bit 0 of the VM-function controls).

28.5.7.2 General Operation of the VMFUNC Instruction

The **VMFUNC** instruction causes an invalid-opcode exception (#UD) if the “enable VM functions” VM-execution controls is 0¹ or the value of EAX is greater than 63 (only VM functions 0–63 can be enable). Otherwise, the instruction causes a VM exit if the bit at position EAX is 0 in the VM-function controls (the selected VM function is not enabled). If such a VM exit occurs, the basic exit reason used is 59 (3BH), indicating “VMFUNC”, and the length

1. “Enable VM functions” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “enable VM functions” VM-execution control were 0. See Section 27.6.2.

of the VMFUNC instruction is saved into the VM-exit instruction-length field. If the instruction causes neither an invalid-opcode exception nor a VM exit due to a disabled VM function, it performs the functionality of the VM function specified by the value in EAX.

Individual VM functions may perform additional fault checking (e.g., one might cause a general-protection exception if $CPL > 0$). In addition, specific VM functions may include checks that might result in a VM exit. If such a VM exit occurs, VM-exit information is saved as described in the previous paragraph. The specification of a VM function may indicate that additional VM-exit information is provided.

The specific behavior of the EPTP-switching VM function (including checks that result in VM exits) is given in Section 28.5.7.3.

28.5.7.3 EPTP Switching

EPTP switching is VM function 0. This VM function allows software in VMX non-root operation to load a new value for the EPT pointer (EPTP), thereby establishing a different EPT paging-structure hierarchy (see Section 31.3 for details of the operation of EPT). Software is limited to selecting from a list of potential EPTP values configured in advance by software in VMX root operation.

Specifically, the value of ECX is used to select an entry from the EPTP list, the 4-KByte structure referenced by the EPTP-list address (see Section 27.6.14; because this structure contains 512 8-Byte entries, VMFUNC causes a VM exit if $ECX \geq 512$). The EPTP value in the selected entry is evaluated to determine whether it is valid for EPTP switching: a value is valid if (1) it is the same as the current EPTP value in bits 5:3 (these bits specify the EPT page-walk length); and (2) it would not cause VM entry to fail (see Section 29.2.1.1). If the value is invalid, a VM exit occurs. Otherwise, the value is stored in the EPTP field of the current VMCS and is used for subsequent accesses using guest-physical addresses. The following pseudocode provides details:

```
IF ECX ≥ 512
    THEN VM exit;
ELSE
    tent_EPTP := 8 bytes from EPTP-list address + 8 * ECX;
    IF tent_EPTP is not a valid EPTP value (would cause VM entry to fail if in EPTP or change EPT page-walk length)
        THEN VM exit;
    ELSE
        write tent_EPTP to the EPTP field in the current VMCS;
        use tent_EPTP as the new EPTP value for address translation;
        IF processor supports the 1-setting of the "EPT-violation #VE" VM-execution control
            THEN
                write ECX[15:0] to EPTP-index field in current VMCS;
                use ECX[15:0] as EPTP index for subsequent EPT-violation virtualization exceptions (see Section 28.5.8.2);
            FI;
        FI;
    FI;
```

Execution of the EPTP-switching VM function does not modify the state of any registers; no flags are modified.

If the "Intel PT uses guest physical addresses" VM-execution control is 1 and $IA32\_RTIT\_CTL.TraceEn = 1$, any execution of the EPTP-switching VM function causes a VM exit.¹

As noted in Section 28.5.7.2, an execution of the EPTP-switching VM function that causes a VM exit (as specified above), uses the basic exit reason 59, indicating "VMFUNC". The length of the VMFUNC instruction is saved into the VM-exit instruction-length field. No additional VM-exit information is provided.

An execution of VMFUNC loads EPTP from the EPTP list (and thus does not cause a fault or VM exit) is called an **EPTP-switching VMFUNC**. After an EPTP-switching VMFUNC, control passes to the next instruction. The logical processor starts creating and using guest-physical and combined mappings associated with the new value of bits 51:12 of EPTP; the combined mappings created and used are associated with the current VPID and PCID (these are not changed by VMFUNC).² If the "enable VPID" VM-execution control is 0, an EPTP-switching VMFUNC invalidates

1. Such a VM exit ensures the proper recording of trace data that might otherwise be lost during the change of EPT paging-structure hierarchy. Software handling the VM exit can change emulate the VM function and then resume the guest.

combined mappings associated with VPID 0000H (for all PCIDs and for all EPTRTA values, where EPTRTA is the value of bits 51:12 of EPTP).

Because an EPTP-switching VMFUNC may change the translation of guest-physical addresses, it may affect use of the guest-physical address in CR3. The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration due to the translation of that guest-physical address through the new EPT paging structures. The following items provide details that apply if CR0.PG = 1:

- If 32-bit paging or 4-level paging¹ is in use (either CR4.PAE = 0 or IA32_EFER.LMA = 1), the next memory access with a linear address uses the translation of the guest-physical address in CR3 through the new EPT paging structures. As a result, this access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.
- If PAE paging is in use (CR4.PAE = 1 and IA32_EFER.LMA = 0), an EPTP-switching VMFUNC **does not** load the four page-directory-pointer-table entries (PDPTes) from the guest-physical address in CR3. The logical processor continues to use the four guest-physical addresses already present in the PDPTes. The guest-physical address in CR3 is not translated through the new EPT paging structures (until some operation that would load the PDPTes).

The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during the translation of a guest-physical address in any of the PDPTes. A subsequent memory access with a linear address uses the translation of the guest-physical address in the appropriate PDPTE through the new EPT paging structures. As a result, such an access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.

If an EPTP-switching VMFUNC establishes an EPTP value that enables accessed and dirty flags for EPT (by setting bit 6), subsequent memory accesses may fail to set those flags as specified if there has been no appropriate execution of INVEPT since the last use of an EPTP value that does not enable accessed and dirty flags for EPT (because bit 6 is clear) and that is identical to the new value on bits 51:12.

If the processor supports the 1-setting of the “EPT-violation #VE” VM-execution control, an EPTP-switching VMFUNC loads the value in ECX[15:0] into the EPTP-index field in current VMCS. Subsequent EPT-violation virtualization exceptions will save this value into the virtualization-exception information area (see Section 28.5.8.2).

28.5.8 Virtualization Exceptions

A **virtualization exception** is a new processor exception. It uses vector 20 and is abbreviated #VE.

A virtualization exception can occur only in VMX non-root operation. Virtualization exceptions occur only with certain settings of certain VM-execution controls. Generally, these settings imply that certain conditions that would normally cause VM exits instead cause virtualization exceptions

In particular, the 1-setting of the “EPT-violation #VE” VM-execution control causes some EPT violations to generate virtualization exceptions instead of VM exits. Section 28.5.8.1 provides the details of how the processor determines whether an EPT violation causes a virtualization exception or a VM exit.

When the processor encounters a virtualization exception, it saves information about the exception to the virtualization-exception information area; see Section 28.5.8.2.

After saving virtualization-exception information, the processor delivers a virtualization exception as it would any other exception; see Section 28.5.8.3 for details.

28.5.8.1 Convertible EPT Violations

If the “EPT-violation #VE” VM-execution control is 0 (e.g., on processors that do not support this feature), EPT violations always cause VM exits. If instead the control is 1, certain EPT violations may be converted to cause virtualization exceptions instead; such EPT violations are **convertible**.

The values of certain EPT paging-structure entries determine which EPT violations are convertible. Specifically, bit 63 of certain EPT paging-structure entries may be defined to mean **suppress #VE**:

2. If the “enable VPID” VM-execution control is 0, the current VPID is 0000H; if CR4.PCIDE = 0, the current PCID is 000H.

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

- If bits 2:0 of an EPT paging-structure entry are all 0, the entry is not **present**.¹ If the processor encounters such an entry while translating a guest-physical address, it causes an EPT violation. The EPT violation is convertible if and only if bit 63 of the entry is 0.
- If an EPT paging-structure entry is present, the following cases apply:
 - If the value of the EPT paging-structure entry is not supported, the entry is **misconfigured**. If the processor encounters such an entry while translating a guest-physical address, it causes an EPT misconfiguration (not an EPT violation). EPT misconfigurations always cause VM exits.
 - If the value of the EPT paging-structure entry is supported, the following cases apply:
 - If bit 7 of the entry is 1, or if the entry is an EPT PTE, the entry maps a page. If the processor uses such an entry to translate a guest-physical address, and if an access to that address causes an EPT violation, the EPT violation is convertible if and only if bit 63 of the entry is 0.
 - If bit 7 of the entry is 0 and the entry is not an EPT PTE, the entry references another EPT paging structure. The processor does not use the value of bit 63 of the entry to determine whether any subsequent EPT violation is convertible.

If an access to a guest-physical address causes an EPT violation, bit 63 of exactly one of the EPT paging-structure entries used to translate that address is used to determine whether the EPT violation is convertible: either a entry that is not present (if the guest-physical address does not translate to a physical address) or an entry that maps a page (if it does).

A convertible EPT violation instead causes a virtualization exception if the following all hold:

- CR0.PE = 1;
- the logical processor is not in the process of delivering an event;
- the EPT violation did not cause a shadow stack to become prematurely busy (see Section 28.4.3);
- the EPT violation does not result from the output process of Intel Processor Trace (Section 28.5.4); and
- the EPT violation does not result from PEBS (Section 28.5.5); and
- the 32 bits at offset 4 in the virtualization-exception information area are all 0.

Delivery of virtualization exceptions writes the value FFFFFFFFH to offset 4 in the virtualization-exception information area (see Section 28.5.8.2). Thus, once a virtualization exception occurs, another can occur only if software clears this field.

28.5.8.2 Virtualization-Exception Information

Virtualization exceptions save data into the virtualization-exception information area (see Section 27.6.19). Table 28-23 enumerates the data saved and the format of the area.

Table 28-23. Format of the Virtualization-Exception Information Area

Byte Offset	Contents
0	The 32-bit value that would have been saved into the VMCS as an exit reason had a VM exit occurred instead of the virtualization exception. For EPT violations, this value is 48 (00000030H)
4	FFFFFFFFH
8	The 64-bit value that would have been saved into the VMCS as an exit qualification had a VM exit occurred instead of the virtualization exception
16	The 64-bit value that would have been saved into the VMCS as a guest-linear address had a VM exit occurred instead of the virtualization exception

1. If the “mode-based execute control for EPT” VM-execution control is 1, an EPT paging-structure entry is present if any of bits 2:0 or bit 10 is 1.

Table 28-23. Format of the Virtualization-Exception Information Area (Contd.)

Byte Offset	Contents
24	The 64-bit value that would have been saved into the VMCS as a guest-physical address had a VM exit occurred instead of the virtualization exception
32	The current 16-bit value of the EPTP index VM-execution control (see Section 27.6.19 and Section 28.5.7.3)

A VMM may allow guest software to access the virtualization-exception information area. If it does, the guest software may modify that memory (e.g., to clear the 32-bit value at offset 4; see Section 28.5.8.1). (This is an exception to the general requirement given in Section 27.11.4.)

28.5.8.3 Delivery of Virtualization Exceptions

After saving virtualization-exception information, the processor treats a virtualization exception as it does other exceptions:

- If bit 20 (#VE) is 1 in the exception bitmap in the VMCS, a virtualization exception causes a VM exit (see below). If the bit is 0, the virtualization exception is delivered using vector 20.
- Virtualization exceptions produce no error code. Delivery of a virtualization exception pushes no error code on the stack.
- With respect to double faults, virtualization exceptions have the same severity as page faults. If delivery of a virtualization exception encounters a nested fault that is either contributory or a page fault, a double fault (#DF) is generated. See Chapter 7, “Interrupt 8—Double Fault Exception (#DF)” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

It is not possible for a virtualization exception to be encountered while delivering another exception (see Section 28.5.8.1).

If a virtualization exception causes a VM exit directly (because bit 20 is 1 in the exception bitmap), information about the exception is saved normally in the exiting-event identification field in the VMCS (see Section 30.2.2). Specifically, the event is reported as a hardware exception with vector 20 and no error code. Bit 12 of the field (NMI unblocking due to IRET) is set normally.

If a virtualization exception causes a VM exit indirectly (because bit 20 is 0 in the exception bitmap and delivery of the exception generates an event that causes a VM exit), information about the exception is saved normally in the original-event identification field in the VMCS (see Section 30.2.4). Specifically, the event is reported as a hardware exception with vector 20 and no error code.

28.5.9 PASID Translation

The ENQCMD and ENQCMLS instructions each performs a 64-byte enqueue store that includes a 20-bit PASID value in bits 19:0. For ENQCMD, the PASID is normally the value of IA32_PASID[19:0], while for ENQCMLS, the PASID is normally read from memory.

If the “PASID translation” VM-execution control is 1, the PASID value identified in the previous paragraph is treated as a **guest PASID**. PASID translation converts this guest PASID to a 20-bit **host PASID**. After this translation, the enqueue store is performed, using the host PASID in place of the guest PASID.

PASID translation is implemented by two hierarchies of data structures (**PASID-translation hierarchies**) configured by a VMM. Guest PASIDs 00000H to 7FFFFH are translated through the low PASID-translation hierarchy, while guest PASIDs 80000 to FFFFFH are translated through the high PASID-translation hierarchy.

The root of each PASID-translation hierarchy is a 4-KByte **PASID directory**. The low PASID directory is located at the low PASID directory address, and the high PASID directory is located at the high PASID directory address (these physical addresses are VM-execution control fields in the VMCS). A PASID directory comprises 512 8-byte entries, each of which has the following format:

- Bit 0 is the entry’s present bit. The entry is used only if this bit is 1.

- Bits 11:1 are reserved and must be 0.
- Bits MAXPHYADDR-1:12 specify the 4-KByte aligned address of a PASID table (see below), where MAXPHYADDR is defined as follows:
 - Usually, MAXPHYADDR is the processor's supported physical-address width as enumerated in CPUID.80000008H:EAX[7:0] (this width is at most 52).
 - If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.¹
- Bits 63:MAXPHYADDR are reserved and must be 0.

A PASID-translation hierarchy also includes up to 512 4-KByte **PASID tables**; each of these is referenced by a PASID directory entry (see above). A PASID table comprises 1024 4-byte entries, each of which has the following format:

- Bits 19:0 are the host PASID specified by the entry.
- Bits 30:20 are reserved and must be 0.
- Bit 31 is the entry's valid bit. The entry is used only if this bit is 1.

When PASID translation is enabled, the guest PASID determined by the instruction (see above) is converted to a host PASID using the following process:

- If bit 19 of guest PASID is clear, the low PASID directory is used; otherwise, the high PASID directory is used.
- Bits 18:10 of the guest PASID select an entry from the PASID directory. A VM exit occurs if the entry's present bit is clear or if any reserved bit is set. Otherwise, bits MAXPHYADDR-1:0 of the entry (with bit 0 cleared) contain the physical address of a PASID table.
- Bits 9:0 of the guest PASID select an entry from the PASID table. A VM exit occurs if the entry's valid bit is clear or if any reserved bit is set. Otherwise, bits 19:0 of the entry are the host PASID.

If PASID translation results in a VM exit (due to a present or valid bit being clear, or a reserved bit being set), the instruction does not complete and no enqueue store is performed.

28.6 UNRESTRICTED GUESTS

The first processors to support VMX operation require CR0.PE and CR0.PG to be 1 in VMX operation (see Section 26.8). This restriction implies that guest software cannot be run in unpagged protected mode or in real-address mode. Later processors support a VM-execution control called "unrestricted guest".² If this control is 1, CR0.PE and CR0.PG may be 0 in VMX non-root operation. Such processors allow guest software to run in unpagged protected mode or in real-address mode. The following items describe the behavior of such software:

- The MOV CR0 instructions does not cause a general-protection exception simply because it would set either CR0.PE and CR0.PG to 0. See Section 28.3 for details.
- A logical processor treats the values of CR0.PE and CR0.PG in VMX non-root operation just as it does outside VMX operation. Thus, if CR0.PE = 0, the processor operates as it does normally in real-address mode (for example, it uses the 16-bit **interrupt table** to deliver interrupts and exceptions). If CR0.PG = 0, the processor operates as it does normally when paging is disabled.
- Processor operation is modified by the fact that the processor is in VMX non-root operation and by the settings of the VM-execution controls just as it is in protected mode or when paging is enabled. Instructions, interrupts, and exceptions that cause VM exits in protected mode or when paging is enabled also do so in real-address mode or when paging is disabled. The following examples should be noted:
 - If CR0.PG = 0, page faults do not occur and thus cannot cause VM exits.
 - If CR0.PE = 0, invalid-TSS exceptions do not occur and thus cannot cause VM exits.

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.
2. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "unrestricted guest" VM-execution control were 0. See Section 27.6.2.

- If CR0.PE = 0, the following instructions cause invalid-opcode exceptions and do not cause VM exits: INVEPT, INVVPID, LLDT, LTR, SLDT, STR, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.
- If CR0.PG = 0, each linear address is passed directly to the EPT mechanism for translation to a physical address.¹ The guest memory type passed on to the EPT mechanism is WB (writeback).

1. As noted in Section 29.2.1.1, the “enable EPT” VM-execution control must be 1 if the “unrestricted guest” VM-execution control is 1.

13. Updates to Chapter 29, Volume 3C

Change bars and violet text show changes to Chapter 29 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- "PEBS uses guest physical addresses" was added to the list of VM-execution controls that can be set when the "enabled EPT" VM-execution control is 1 (Section 29.2.1.1).

Software can enter VMX non-root operation using either of the VM-entry instructions VMLAUNCH and VMRESUME. VMLAUNCH can be used only with a VMCS whose launch state is clear and VMRESUME can be used only with a VMCS whose the launch state is launched. VMLAUNCH should be used for the first VM entry after VMCLEAR; VMRESUME should be used for subsequent VM entries with the same VMCS.

Each VM entry performs the following steps in the order indicated:

1. Basic checks are performed to ensure that VM entry can commence (Section 29.1).
2. The control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation and that the VMCS is correctly configured to support the next VM exit (Section 29.2).
3. The following are performed in parallel or in any order (Section 29.3):
 - The guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures.
 - Processor state is loaded from the guest-state area and based on controls in the VMCS.
 - Address-range monitoring is cleared.
4. MSRs are loaded from the VM-entry MSR-load area (Section 29.4).
5. If VMLAUNCH is being executed, the launch state of the VMCS is set to “launched.”
6. If the “Intel PT uses guest physical addresses” VM-execution control is 1, trace-address pre-translation (TAPT) may occur (see Section 28.5.4 and Section 29.5).
7. An event may be injected in the guest context (Section 29.6).

Steps 1–4 above perform checks that may cause VM entry to fail. Such failures occur in one of the following three ways:

- Some of the checks in Section 29.1 may generate ordinary faults (for example, an invalid-opcode exception). Such faults are delivered normally.
- Some of the checks in Section 29.1 and all the checks in Section 29.2 cause control to pass to the instruction following the VM-entry instruction. The failure is indicated by setting RFLAGS.ZF¹ (if there is a current VMCS) or RFLAGS.CF (if there is no current VMCS). If there is a current VMCS, an error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 32 for the error numbers.
- The checks in Section 29.3 and Section 29.4 cause processor state to be loaded from the host-state area of the VMCS (as would be done on a VM exit). Information about the failure is stored in the VM-exit information fields. See Section 29.8 for details.

EFLAGS.TF = 1 causes a VM-entry instruction to generate a single-step debug exception only if failure of one of the checks in Section 29.1 and Section 29.2 causes control to pass to the following instruction. A VM-entry does not generate a single-step debug exception in any of the following cases: (1) the instruction generates a fault; (2) failure of one of the checks in Section 29.3 or in loading MSRs causes processor state to be loaded from the host-state area of the VMCS; or (3) the instruction passes all checks in Section 29.1, Section 29.2, and Section 29.3 and there is no failure in loading MSRs.

Section 34.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, code running in SMM returns using VM entries instead of the RSM instruction. A VM entry **returns from SMM** if it is executed in SMM and the “entry to SMM” VM-entry control is 0. VM entries that return from SMM differ from ordinary VM entries in ways that are detailed in Section 34.15.4.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

29.1 BASIC VM-ENTRY CHECKS

Before a VM entry commences, the current state of the logical processor is checked in the following order:

1. If the logical processor is in virtual-8086 mode or compatibility mode, an invalid-opcode exception is generated.
2. If the current privilege level (CPL) is not zero, a general-protection exception is generated.
3. If there is no current VMCS, RFLAGS.CF is set to 1 and control passes to the next instruction.
4. If there is a current VMCS but the current VMCS is a shadow VMCS (see Section 27.10), RFLAGS.CF is set to 1 and control passes to the next instruction.
5. If there is a current VMCS that is not a shadow VMCS, the following conditions are evaluated in order; any of these cause VM entry to fail:
 - a. If there is MOV-SS blocking (see Table 27-3).
 - b. If the VM entry is invoked by VMLAUNCH and the VMCS launch state is not clear.
 - c. If the VM entry is invoked by VMRESUME and the VMCS launch state is not launched.

If any of these checks fail, RFLAGS.ZF is set to 1 and control passes to the next instruction. An error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 32 for the error numbers.

29.2 CHECKS ON VMX CONTROLS AND HOST-STATE AREA

If the checks in Section 29.1 do not cause VM entry to fail, the control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation, that the VMCS is correctly configured to support the next VM exit, and that, after the next VM exit, the processor's state is consistent with the Intel 64 and IA-32 architectures.

VM entry fails if any of these checks fail. When such failures occur, control is passed to the next instruction, RFLAGS.ZF is set to 1 to indicate the failure, and the VM-instruction error field is loaded with an error number that indicates whether the failure was due to the controls or the host-state area (see Chapter 32).

These checks may be performed in any order. Thus, an indication by error number of one cause (for example, host state) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same VMCS. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The checks on the controls and the host-state area are presented in Section 29.2.1 through Section 29.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

29.2.1 Checks on VMX Controls

This section identifies VM-entry checks on the VMX control fields.

Some of these fields contain physical addresses. These fields are checked against the allowed address width:

- If IA32_VMX_BASIC[48] is read as 1 (implying that the processor does not support Intel 64 architecture; see Section A.1, "Basic VMX Information"), these addresses must not set any bits in the range 63:32.
- Otherwise, the widths of physical addresses are limited by MAXPHYADDR, a value derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.¹ Physical address should not set bits in the range 63:MAXPHYADDR.

29.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:¹

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.1).
- Reserved bits in the primary processor-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.2).
- If the “activate secondary controls” primary processor-based VM-execution control is 1, reserved bits in the secondary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.3).

If the “activate secondary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary processor-based VM-execution controls. The logical processor operates as if all the secondary processor-based VM-execution controls were 0.

- If the “activate tertiary controls” primary processor-based VM-execution control is 1, reserved bits in the tertiary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.4).

If the “activate tertiary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the tertiary processor-based VM-execution controls. The logical processor operates as if all the tertiary processor-based VM-execution controls were 0.

- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR `IA32_VMX_MISC` to determine the number of values supported (see Appendix A.6).
- Bits 11:0 of the following fields must be 0, and the fields are also subject to the checks on physical-address width described above in Section 28.2.1:
 - If the “use I/O bitmaps” VM-execution control is 1, each I/O-bitmap address.
 - If the “use MSR bitmaps” VM-execution control is 1, the MSR-bitmap address.
 - If the “use TPR shadow” VM-execution control is 1, the virtual-APIC address.
 If the checks on that address are satisfied, bytes 3:1 of VTPR (see Section 31.1.1) may be cleared (behavior may be implementation-specific).
 The clearing of these bytes may occur even if the VM entry fails. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it causes processor state to be loaded from the host-state area of the VMCS.
 - If the “virtualize APIC-accesses” VM-execution control is 1, the APIC-access address.
 - If the “enable PML” VM-execution control is 1, the PML address.
 - If the “sub-page write permissions for EPT” VM-execution control is 1, the SPPTP VM-execution control field (see Table 26-11 in Section 26.6.21).
 - If the “enable VM functions” processor-based VM-execution control is 1 and the “EPTP switching” VM-function control is 1, the EPTP-list address.
 - If the “VMCS shadowing” VM-execution control is 1, the VMREAD-bitmap and VMWRITE-bitmap addresses.
 - If the “EPT-violation #VE” VM-execution control is 1, the virtualization-exception information address.
 - If the “PASID translation” VM-execution control is 1, the low PASID directory address and the high PASID directory address.

1. `IA32_TME_ACTIVATE[39:36]` is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than `IA32_TME_ACTIVATE[35:32]`, which is the number physical-address bits used for key identifiers generally.

1. If the “activate secondary controls” primary processor-based VM-execution control is 0, VM entry operates as if each secondary processor-based VM-execution control were 0. Similarly, if the “activate tertiary controls” primary processor-based VM-execution control is 0, VM entry operates as if each tertiary processor-based VM-execution control were 0. See Section 27.6.2.

- If the “use TPR shadow” VM-execution control is 1 and the “virtual-interrupt delivery” VM-execution control is 0, bits 31:4 of the TPR threshold VM-execution control field must be 0.
- The following check is performed if the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” and “virtual-interrupt delivery” VM-execution controls are both 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 of VTPR (see Section 32.1.1).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” VM-execution control must be 0.
- If the “virtual NMIs” VM-execution control is 0, the “NMI-window exiting” VM-execution control must be 0.
- If the “use TPR shadow” VM-execution control is 0, the following VM-execution controls must also be 0: “virtualize x2APIC mode”, “APIC-register virtualization”, “virtual-interrupt delivery”, and “IPI virtualization”.
- If the “use TPR shadow” VM-execution control is 0, the following VM-execution controls must also be 0: “virtualize x2APIC mode”, “APIC-register virtualization”, “virtual-interrupt delivery”, and “IPI virtualization”.
- If the “virtualize x2APIC mode” VM-execution control is 1, the “virtualize APIC accesses” VM-execution control must be 0.
- If the “virtual-interrupt delivery” VM-execution control is 1, the “external-interrupt exiting” VM-execution control must be 1.
- If the “process posted interrupts” VM-execution control is 1, the following must be true:
 - The “virtual-interrupt delivery” VM-execution control is 1.
 - The “acknowledge interrupt on exit” VM-exit control is 1.
 - The posted-interrupt notification vector has a value in the range 0–255 (bits 15:8 are all 0).
 - Bits 5:0 of the posted-interrupt descriptor address are all 0.
 - The posted-interrupt descriptor address satisfies the checks on physical-address width described above in Section 29.2.1.
- If the “IPI virtualization” VM-execution control is 1, the following must be true:
 - Bits 2:0 of the PID-pointer table address are all 0.
 - The PID-pointer table address satisfies the checks on physical-address width described above in Section 29.2.1.
 - The address of the last entry in the PID-pointer table satisfies the checks on physical-address width described above. (This address is the PID-pointer table address plus 8 times the last PID-pointer index.)
- If the “enable VPID” VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.
- If the “enable EPT” VM-execution control is 1, the EPTP VM-execution control field (see Table 27-9 in Section 27.6.11) must satisfy the following checks:
 - The EPT memory type (bits 2:0) must be a value supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10).
 - Bits 5:3 must contain a value 1 less than an EPT page-walk length supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Section 31.3.2 and Appendix A.10).
 - Bit 6 (enable bit for accessed and dirty flags for EPT) must be 0 if bit 21 of the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10) is read as 0, indicating that the processor does not support accessed and dirty flags for EPT.
 - Reserved bits 11:7 must all be 0.
 - The 4-KByte-aligned address defined by bits 63:12 must satisfy the checks on physical-address width described above in Section 29.2.1.
- The “enable EPT” VM-execution control must be 1 if any of the following VM-execution controls is 1: “enable PML”, “unrestricted guest”, “mode-based execute control for EPT”, “sub-page write permissions for EPT”, “Intel PT uses guest physical addresses”, “PEBS uses guest physical addresses”, “enable HLAT”, “EPT paging-write control”, or “guest-paging verification.”
- The following checks apply if the logical processor is in SEAM at the time of VM entry:

- The “enable EPT” VM-execution control must be 1.
- Bits 11:0 and bits 63:M must be zero in the SEAM shared EPT pointer, where M is the value enumerated by CPUID.80000008H:EAX[7:0].
- If the “enable VM functions” processor-based VM-execution control is 1, reserved bits in the VM-function controls must be clear. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.11). In addition, the following check is performed based on the setting of bits in the VM-function controls (see Section 27.6.14):
 - If “EPTP switching” VM-function control is 1, the “enable EPT” VM-execution control must also be 1.

If the “enable VM functions” processor-based VM-execution control is 0, no checks are performed on the VM-function controls.
- If the logical processor is operating with Intel PT enabled (if IA32_RTIT_CTL.TraceEn = 1) at the time of VM entry, the “load IA32_RTIT_CTL” VM-entry control must be 0.
- If the “Intel PT uses guest physical addresses” VM-execution control is 1, the “load IA32_RTIT_CTL” VM-entry control and the “clear IA32_RTIT_CTL” VM-exit control must both be 1.
- If the “use TSC scaling” VM-execution control is 1, the TSC-multiplier must not be zero.
- If the “enable HLAT” VM-execution control is 1, the following bits in the HLATP VM-execution control field (see Table 27-12 in Section 27.6.22) must be zero: bits 2:0, bits 11:5. In addition, the 4-KByte-aligned address defined by bits 63:12 must satisfy the checks on physical-address width described above in Section 29.2.1.

29.2.1.2 VM-Exit Control Fields

VM entries perform the following checks on the VM-exit control fields.

- Reserved bits in the primary VM-exit controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.4.1).
- If the “activate secondary controls” primary VM-exit control is 1, reserved bits in the secondary VM-exit controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.4.2).
- If the “activate secondary controls” primary VM-exit control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary VM-exit controls. The logical processor operates as if all the secondary VM-exit controls were 0.
- If the “activate VMX-preemption timer” VM-execution control is 0, the “save VMX-preemption timer value” VM-exit control must also be 0.
- The following checks are performed for the VM-exit MSR-store address if the VM-exit MSR-store count field is non-zero:
 - The lower 4 bits of the VM-exit MSR-store address must be 0. The address must satisfy the checks on physical-address width described above in Section 29.2.1.
 - The address of the last byte in the VM-exit MSR-store area must satisfy the checks on physical-address width described above. The address of this last byte is VM-exit MSR-store address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)
- The following checks are performed for the VM-exit MSR-load address if the VM-exit MSR-load count field is non-zero:
 - The lower 4 bits of the VM-exit MSR-load address must be 0. The address must satisfy the checks on physical-address width described above.
 - The address of the last byte in the VM-exit MSR-load area must satisfy the checks on physical-address width described above. The address of this last byte is VM-exit MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)

29.2.1.3 VM-Entry Control Fields

VM entries perform the following checks on the VM-entry control fields.

- Reserved bits in the VM-entry controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.5).
- Fields relevant to VM-entry event injection must be set properly. These fields are the injected-event identification field (see Table 27-18 in Section 27.8.3, “VM-Entry Controls for Event Injection”), the injected-event error code, and the VM-entry instruction length. If the valid bit (bit 31) in the injected-event identification field is 1, the following must hold:
 - The field’s event type (bits 10:8) is not set to a reserved value. Value 1 is reserved on all logical processors; value 7 (other event) is reserved on logical processors that support neither the 1-setting of the “monitor trap flag” VM-execution control nor FRED transitions.
 - The field’s vector (bits 7:0) is consistent with the event type:
 - If the event type is non-maskable interrupt (NMI), the vector is 2.
 - If the event type is hardware exception, the vector is at most 31.
 - If the event type is other event, the vector is 0 (indicating a pending MTF VM exit, only if “monitor trap flag” is supported) or, if FRED transitions would be enabled following VM entry (bit 32 of the CR4 field — FRED — in the guest-state area is 1), the vector is 1 (indicating SYSCALL) or 2 (indicating SYSENTER).
 - The field’s deliver-error-code bit (bit 11) is 1 if each of the following holds: (1) the event type is hardware exception; (2) either (a) the “unrestricted guest” VM-execution control is 0; or (b) bit 0 (corresponding to CR0.PE) is set in the CR0 field in the guest-state area; (3) IA32_VMX_BASIC[56] is read as 0 (see Appendix A.1); and (4) the vector indicates one of the following exceptions: #DF (vector 8), #TS (10), #NP (11), #SS (12), #GP (13), #PF (14), or #AC (17).
 - The field’s deliver-error-code bit is 0 if any of the following holds: (1) the event type is not hardware exception; (2) both (a) the “unrestricted guest” VM-execution control is 1; and (b) bit 0 is clear in the CR0 field in the guest-state area; or (3) IA32_VMX_BASIC[56] is read as 0 and the vector is in one of the following ranges: 0–7, 9, 15, 16, or 18–31.

NOTE

IA32_VMX_BASIC[56] will be read as 1 on any processor that supports FRED transitions.

- The field’s nested-exception bit (bit 13) is 0 if either of the following hold: (1) the event type is not hardware exception; or (2) IA32_VMX_BASIC[58] is read as 0.
- Reserved bits in the field (30:14 and 12) are 0.
- If the deliver-error-code bit (bit 11) is 1, bits 31:16 of the injected-event error-code field are 0.
- The VM-entry instruction-length field is in the range 0–15 if the event type is software interrupt, software exception, or privileged software exception; or if the event type is other event and the vector is either 1 (SYSCALL) or 2 (SYSENTER). A VM-entry instruction length of 0 is allowed only if IA32_VMX_MISC[30] is read as 1; see Appendix A.6.
- The following checks are performed for the VM-entry MSR-load address if the VM-entry MSR-load count field is non-zero:
 - The lower 4 bits of the VM-entry MSR-load address must be 0. The address must satisfy the checks on physical-address width described above in Section 29.2.1.
 - The address of the last byte in the VM-entry MSR-load area must satisfy the checks on physical-address width described above. The address of this last byte is VM-entry MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)

If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1, “Basic VMX Information”.
- If the processor is not in SMM, the “entry to SMM” and “deactivate dual-monitor treatment” VM-entry controls must be 0.
- The “entry to SMM” and “deactivate dual-monitor treatment” VM-entry controls cannot both be 1.

29.2.2 Checks on Host Control Registers, MSRs, and SSP

The following checks are performed on fields in the host-state area that correspond to control registers and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 26.8).¹
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 26.8).
- If bit 23 in the CR4 field (corresponding to CET) is 1, bit 16 in the CR0 field (WP) must also be 1.
- On processors that support Intel 64 architecture, the CR3 field must be such that bits reserved in CR3 are 0.² In particular, bits 63:32 are checked for allowed address width:
 - If IA32_VMX_BASIC[48] is read as 1 (implying that the processor does not support Intel 64 architecture; see Appendix A.1, “Basic VMX Information”), the field must not set any of bits 63:32.
 - Otherwise, the field’s width is limited by MAXPHYADDR, which is derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. The field should not set bits in the range 63:MAXPHYADDR.
- On processors that support Intel 64 architecture, the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-exit control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 22-3).
- If the “load IA32_PAT” VM-exit control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the “load IA32_EFER” VM-exit control is 1, bits reserved in the IA32_EFER MSR must be 0 in the field for that register. In addition, the values of the LMA and LME bits in the field must each be that of the “host address-space size” VM-exit control.
- If the “load CET state” VM-exit control is 1, the IA32_S_CET field must not set any bits reserved in the IA32_S_CET MSR, and bit 10 (corresponding to SUPPRESS) and bit 11 (TRACKER) in the field cannot both be set.
- If the “load CET state” VM-exit control is 1, bits 1:0 must be 0 in the SSP field.
- If the “load PKRS” VM-exit control is 1, bits 63:32 must be 0 in the IA32_PKRS field.
- If the “load FRED” VM-exit control is 1, the following must hold for the indicated fields:
 - IA32_FRED_CONFIG: Bit 2, bits 5:4, and bit 11 of the field must be zero. The upper bits of the field must be such that the field’s value is CPU canonical (see Section 4.5).
 - IA32_FRED_RSP1–IA32_FRED_RSP3: The value of each of these fields must be CPU canonical, and bits 5:0 of each field must be zero.
 - IA32_FRED_SSP1–IA32_FRED_SSP3: The value of each of these fields must be CPU canonical, and bits 2:0 of each field must be zero.

29.2.3 Checks on Host Segment and Descriptor-Table Registers

The following checks are performed on fields in the host-state area that correspond to segment and descriptor-table registers:

- In the selector field for each of CS, SS, DS, ES, FS, GS, and TR, the RPL (bits 1:0) and the TI flag (bit 2) must be 0.
- The selector fields for CS and TR cannot be 0000H.

1. The bits corresponding to CR0.NW (bit 29) and CR0.CD (bit 30) are never checked because the values of these bits are not changed by VM exit; see Section 30.5.1.

2. 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.

- 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.

29.2.4 Checks Related to Address-Space Size

On processors that support Intel 64 architecture, the following checks related to address-space size are performed on VMX controls and fields in the host-state area:

- If the logical processor is outside IA-32e mode (if `IA32_EFER.LMA = 0`) at the time of VM entry, the following must hold:
 - The “IA-32e mode guest” VM-entry control is 0.
 - The “host address-space size” VM-exit control is 0.
- If the logical processor is in IA-32e mode (if `IA32_EFER.LMA = 1`) at the time of VM entry, the “host address-space size” VM-exit control must be 1.
- If the “host address-space size” VM-exit control is 0, the following must hold:
 - The “IA-32e mode guest” VM-entry control is 0.
 - Bit 17 of the CR4 field (corresponding to CR4.PCIDE) is 0.
 - Bit 32 of the CR4 field (corresponding to CR4.FRED) is 0.
 - Bits 63:32 in the RIP field are 0.
 - If the “load CET state” VM-exit control is 1, bits 63:32 in the `IA32_S_CET` field and in the SSP field are 0.
- If the “host address-space size” VM-exit control is 1, the following must hold:
 - Bit 5 of the CR4 field (corresponding to CR4.PAE) is 1.
 - The RIP field contains a canonical address.
 - If the “load CET state” VM-exit control is 1, the `IA32_S_CET` field and the SSP field contain canonical addresses.
- If the “load CET state” VM-exit control is 1, the `IA32_INTERRUPT_SSP_TABLE_ADDR` field contains a canonical address.

On processors that do not support Intel 64 architecture, checks are performed to ensure that the “IA-32e mode guest” VM-entry control and the “host address-space size” VM-exit control are both 0.

29.3 CHECKING AND LOADING GUEST STATE

If all checks on the VMX controls and the host-state area pass (see Section 29.2), the following operations take place concurrently: (1) the guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures; (2) processor state is loaded from the guest-state area or as specified by the VM-entry control fields; and (3) address-range monitoring is cleared.

Because the checking and the loading occur concurrently, a failure may be discovered only after some state has been loaded. For this reason, the logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 29.8.

29.3.1 Checks on the Guest State Area

This section describes checks performed on fields in the guest-state area. These checks may be performed in any order. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The following subsections reference fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

29.3.1.1 Checks on Guest Control Registers, Debug Registers, and MSRs

The following checks are performed on fields in the guest-state area corresponding to control registers, debug registers, and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 26.8). The following are exceptions:
 - Bit 0 (corresponding to CR0.PE) and bit 31 (PG) are not checked if the “unrestricted guest” VM-execution control is 1.¹
 - Bit 29 (corresponding to CR0.NW) and bit 30 (CD) are never checked because the values of these bits are not changed by VM entry; see Section 29.3.2.1.
- If bit 31 in the CR0 field (corresponding to PG) is 1, bit 0 in that field (PE) must also be 1.²
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 26.8).
- If bit 23 in the CR4 field (corresponding to CET) is 1, bit 16 in the CR0 field (WP) must also be 1.
- If the “load debug controls” VM-entry control is 1, bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally.
- The following checks are performed on processors that support Intel 64 architecture:
 - If the “IA-32e mode guest” VM-entry control is 1, bit 31 in the CR0 field (corresponding to CR0.PG) and bit 5 in the CR4 field (corresponding to CR4.PAE) must each be 1.³
 - If the “IA-32e mode guest” VM-entry control is 0, bit 17 in the CR4 field (corresponding to CR4.PCIDE) must be 0.
 - If the “IA-32e mode guest” VM-entry control is 0, bit 32 in the CR4 field (corresponding to CR4.FRED) must be 0.
 - The CR3 field must be such that bits reserved in CR3 are 0.⁴ In particular, the field’s width is limited by MAXPHYADDR, a value derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If the “enable EPT” VM-execution control is 0 (implying that the logical processor is not in SEAM) and IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36]. The CR3 field should not set any of bits 63:MAXPHYADDR.
 - If the “load debug controls” VM-entry control is 1, bits 63:32 in the DR7 field must be 0. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally (if they supported Intel 64 architecture).
 - The IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
 - If the “load CET state” VM-entry control is 1, the IA32_S_CET field and the IA32_INTERRUPT_SSP_TABLE_ADDR field must contain canonical addresses.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 22-3).

1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 27.6.2.

2. If the capability MSR IA32_VMX_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.

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_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 36-6).
- If the “load CET state” VM-entry control is 1, the IA32_S_CET field must not set any bits reserved in the IA32_S_CET MSR, and bit 10 (corresponding to SUPPRESS) and bit 11 (TRACKER) of the field cannot both be set.
- If the “load guest IA32_LBR_CTL” VM-entry control is 1, bits reserved in the IA32_LBR_CTL MSR must be 0 in the field for that register.
- If the “load PKRS” VM-entry control is 1, bits 63:32 must be 0 in the IA32_PKRS field.
- If the “load UINV” VM-entry control is 1, bits 15:8 must be 0 in the guest UINV field.
- If the “load FRED” VM-entry control is 1, the following must hold for the indicated fields:
 - IA32_FRED_CONFIG: Bit 2, bits 5:4, and bit 11 of the field must be zero. The upper bits of the field must be such that the field’s value is CPU canonical (see Section 4.5).
 - IA32_FRED_RSP1–IA32_FRED_RSP3: The value of each of these fields must be CPU canonical, and bits 5:0 of each field must be zero.
 - IA32_FRED_SSP1–IA32_FRED_SSP3: The value of each of these fields must be CPU canonical, and bits 2:0 of each field must be zero.

29.3.1.2 Checks on Guest Segment Registers

This section specifies the checks on the fields for CS, SS, DS, ES, FS, GS, TR, and LDTR. The following terms are used in defining these checks:

- The guest will be **virtual-8086** if the VM flag (bit 17) is 1 in the RFLAGS field in the guest-state area.
- The guest will be **IA-32e mode** if the “IA-32e mode guest” VM-entry control is 1. (This is possible only on processors that support Intel 64 architecture.)
- The guest will **use FRED transitions** if the FRED bit (bit 32) is 1 in the CR4 field in the guest-state area.
- Any one of these registers is said to be **usable** if the unusable bit (bit 16) is 0 in the access-rights field for that register.

The following are the checks on these fields:

- Selector fields.
 - TR. The TI flag (bit 2) must be 0.
 - LDTR. If LDTR is usable, the TI flag (bit 2) must be 0.
 - SS. If the guest will not be virtual-8086 and the “unrestricted guest” VM-execution control is 0, the RPL (bits 1:0) must equal the RPL of the selector field for CS.²

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- Base-address fields.
 - CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the address must be the selector field shifted left 4 bits (multiplied by 16).
 - The following checks are performed on processors that support Intel 64 architecture:
 - TR, FS, GS. The address must be canonical.
 - LDTR. If LDTR is usable, the address must be canonical.
 - CS. Bits 63:32 of the address must be zero.
 - SS, DS, ES. If the register is usable, bits 63:32 of the address must be zero.
- Limit fields for CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the field must be 0000FFFFH.
- Access-rights fields.
 - CS, SS, DS, ES, FS, GS.
 - If the guest will be virtual-8086, the field must be 000000F3H. This implies the following:
 - Bits 3:0 (Type) must be 3, indicating an expand-up read/write accessed data segment.
 - Bit 4 (S) must be 1.
 - Bits 6:5 (DPL) must be 3.
 - Bit 7 (P) must be 1.
 - Bits 11:8 (reserved), bit 12 (software available), bit 13 (reserved/L), bit 14 (D/B), bit 15 (G), bit 16 (unusable), and bits 31:17 (reserved) must all be 0.
 - If the guest will not be virtual-8086, the different sub-fields are considered separately:
 - Bits 3:0 (Type).
 - CS. The values allowed depend on the setting of the “unrestricted guest” VM-execution control:
 - If the control is 0, the Type must be 9, 11, 13, or 15 (accessed code segment).
 - If the control is 1, the Type must be either 3 (read/write accessed expand-up data segment) or one of 9, 11, 13, and 15 (accessed code segment).
 - SS. If SS is usable, the Type must be 3 or 7 (read/write, accessed data segment).
 - DS, ES, FS, GS. The following checks apply if the register is usable:
 - Bit 0 of the Type must be 1 (accessed).
 - If bit 3 of the Type is 1 (code segment), then bit 1 of the Type must be 1 (readable).
 - Bit 4 (S). If the register is CS or if the register is usable, S must be 1.
 - Bits 6:5 (DPL).
 - CS.
 - If the Type is 3 (read/write accessed expand-up data segment), the DPL must be 0. The Type can be 3 only if the “unrestricted guest” VM-execution control is 1.
 - If the Type is 9 or 11 (non-conforming code segment), the DPL must equal the DPL in the access-rights field for SS.
 - If the Type is 13 or 15 (conforming code segment), the DPL cannot be greater than the DPL in the access-rights field for SS.
 - SS.
 - If the “unrestricted guest” VM-execution control is 0, the DPL must equal the RPL from the selector field.

2. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 27.6.2.

- The DPL must be 0 either if the Type in the access-rights field for CS is 3 (read/write accessed expand-up data segment) or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.¹
- The following checks apply if the guest will use FRED transitions:
 - The DPL must be 0 or 3.
 - If the DPL is 0, the L bit in the CS access-rights field (bit 13) must be 1.
 - If the DPL is 3, the IOPL value in the RFLAGS field (bits 13:12) and the “blocking by STI” bit in the interruptibility-state field (bit 0) must both be 0.
- DS, ES, FS, GS. The DPL cannot be less than the RPL in the selector field if (1) the “unrestricted guest” VM-execution control is 0; (2) the register is usable; and (3) the Type in the access-rights field is in the range 0 – 11 (data segment or non-conforming code segment).
- Bit 7 (P). If the register is CS or if the register is usable, P must be 1.
- Bits 11:8 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
- Bit 14 (D/B). For CS, D/B must be 0 if the guest will be IA-32e mode and the L bit (bit 13) is 1.
- Bit 15 (G). The following checks apply if the register is CS or if the register is usable:
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
- Bits 31:17 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
- TR. The different sub-fields are considered separately:
 - Bits 3:0 (Type).
 - If the guest will not be IA-32e mode, the Type must be 3 (16-bit busy TSS) or 11 (32-bit busy TSS).
 - If the guest will be IA-32e mode, the Type must be 11 (64-bit busy TSS).
 - Bit 4 (S). S must be 0.
 - Bit 7 (P). P must be 1.
 - Bits 11:8 (reserved). These bits must all be 0.
 - Bit 15 (G).
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
 - Bit 16 (Unusable). The unusable bit must be 0.
 - Bits 31:17 (reserved). These bits must all be 0.
- LDTR. The following checks on the different sub-fields apply only if LDTR is usable:
 - Bits 3:0 (Type). The Type must be 2 (LDT).
 - Bit 4 (S). S must be 0.
 - Bit 7 (P). P must be 1.
 - Bits 11:8 (reserved). These bits must all be 0.
 - Bit 15 (G).
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.

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.

- Bits 31:17 (reserved). These bits must all be 0.

29.3.1.3 Checks on Guest Descriptor-Table Registers

The following checks are performed on the fields for GDTR and IDTR:

- On processors that support Intel 64 architecture, the base-address fields must contain canonical addresses.
- Bits 31:16 of each limit field must be 0.

29.3.1.4 Checks on Guest RIP, RFLAGS, and SSP

The following checks are performed on fields in the guest-state area corresponding to RIP, RFLAGS, and SSP (shadow-stack pointer):

- RIP. The following checks are performed on processors that support Intel 64 architecture:
 - Bits 63:32 must be 0 if the “IA-32e mode guest” VM-entry control is 0 or if the L bit (bit 13) in the access-rights field for CS is 0.
 - If the processor supports $N < 64$ linear-address bits, bits 63:N must be identical if the “IA-32e mode guest” VM-entry control is 1 and the L bit in the access-rights field for CS is 1.¹ (No check applies if the processor supports 64 linear-address bits.) The guest RIP value is not required to be canonical; the value of bit N-1 may differ from that of bit N.
- RFLAGS.
 - Reserved bits 63:22 (bits 31:22 on processors that do not support Intel 64 architecture), bit 15, bit 5 and bit 3 must be 0 in the field, and reserved bit 1 must be 1.
 - The VM flag (bit 17) must be 0 either if the “IA-32e mode guest” VM-entry control is 1 or if bit 0 in the CRO field (corresponding to CRO.PE) is 0.²
 - The IF flag (RFLAGS[bit 9]) must be 1 if the valid bit (bit 31) in the injected-event identification field is 1 and the event type (bits 10:8) is external interrupt.
- SSP. The following checks are performed if the “load CET state” VM-entry control is 1
 - Bits 1:0 must be 0.
 - If the processor supports the Intel 64 architecture, bits 63:N must be identical, where N is the CPU’s maximum linear-address width. (This check does not apply if the processor supports 64 linear-address bits.) The guest SSP value is not required to be canonical; the value of bit N-1 may differ from that of bit N.

29.3.1.5 Checks on Guest Non-Register State

The following checks are performed on fields in the guest-state area corresponding to non-register state:

- Activity state.
 - The activity-state field must contain a value in the range 0 – 3, indicating an activity state supported by the implementation (see Section 27.4.2). 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).

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

2. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CRO.PE must be 1 in VMX operation, bit 0 in the CRO field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

3. As noted in Section 27.4.1, SS.DPL corresponds to the logical processor’s current privilege level (CPL).

- If the valid bit (bit 31) in the injected-event identification field is 1, the event to be delivered (as defined by event type and vector) must not be one that would normally be blocked while a logical processor is in the activity state corresponding to the contents of the activity-state field. The following items enumerate the events (as specified in the injected-event identification field) whose injection is allowed for the different activity states:
 - Active. Any event is allowed.
 - HLT. The only events allowed are the following:
 - Those with event type external interrupt or non-maskable interrupt (NMI).
 - Those with event type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
 - Those with event type other event and vector 0 (pending MTF VM exit).
 See Table 27-18 in Section 27.8.3 for details regarding the format of the injected-event identification field.
 - Shutdown. Only NMIs and machine-check exceptions are allowed.
 - Wait-for-SIPI. No events are allowed.
- The activity-state field must not indicate the wait-for-SIPI state if the “entry to SMM” VM-entry control is 1.
- Interruptibility state.
 - The reserved bits (bits 31:5) must be 0.
 - The field cannot indicate blocking by both STI and MOV SS (bits 0 and 1 cannot both be 1).
 - Bit 0 (blocking by STI) must be 0 if the IF flag (bit 9) is 0 in the RFLAGS field.
 - Bit 0 (blocking by STI) and bit 1 (blocking by MOV-SS) must both be 0 if the valid bit (bit 31) in the injected-event identification field is 1 and the event type (bits 10:8) in that field has value 0, indicating external interrupt, or value 2, indicating non-maskable interrupt (NMI).
 - Bit 2 (blocking by SMI) must be 0 if the processor is not in SMM.
 - Bit 2 (blocking by SMI) must be 1 if the “entry to SMM” VM-entry control is 1.
 - Bit 3 (blocking by NMI) must be 0 if the “virtual NMIs” VM-execution control is 1, the valid bit (bit 31) in the injected-event identification field is 1, and the event type (bits 10:8) in that field has value 2 (indicating NMI).
 - If bit 4 (enclave interruption) is 1, bit 1 (blocking by MOV-SS) must be 0 and the processor must support for SGX by enumerating CPUID.07H.00H:EBX.SGX[2] as 1.

NOTE

If the “virtual NMIs” VM-execution control is 0, there is no requirement that bit 3 be 0 if the valid bit in the injected-event identification field is 1 and the event type in that field has value 2.

- Pending debug exceptions.
 - Bits 11:4, bit 13, bit 15, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0.
 - The following checks are performed if any of the following holds: (1) the interruptibility-state field indicates blocking by STI (bit 0 in that field is 1); (2) the interruptibility-state field indicates blocking by MOV SS (bit 1 in that field is 1); or (3) the activity-state field indicates HLT:
 - Bit 14 (BS) must be 1 if the TF flag (bit 8) in the RFLAGS field is 1 and the BTF flag (bit 1) in the IA32_DEBUGCTL field is 0.
 - Bit 14 (BS) must be 0 if the TF flag (bit 8) in the RFLAGS field is 0 or the BTF flag (bit 1) in the IA32_DEBUGCTL field is 1.
 - The following checks are performed if bit 16 (RTM) is 1:

- Bits 11:0, bits 15:13, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0; bit 12 must be 1.
- The processor must support for RTM by enumerating CPUID.07H.00H:EBX[11] as 1.
- The interruptibility-state field must not indicate blocking by MOV SS (bit 1 in that field must be 0).
- VMCS link pointer. The following checks apply if the field contains a value other than FFFFFFFF_FFFFFFFFH:
 - Bits 11:0 must be 0.
 - The field's width is limited by MAXPHYADDR, a value derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. The VMCS link pointer should not set bits in the range 63:MAXPHYADDR.
 - The 4 bytes located in memory referenced by the value of the field (as a physical address) must satisfy the following:
 - Bits 30:0 must contain the processor's VMCS revision identifier (see Section 27.2).¹
 - Bit 31 must contain the setting of the "VMCS shadowing" VM-execution control.² This implies that the referenced VMCS is a shadow VMCS (see Section 27.10) if and only if the "VMCS shadowing" VM-execution control is 1.
 - If the processor is not in SMM or the "entry to SMM" VM-entry control is 1, the field must not contain the current VMCS pointer.
 - If the processor is in SMM and the "entry to SMM" VM-entry control is 0, the field must differ from the executive-VMCS pointer.

29.3.1.6 Checks on Guest Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, the logical processor uses **PAE paging** (see Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A).³ When PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTes). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTes.

A VM entry is to a guest that uses PAE paging if (1) bit 31 (corresponding to CR0.PG) is set in the CR0 field in the guest-state area; (2) bit 5 (corresponding to CR4.PAE) is set in the CR4 field; and (3) the "IA-32e mode guest" VM-entry control is 0. Such a VM entry checks the validity of the PDPTes:

- If the "enable EPT" VM-execution control is 0, VM entry checks the validity of the PDPTes referenced by the CR3 field in the guest-state area if either (1) PAE paging was not in use before the VM entry; or (2) the value of CR3 is changing as a result of the VM entry. VM entry may check their validity even if neither (1) nor (2) hold.⁴
- If the "enable EPT" VM-execution control is 1, VM entry checks the validity of the PDPTE fields in the guest-state area (see Section 27.4.2).

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.⁵ If MOV to CR3 would cause a general-protection exception due to the PDPTes that would be loaded (e.g., because a reserved bit is set), the VM entry fails.

-
1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.
 2. "VMCS shadowing" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "VMCS shadowing" VM-execution control were 0. See Section 27.6.2.
 3. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine the number physical-address bits supported by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
 4. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "enable EPT" VM-execution control were 0. See Section 27.6.2.

29.3.2 Loading Guest State

Processor state is updated on VM entries in the following ways:

- Some state is loaded from the guest-state area.
- Some state is determined by VM-entry controls.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order and in parallel with the checking of VMCS contents (see Section 29.3.1).

The loading of guest state is detailed in Section 29.3.2.1 to Section 29.3.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

In addition to the state loading described in this section, VM entries may load MSRs from the VM-entry MSR-load area (see Section 29.4). This loading occurs only after the state loading described in this section and the checking of VMCS contents described in Section 29.3.1.

29.3.2.1 Loading Guest Control Registers, Debug Registers, and MSRs

The following items describe how guest control registers, debug registers, and MSRs are loaded on VM entry:

- CR0 is loaded from the CR0 field with the exception of the following bits, which are never modified on VM entry: ET (bit 4); reserved bits 15:6, 17, and 28:19; NW (bit 29) and CD (bit 30).¹ The values of these bits in the CR0 field are ignored.
- CR3 and CR4 are loaded from the CR3 field and the CR4 field, respectively.
- If the “load debug controls” VM-entry control is 1, DR7 is loaded from the DR7 field with the exception that bit 12 and bits 15:14 are always 0 and bit 10 is always 1. The values of these bits in the DR7 field are ignored. The first processors to support the virtual-machine extensions supported only the 1-setting of the “load debug controls” VM-entry control and thus always loaded DR7 from the DR7 field.
- The following describes how certain MSRs are loaded using fields in the guest-state area:
 - If the “load debug controls” VM-entry control is 1, the IA32_DEBUGCTL MSR is loaded from the IA32_DEBUGCTL field. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always loaded the IA32_DEBUGCTL MSR from the IA32_DEBUGCTL field.
 - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since this field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
 - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
 - The following are performed on processors that support Intel 64 architecture:
 - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 29.3.2.2).
 - If the “load IA32_EFER” VM-entry control is 0, bits in the IA32_EFER MSR are modified as follows:
 - IA32_EFER.LMA is loaded with the setting of the “IA-32e mode guest” VM-entry control.
 - If CR0 is being loaded so that CR0.PG = 1, IA32_EFER.LME is also loaded with the setting of the “IA-32e mode guest” VM-entry control.² Otherwise, IA32_EFER.LME is unmodified.

5. This implies that (1) bits 11:9 in each PDPTE are ignored; and (2) if bit 0 (present) is clear in one of the PDPTEs, bits 63:1 of that PDPTE are ignored.

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.

See below for the case in which the “load IA32_EFER” VM-entry control is 1

- If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field.
- If the “load IA32_PAT” VM-entry control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field.
- If the “load IA32_EFER” VM-entry control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field.
- If the “load IA32_BNDCFGS” VM-entry control is 1, the IA32_BNDCFGS MSR is loaded from the IA32_BNDCFGS field.
- If the “load IA32_RTIT_CTL” VM-entry control is 1, the IA32_RTIT_CTL MSR is loaded from the IA32_RTIT_CTL field.
- If the “load CET” VM-entry control is 1, the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are loaded from the IA32_S_CET field and the IA32_INTERRUPT_SSP_TABLE_ADDR field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
- If the “load guest IA32_LBR_CTL” VM-entry control is 1, the IA32_LBR_CTL MSR is loaded from the IA32_LBR_CTL guest state field.
- If the “load PKRS” VM-entry control is 1, the IA32_PKRS MSR is loaded from the IA32_PKRS field.
- If the “load UINV” VM-entry control is 1, UINV is loaded with the low 8 bits of the UINV field. UINV is represented in bits 39:32 of the IA32_UINTR_MISC MSR. The remainder of the MSR is not modified.
- If the “load FRED” VM-entry control is 1, the following MSRs are loaded from the corresponding fields: IA32_FRED_CONFIG, IA32_FRED_RSP1, IA32_FRED_RSP2, IA32_FRED_RSP3, IA32_FRED_STKLVLs, IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-entry MSR-load area. See Section 29.4.

- The SMBASE register is unmodified by all VM entries except those that return from SMM.

29.3.2.2 Loading Guest Segment Registers and Descriptor-Table Registers

For each of CS, SS, DS, ES, FS, GS, TR, and LDTR, fields are loaded from the guest-state area as follows:

- The unusable bit is loaded from the access-rights field. This bit can never be set for TR (see Section 29.3.1.2). If it is set for one of the other registers, the following apply:
 - For each of CS, SS, DS, ES, FS, and GS, uses of the segment cause faults (general-protection exception or stack-fault exception) outside 64-bit mode, just as they would had the segment been loaded using a null selector. This bit does not cause accesses to fault in 64-bit mode.
 - If this bit is set for LDTR, uses of LDTR cause general-protection exceptions in all modes, just as they would had LDTR been loaded using a null selector.
- If this bit is clear for any of CS, SS, DS, ES, FS, GS, TR, and LDTR, a null selector value does not cause a fault (general-protection exception or stack-fault exception).
- TR. The selector, base, limit, and access-rights fields are loaded.
- CS.
 - The following fields are always loaded: selector, base address, limit, and (from the access-rights field) the L, D, and G bits.
 - For the other fields, the unusable bit of the access-rights field is consulted:
 - If the unusable bit is 0, all of the access-rights field is loaded.
 - If the unusable bit is 1, the remainder of CS access rights are undefined after VM entry.
- SS, DS, ES, FS, GS, and LDTR.
 - The selector fields are loaded.

- For the other fields, the unusable bit of the corresponding access-rights field is consulted:
 - If the unusable bit is 0, the base-address, limit, and access-rights fields are loaded.
 - If the unusable bit is 1, the base address, the segment limit, and the remainder of the access rights are undefined after VM entry with the following exceptions:
 - Bits 3:0 of the base address for SS are cleared to 0.
 - SS.DPL is always loaded from the SS access-rights field. This will be the current privilege level (CPL) after the VM entry completes.
 - SS.B is always set to 1.
 - The base addresses for FS and GS are loaded from the corresponding fields in the VMCS. On processors that support Intel 64 architecture, the values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
 - On processors that support Intel 64 architecture, bits 63:32 of the base addresses for SS, DS, and ES are cleared to 0.

GDTR and IDTR are loaded using the base and limit fields.

29.3.2.3 Loading Guest RIP, RSP, RFLAGS, and SSP

RSP, RIP, and RFLAGS are loaded from the RSP field, the RIP field, and the RFLAGS field, respectively.

If the “load CET” VM-entry control is 1, SSP (shadow-stack pointer) is loaded from the SSP field.

The following items regard the upper 32 bits of these fields on VM entries that are not to 64-bit mode:

- Bits 63:32 of RSP are undefined outside 64-bit mode. Thus, a logical processor may ignore the contents of bits 63:32 of the RSP field on VM entries that are not to 64-bit mode.
- As noted in Section 29.3.1.4, bits 63:32 of the RIP and RFLAGS fields must be 0 on VM entries that are not to 64-bit mode. (The same is true for SSP for VM entries that are not to 64-bit mode when the “load CET” VM-entry control is 1.)

29.3.2.4 Loading Page-Directory-Pointer-Table Entries

As noted in Section 29.3.1.6, the logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0. A VM entry to a guest that uses PAE paging loads the 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 29.3.2.1). 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 27.4.2). The values loaded are treated as guest-physical addresses in VMX non-root operation.

29.3.2.5 Updating Non-Register State

Section 31.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM entries invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP).
- VM entries are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

If the “virtual-interrupt delivery” VM-execution control is 1, VM entry loads the values of RVI and SVI from the guest interrupt-status field in the VMCS (see Section 27.4.2). After doing so, the logical processor first causes PPR virtualization (Section 32.1.3) and then evaluates pending virtual interrupts (Section 32.2.1).

If a virtual interrupt is recognized, it may be delivered in VMX non-root operation immediately after VM entry (including any specified event injection) completes; see Section 29.7.5. See Section 32.2.2 for details regarding the delivery of virtual interrupts.

29.3.3 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 11.10.4 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A. VM entries clear any address-range monitoring that may be in effect.

29.4 LOADING MSRS

VM entries may load MSRs from the VM-entry MSR-load area (see Section 27.8.2). Specifically each entry in that area (up to the number specified in the VM-entry MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.¹

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101 (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM entry did not commence in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM entries for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.²

The VM entry fails if processing fails for any entry. The logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 29.8.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM entry, the logical processor will not use any translations that were cached before the transition.

29.5 TRACE-ADDRESS PRE-TRANSLATION (TAPT)

When the "Intel PT uses guest physical addresses" VM-execution control is 1, the addresses used by Intel PT are treated as guest-physical addresses, and these are translated to physical addresses using EPT.

VM entry uses **trace-address pre-translation (TAPT)** to prevent buffered trace data from being lost due to an EPT violation; see Section 28.5.4.2. VM entry uses TAPT only if Intel PT will be enabled following VM entry (IA32_RTIT_CTL.TraceEn = 1) and only if the "Intel PT uses guest physical addresses" VM-execution control is 1.

As noted in Section 28.5.4, TAPT may cause a VM exit due to an EPT violation, EPT misconfiguration, page-modification log-full event, or APIC access. If such a VM exit occurs as a result of TAPT during VM entry, the VM exit operates as if it had occurred in VMX non-root operation after the VM entry completed (in the guest context).

-
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.

If TAPT during VM entry causes a VM exit, the VM entry does not perform event injection (Section 29.6), even if the valid bit in the injected-event identification field is 1. Such VM exits save the contents of injected-event identification and injected-event error code fields into the original-event identification and original-event error code fields, respectively.

29.6 EVENT INJECTION

If the valid bit in the injected-event identification field (see Section 27.8.3) is 1, VM entry causes an event to be delivered (or, in one case, made pending) after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.

- If the event type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception), the event is delivered as described in Section 29.6.1. This is done also if the event type is 7 (other event) and the vector field is 1 (SYSCALL) or 2 (SYSENTER).
- If the event type in the field is 7 (other event) and the vector field is 0, an MTF VM exit is pending after VM entry. See Section 29.6.2.

29.6.1 Injection of Interrupts and Exceptions

VM entry delivers an injected event within the guest context established by VM entry. This means that delivery occurs after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established:¹

- If FRED transitions are not enabled, the event is delivered using the vector in that field to select a descriptor in the IDT. (Since event injection occurs after loading IDTR from the guest-state area, this is the guest IDT.)
- If FRED transitions are enabled, the event is delivered using FRED event delivery. (Since event injection occurs after any optional loading of the MSRs that control FRED transitions, those MSRs control this event delivery.)

Section 29.6.1.1 provides details of event injection. In general, the event is delivered exactly as if it had been generated normally.

An exception is made if the following all hold: bit 25 (UINTR) is set to 1 in the guest CR4 field and the “IA-32e mode guest” VM-entry control is 1, and VM entry is injecting an external interrupt whose vector is the value that UINV would have after VM entry. In this case, the logical processor does not deliver the interrupt but instead performs user-interrupt notification processing as specified in Section 9.5.2. (If the guest activity-state field indicated the HLT state, the logical processor enters the HLT state following user-interrupt notification processing.)

If event delivery (or user-interrupt notification processing; see above) encounters a nested exception (for example, a general-protection exception because the vector indicates a descriptor beyond the IDT limit), the exception bitmap is consulted using the vector of that exception:

- If the bit for the nested exception is 0, the nested exception is handled normally. If the nested exception is benign, it is delivered normally. If it is contributory or a page fault, a double fault may be generated, depending on the nature of the event whose delivery encountered the nested exception. See Chapter 7, “Interrupt 8—Double Fault Exception (#DF)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.²
- If the bit for the nested exception is 1, a VM exit occurs. Section 29.6.1.2 details cases in which event injection causes a VM exit.

1. This does not imply that injection of an exception or interrupt will cause a VM exit due to the settings of VM-execution control fields (such as the exception bitmap) that would cause a VM exit if the event had occurred in VMX non-root operation. In contrast, a nested exception encountered during event delivery may cause a VM exit; see Section 29.6.1.1.

2. Hardware exceptions with the following unused vectors are considered benign: 15 and 21–31. A hardware exception with vector 20 is considered benign unless the processor supports the 1-setting of the “EPT-violation #VE” VM-execution control; in that case, it has the same severity as page faults.

29.6.1.1 Details of Event Injection

The event-injection process is controlled by the contents of the injected-event identification field (format given in Table 27-18), the injected-event error-code field, the injected-event data field (for FRED event delivery), and the VM-entry instruction-length field. The following items provide details of the process:

- The value pushed on the stack for RFLAGS is generally that which was loaded from the guest-state area. The value pushed for the RF flag is not modified based on the type of event being delivered. However, the pushed value of RFLAGS may be modified if a software interrupt is being injected into a guest that will be in virtual-8086 mode (see below). After RFLAGS is pushed on the stack, the value in the RFLAGS register is modified as is done normally when delivering an event.
 - The instruction pointer that is pushed on the stack depends on the type of event and whether nested exceptions occur during its delivery. The term **current guest RIP** refers to the value to be loaded from the guest-state area. The value pushed is determined as follows:¹
 - If VM entry successfully injects (with no nested exception) an event with event type external interrupt, NMI, or hardware exception, the current guest RIP is pushed on the stack.
 - If VM entry successfully injects (with no nested exception) an event with event type software interrupt, privileged software exception, or software exception, the current guest RIP is incremented by the VM-entry instruction length before being pushed on the stack. This item applies also if VM entry is injecting SYSCALL or SYSENTER (with FRED event delivery).
 - If VM entry encounters an exception while injecting an event and that exception does not cause a VM exit, the current guest RIP is pushed on the stack regardless of event type or VM-entry instruction length. If the encountered exception does cause a VM exit that saves RIP, the saved RIP is current guest RIP.
 - If the deliver-error-code bit (bit 11) is set in the injected-event identification field, the contents of the injected-event error-code field is pushed on the stack as an error code would be pushed during delivery of an exception. If this bit is clear, FRED event delivery pushes zero for an error code.
 - DR6, DR7, and the IA32_DEBUGCTL MSR are not modified by event injection, even if the event has vector 1 (normal deliveries of debug exceptions, which have vector 1, do update these registers).
 - If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode (RFLAGS.VM = 1), no general-protection exception can occur due to RFLAGS.IOPL < 3. A VM monitor should check RFLAGS.IOPL before injecting such an event and, if desired, inject a general-protection exception instead of a software interrupt.
 - If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode with virtual-8086 mode extensions (RFLAGS.VM = CR4.VME = 1), event delivery is subject to VME-based interrupt redirection based on the software interrupt redirection bitmap in the task-state segment (TSS) as follows:
 - If bit n in the bitmap is clear (where n is the number of the software interrupt), the interrupt is directed to an 8086 program interrupt handler: the processor uses a 16-bit interrupt-vector table (IVT) located at linear address zero. If the value of RFLAGS.IOPL is less than 3, the following modifications are made to the value of RFLAGS that is pushed on the stack: IOPL is set to 3, and IF is set to the value of VIF.
 - If bit n in the bitmap is set (where n is the number of the software interrupt), the interrupt is directed to a protected-mode interrupt handler. (In other words, the injection is treated as described in the next item.) In this case, the software interrupt does not invoke such a handler if RFLAGS.IOPL < 3 (a general-protection exception occurs instead). However, as noted above, RFLAGS.IOPL cannot cause an injected software interrupt to cause such an exception. Thus, in this case, the injection invokes a protected-mode interrupt handler independent of the value of RFLAGS.IOPL.
- Injection of events of other types are not subject to this redirection.
- If VM entry is injecting a software interrupt (not redirected as described above) or software exception, privilege checking is performed on the IDT descriptor being accessed as would be the case for executions of INT n , INT3, or INTO (the descriptor's DPL cannot be less than CPL). There is no checking of RFLAGS.IOPL, even if the guest will be in virtual-8086 mode. Failure of this check may lead to a nested exception. Injection of an event with event type external interrupt, NMI, hardware exception, and privileged software exception, or with event type software interrupt and being redirected as described above, do not perform these checks.

1. 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 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 event type external interrupt, NMI, hardware exception, or privileged software exception and encounters a nested exception (but does not produce a double fault), the error code for that exception sets the EXT bit.
 - If event being injected is a software interrupt, a software exception, or (for FRED event delivery) SYSCALL or SYSENTER, and encounters a nested exception, the error code for that exception clears the EXT bit.
 - If event delivery encounters a nested exception and delivery of that exception encounters another exception (but does not produce a double fault), the error code for that exception sets the EXT bit.
 - If a double fault is produced, the error code for the double fault is 0000H (the EXT bit is clear).
- The following items apply specifically when the injected event is delivered with FRED event delivery (because FRED transitions will be enabled after VM entry):
 - Parts of FRED event delivery depend on the CPL at the time the event occurred. For an event injected by VM entry, the value used for CPL is that which VM entry loaded for DPL in the access-rights field for SS.
 - If bit 13 of the injected-event identification field is 1, FRED event delivery determines stack level as if the event being delivered is an exception that was encountered during delivery of another event. (Otherwise, it does not). FRED event delivery saves the value of this bit for the “nested exception” indication in the event information saved on the stack (specifically, this is bit 58 of the augmented SS field on the stack). This bit is not used for events injected with IDT event delivery.
 - The stack frame established by FRED event delivery includes 64 bits of event data that is defined for certain events stack. For an event injected by VM entry, the event data saved is the value of the injected-event-data field in the VMCS. This is done for all injected events (even those that do not normally define event data).
 - FRED event delivery of certain events saves an instruction length as part of event information. For such an event injected by VM entry, the instruction length saved is the value of the VM-entry instruction-length field.
 - FRED event delivery saves as part of event information a bit that indicates whether event occurred while the logical processor was in enclave mode. For an event injected by VM entry, the value saved is that of bit 4 of the guest interruptibility-state field (which indicates “enclave interruption”).
 - The following items provide details of how FRED event delivery of an event injected by VM entry sets bits in the augmented CS and the augmented SS saved on the stack:¹
 - Bits 17:16 of the augmented CS (stack level) are set to a value that depends on the CPL from which the event is being delivered (see above for how this CPL is determined):
 - If the CPL is 0, bits 17:16 are set to the value that would have been in IA32_FRED_CONFIG[1:0] had VM entry completed without injecting an event.²
 - If the CPL is 3, bits 17:16 are zero.
 - Bit 18 of the augmented CS (indirect branch tracker) is set to a value that depends on the CPL from which the event is being delivered and the value of bit 23 (CET) in the guest CR4 field:
 - If the CPL is 3 or the guest CR4.CET is 0, bit 18 is 0.

1. If supervisor shadow stacks will be enabled, the same augmented CS is pushed on the shadow stack;

2. If the “load FRED” VM-entry control is 1, this will be bits 1:0 of the IA32_FRED_CONFIG field, unless the IA32_FRED_CONFIG MSR was subsequently loaded from the VM-entry MSR-load area (in which case it will be bits 1:0 of the value loaded from memory).

- If the CPL is 0 and the guest CR4.CET is 1, the bit is set to the value of IA32_S_CET.ENDBR AND IA32_S_CET.TRACKER, referring to the value that the IA32_S_CET MSR will have following VM entry.
- Bit 16 of the augmented SS (blocking by STI) is set to the value of bit 0 of the guest interruptibility-state (that bit indicates blocking by STI). (There is no blocking by STI after event injection.)
- Bit 17 of the augmented SS (software interrupt or system call) is set if and only if the value of bits 10:8 in the injected-event identification field is 4 (indicating event type software interrupt) or 7 (indicating event type “other event,” when used for SYSCALL and SYSENTER).¹
- Bit 18 of the augmented SS (NMI) is set if and only if the value of bits 10:8 in the injected-event identification field in the VMCS is 2 (indicating event type NMI).

29.6.1.2 VM Exits During Event Injection

An event being injected never causes a VM exit directly regardless of the settings of the VM-execution controls. For example, setting the “NMI exiting” VM-execution control to 1 does not cause a VM exit due to injection of an NMI.

However, the event-delivery process may lead to a VM exit:

- If the vector in the injected-event identification field identifies a task gate in the IDT, the attempted task switch may cause a VM exit just as it would had the injected event occurred during normal execution in VMX non-root operation (see Section 28.4.2).
- If event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap (see Section 28.2).
- If event delivery generates a double-fault exception (due to a nested exception); the logical processor encounters another nested exception while attempting to call the double-fault handler; and that exception does not cause a VM exit due to the exception bitmap; then a VM exit occurs due to triple fault (see Section 28.2).
- If event delivery injects a double-fault exception and encounters a nested exception that does not cause a VM exit due to the exception bitmap, then a VM exit occurs due to triple fault (see Section 28.2).
- If the “virtualize APIC accesses” VM-execution control is 1 and event delivery generates an access to the APIC-access page, that access is treated as described in Section 32.4 and may cause a VM exit.²

If the event-delivery process does cause a VM exit, the processor state before the VM exit is determined just as it would be had the injected event occurred during normal execution in VMX non-root operation. If the injected event directly accesses a task gate that cause a VM exit or if the first nested exception encountered causes a VM exit, information about the injected event is saved in the original-event identification field (see Section 30.2.4).

The material in this section applies also if injection of an external interrupt results in user-interrupt notification processing instead of event delivery (see Section 29.6.1 earlier).

29.6.1.3 Event Injection for VM Entries to Real-Address Mode

If VM entry is loading CR0.PE with 0, any injected event is delivered as would normally be done in real-address mode.³ Specifically, VM entry uses the vector provided in the injected-event identification field to select a 4-byte entry from an interrupt-vector table at the linear address in IDTR.base. Further details are provided in Section 16.1.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Because bit 11 (deliver error code) in the injected-event identification field must be 0 if CR0.PE will be 0 after VM entry (see Section 29.2.1.3), events injected with CR0.PE = 0 do not push an error code on the stack. This is consistent with event delivery in real-address mode.

1. Event type 7 may also be used for pending MTF VM exits, but these are not delivered using FRED event delivery.

2. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 27.6.2.

3. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PE must be 1 in VMX operation, VM entry must be loading CR0.PE with 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

If event delivery encounters a fault (due to a violation of IDTR.limit or of SS.limit), the fault is treated as if it had occurred during event delivery in VMX non-root operation. Such a fault may lead to a VM exit as discussed in Section 29.6.1.2.

29.6.2 Injection of Pending MTF VM Exits

If the event type in the injected-event identification field is 7 (other event) and the vector field is 0, VM entry causes an MTF VM exit to be pending on the instruction boundary following VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0. See Section 28.5.2 for the treatment of pending MTF VM exits.

29.7 SPECIAL FEATURES OF VM ENTRY

This section details a variety of features of VM entry. It uses the following terminology: a VM entry is **injecting** if the valid bit (bit 31) of the injected-event identification is 1 and the event type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception).

29.7.1 Interruptibility State

The interruptibility-state field in the guest-state area (see Table 27-3) contains bits that control blocking by STI, blocking by MOV SS, and blocking by NMI. This field impacts event blocking after VM entry as follows:

- If the VM entry is injecting, there is no blocking by STI or by MOV SS following the VM entry, regardless of the contents of the interruptibility-state field.
 - If the VM entry is not injecting, the following apply:
 - Events are blocked by STI if and only if bit 0 in the interruptibility-state field is 1. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry; see Section 29.7.3).
 - Events are blocked by MOV SS if and only if bit 1 in the interruptibility-state field is 1. This may affect the treatment of pending debug exceptions; see Section 29.7.3. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry).
 - The blocking of non-maskable interrupts (NMIs) is determined as follows:
 - If the “virtual NMIs” VM-execution control is 0, NMIs are blocked if and only if bit 3 (blocking by NMI) in the interruptibility-state field is 1. If the “NMI exiting” VM-execution control is 0, execution of the IRET instruction removes this blocking (even if the instruction generates a fault). If the “NMI exiting” control is 1, IRET does not affect this blocking.
 - The following items describe the use of bit 3 (blocking by NMI) in the interruptibility-state field if the “virtual NMIs” VM-execution control is 1:
 - The bit’s value does not affect the blocking of NMIs after VM entry. NMIs are not blocked in VMX non-root operation (except for ordinary blocking for other reasons, such as by the MOV SS instruction, the wait-for-SIPI state, etc.)
 - The bit’s value determines whether there is virtual-NMI blocking after VM entry. If the bit is 1, virtual-NMI blocking is in effect after VM entry. If the bit is 0, there is no virtual-NMI blocking after VM entry unless the VM entry is injecting an NMI (see Section 29.6.1.1). Execution of IRET removes virtual-NMI blocking (even if the instruction generates a fault).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 29.2.1.1.
- Blocking of system-management interrupts (SMIs) is determined as follows:
 - If the VM entry was not executed in system-management mode (SMM), SMI blocking is unchanged by VM entry.
 - If the VM entry was executed in SMM, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.

29.7.2 Activity State

The activity-state field in the guest-state area controls whether, after VM entry, the logical processor is active or in one of the inactive states identified in Section 27.4.2. The use of this field is determined as follows:

- If the VM entry is injecting, the logical processor is in the active state after VM entry. While the consistency checks described in Section 29.3.1.5 on the activity-state field do apply in this case, the contents of the activity-state field do not determine the activity state after VM entry.
- If the VM entry is not injecting, the logical processor ends VM entry in the activity state specified in the guest-state area. If VM entry ends with the logical processor in an inactive activity state, the VM entry generates any special bus cycle that is normally generated when that activity state is entered from the active state. If VM entry would end with the logical processor in the shutdown state and the logical processor is in SMX operation,¹ an Intel® TXT shutdown condition occurs. See the Intel® Trusted Execution Technology Measured Launched Environment Developer's Guide. The error code used is 0000H, indicating "legacy shutdown."
- Some activity states unconditionally block certain events. The following blocking is in effect after any VM entry that puts the processor in the indicated state:
 - The active state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the active state and in VMX non-root operation are discarded and do not cause VM exits.
 - The HLT state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the HLT state and in VMX non-root operation are discarded and do not cause VM exits.
 - The shutdown state blocks external interrupts and SIPIs. External interrupts that arrive while a logical processor is in the shutdown state and in VMX non-root operation do not cause VM exits even if the "external-interrupt exiting" VM-execution control is 1. SIPIs that arrive while a logical processor is in the shutdown state and in VMX non-root operation are discarded and do not cause VM exits.
 - The wait-for-SIPI state blocks external interrupts, non-maskable interrupts (NMIs), INIT signals, and system-management interrupts (SMIs). Such events do not cause VM exits if they arrive while a logical processor is in the wait-for-SIPI state and in VMX non-root operation.

29.7.3 Delivery of Pending Debug Exceptions after VM Entry

The pending debug exceptions field in the guest-state area indicates whether there are debug exceptions that have not yet been delivered (see Section 27.4.2). 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 injecting using FRED event delivery or with one of the following event types: external interrupt, non-maskable interrupt (NMI), hardware exception, or privileged software exception.
- The interruptibility-state field does not indicate blocking by MOV SS and the VM entry is injecting with either of the following event types: software interrupt or software exception.
- The VM entry is not injecting and the activity-state field indicates either shutdown or wait-for-SIPI.

If none of the above hold, the pending debug exceptions field specifies the debug exceptions that are pending for the guest. There are **valid pending debug exceptions** if either the BS bit (bit 14) or the enable-breakpoint bit (bit 12) is 1. If there are valid pending debug exceptions, they are handled as follows:

- If the VM entry is not injecting, the pending debug exceptions are treated as they would had they been encountered normally in guest execution:
 - If the logical processor is not blocking such exceptions (the interruptibility-state field indicates no blocking by MOV SS), a debug exception is delivered after VM entry (see below).
 - If the logical processor is blocking such exceptions (due to blocking by MOV SS), the pending debug exceptions are held pending or lost as would normally be the case.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. See Chapter 7, "Safer Mode Extensions Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

- If the VM entry is injecting (using IDT event delivery, with event type software interrupt or software exception, and with blocking by MOV SS), the following items apply:
 - For injection of a software interrupt or of a software exception with vector 3 (#BP) or vector 4 (#OF) — or a privileged software exception with vector 1 (#DB) — the pending debug exceptions are treated as they would had they been encountered normally in guest execution if the corresponding instruction (INT1, INT3, or INTO) were executed after a MOV SS that encountered a debug trap.
 - For injection of a software exception with a vector other than 3 and 4, the pending debug exceptions may be lost or they may be delivered after injection (see below).

If there are no valid pending debug exceptions (as defined above), no pending debug exceptions are delivered after VM entry.

If a pending debug exception is delivered after VM entry, it has the priority of “traps on the previous instruction” (see Section 7.9 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). Thus, INIT signals and system-management interrupts (SMIs) take priority of such an exception, as do VM exits induced by the TPR threshold (see Section 29.7.7) and pending MTF VM exits (see Section 29.7.8). The exception takes priority over any pending non-maskable interrupt (NMI) or external interrupt and also over VM exits due to the 1-settings of the “interrupt-window exiting” and “NMI-window exiting” VM-execution controls.

FRED event delivery saves as part of event information a bit that indicates whether event occurred while the logical processor was in enclave mode. For a pending debug exception is delivered after VM entry, the value saved is that of bit 4 of the guest interruptibility-state field (which indicates “enclave interruption”).

A pending debug exception delivered after VM entry causes a VM exit if the bit 1 (#DB) is 1 in the exception bitmap. If it does not cause a VM exit, it updates DR6 normally.

29.7.4 VMX-Preemption Timer

If the “activate VMX-preemption timer” VM-execution control is 1, VM entry starts the VMX-preemption timer with the unsigned value in the VMX-preemption timer-value field.

It is possible for the VMX-preemption timer to expire during VM entry (e.g., if the value in the VMX-preemption timer-value field is zero). If this happens (and if the VM entry was not to the wait-for-SIPI state), a VM exit occurs with its normal priority after any event injection and before execution of any instruction following VM entry. For example, any pending debug exceptions established by VM entry (see Section 29.7.3) take priority over a timer-induced VM exit. (The timer-induced VM exit will occur after delivery of the debug exception, unless that exception or its delivery causes a different VM exit.)

See Section 28.5.1 for details of the operation of the VMX-preemption timer in VMX non-root operation, including the blocking and priority of the VM exits that it causes.

29.7.5 Interrupt-Window Exiting and Virtual-Interrupt Delivery

If “interrupt-window exiting” VM-execution control is 1, an open interrupt window may cause a VM exit immediately after VM entry (see Section 28.2 for details). If the “interrupt-window exiting” VM-execution control is 0 but the “virtual-interrupt delivery” VM-execution control is 1, a virtual interrupt may be delivered immediately after VM entry (see Section 29.3.2.5 and Section 32.2.1).

The following items detail the treatment of these events:

- These events occur after any event injection specified for VM entry.
- Non-maskable interrupts (NMIs) and higher priority events take priority over these events. These events take priority over external interrupts and lower priority events.
- These events wake the logical processor if it just entered the HLT state because of a VM entry (see Section 29.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

29.7.6 NMI-Window Exiting

The “NMI-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 28.2 for details).

The following items detail the treatment of these VM exits:

- These VM exits follow event injection if such injection is specified for VM entry.
- Debug-trap exceptions (see Section 29.7.3) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.
- VM exits caused by this control wake the logical processor if it just entered either the HLT state or the shutdown state because of a VM entry (see Section 29.7.2). They do not occur if the logical processor just entered the wait-for-SIPI state.

29.7.7 VM Exits Induced by the TPR Threshold

If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1 and the “virtual-interrupt delivery” VM-execution control is 0, a VM exit occurs immediately after VM entry if the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 of VTPR (see Section 32.1.1).¹

The following items detail the treatment of these VM exits:

- The VM exits are not blocked if RFLAGS.IF = 0 or by the setting of bits in the interruptibility-state field in guest-state area.
 - The VM exits follow event injection if such injection is specified for VM entry.
 - VM exits caused by this control take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. They thus have priority over the VM exits described in Section 29.7.5, Section 29.7.6, and Section 29.7.8, as well as any interrupts or debug exceptions that may be pending at the time of VM entry.
 - These VM exits wake the logical processor if it just entered the HLT state as part of a VM entry (see Section 29.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.
- If such a VM exit is suppressed because the processor just entered the shutdown state, it occurs after the delivery of any event that cause the logical processor to leave the shutdown state while remaining in VMX non-root operation (e.g., due to an NMI that occurs while the “NMI-exiting” VM-execution control is 0).
- The basic exit reason is “TPR below threshold.”

29.7.8 Pending MTF VM Exits

As noted in Section 29.6.2, VM entry may cause an MTF VM exit to be pending immediately after VM entry. The following items detail the treatment of these VM exits:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these VM exits. These VM exits take priority over debug-trap exceptions and lower priority events.
- These VM exits wake the logical processor if it just entered the HLT state because of a VM entry (see Section 29.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

29.7.9 VM Entries and Advanced Debugging Features

VM entries are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

1. “Virtualize APIC accesses” and “virtual-interrupt delivery” are secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if these controls were 0. See Section 27.6.2.

29.7.10 User-Interrupt Recognition After VM Entry

A VM entry results in recognition of a pending user interrupt if it completes with $CR4.UINTR = IA32\_EFER.LMA = 1$ and with $UIRR \neq 0$; otherwise, no pending user interrupt is recognized.

29.8 VM-ENTRY FAILURES DURING OR AFTER LOADING GUEST STATE

VM-entry failures due to the checks identified in Section 29.3.1 and failures during the MSR loading identified in Section 29.4 are treated differently from those that occur earlier in VM entry. In these cases, the following steps take place:

1. Information about the VM-entry failure is recorded in the VM-exit information fields:
 - Exit reason.
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM-entry failure. The following numbers are used:
 33. VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 29.3.1.
 34. VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs (see Section 29.4).
 41. VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 29.9).
 - Bit 31 is set to 1 to indicate a VM-entry failure.
 - The remainder of the field (bits 30:16) is cleared.
 - Exit qualification. This field is set based on the exit reason.
 - VM-entry failure due to invalid guest state. In most cases, the exit qualification is cleared to 0. The following non-zero values are used in the cases indicated:
 1. Not used.
 2. Failure was due to a problem loading the PDPTes (see Section 29.3.1.6).
 3. Failure was due to an attempt to inject a non-maskable interrupt (NMI) into a guest that is blocking events through the STI blocking bit in the interruptibility-state field.
 4. Failure was due to an invalid VMCS link pointer (see Section 29.3.1.5).

VM-entry checks on guest-state fields may be performed in any order. Thus, an indication by exit qualification of one cause does not imply that there are not also other errors. Different processors may give different exit qualifications for the same VMCS.
 - VM-entry failure due to MSR loading. The exit qualification is loaded to indicate which entry in the VM-entry MSR-load area caused the problem (1 for the first entry, 2 for the second, etc.).
 - All other VM-exit information fields are unmodified.
2. Processor state is loaded as would be done on a VM exit (see Section 30.5). If this results in $[CR4.PAE \& CR0.PG \& \sim IA32\_EFER.LMA] = 1$, page-directory-pointer-table entries (PDPTes) may be checked and loaded (see Section 30.5.4).
3. The state of blocking by NMI is what it was before VM entry.
4. MSRs are loaded as specified in the VM-exit MSR-load area (see Section 30.6).

Although this process resembles that of a VM exit, many steps taken during a VM exit do not occur for these VM-entry failures:

- Most VM-exit information fields are not updated (see step 1 above).
- The valid bit in the injected-event identification field is not cleared.
- The guest-state area is not modified.
- No MSRs are saved into the VM-exit MSR-store area.

29.9 MACHINE-CHECK EVENTS DURING VM ENTRY

If a machine-check event occurs during a VM entry, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM entry:
 - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If CR4.MCE = 1, a machine-check exception (#MC) is delivered normally.
- The machine-check event is handled after VM entry completes:
 - If the VM entry ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM entry ends with CR4.MCE = 1, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered normally after VM entry.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- A VM-entry failure occurs as described in Section 29.8. The basic exit reason is 41, for “VM-entry failure due to machine-check event.”

The first option is not used if the machine-check event occurs after any guest state has been loaded. The second option is used only if VM entry is able to load all guest state.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

14. Updates to Chapter 30, Volume 3C

Change bars and violet text show changes to Chapter 30 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Section 30.2.1, “Basic VM-Exit Information,” and Section 30.3.4, “Saving Non-Register State,” were updated to include PEBS.

VM exits occur in response to certain instructions and events in VMX non-root operation as detailed in Section 28.1 through Section 28.2. VM exits perform the following operations:

1. Information about the cause of the VM exit is recorded in the VM-exit information fields and VM-entry control fields are modified as described in Section 30.2.
2. Processor state is saved in the guest-state area (Section 30.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 30.4). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.
4. The following may be performed in parallel and in any order (Section 30.5):
 - Processor state is loaded based in part on the host-state area and some VM-exit controls. This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM. See Section 34.15.6 for information on how processor state is loaded by such VM exits.
 - Address-range monitoring is cleared.
5. MSRs may be loaded from the VM-exit MSR-load area (Section 30.6). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.

VM exits are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

Section 30.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 30.2 through Section 30.6.

Section 34.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, ordinary transitions to SMM are replaced by VM exits to a separate SMM monitor. Called **SMM VM exits**, these are caused by the arrival of an SMI or the execution of VMCALL in VMX root operation. SMM VM exits differ from other VM exits in ways that are detailed in Section 34.15.2.

30.1 ARCHITECTURAL STATE BEFORE A VM EXIT

This section describes the architectural state that exists before a VM exit, especially for VM exits caused by events that would normally be delivered to guest software. Note the following:

- An exception causes a VM exit **directly** if the bit corresponding to that exception is set in the exception bitmap. A non-maskable interrupt (NMI) causes a VM exit directly if the “NMI exiting” VM-execution control is 1. An external interrupt causes a VM exit directly if the “external-interrupt exiting” VM-execution control is 1. A start-up IPI (SIPI) that arrives while a logical processor is in the wait-for-SIPI activity state causes a VM exit directly. INIT signals that arrive while the processor is not in the wait-for-SIPI activity state cause VM exits directly.
- An exception, NMI, external interrupt, or software interrupt causes a VM exit **indirectly** if it does not do so directly but delivery of the event causes a nested exception, double fault, task switch, APIC access (see Section 32.4), EPT violation, EPT misconfiguration, page-modification log-full event (see Section 31.3.6), or SPP-related event (see Section 31.3.4) that causes a VM exit.
- An event **results** in a VM exit if it causes a VM exit (directly or indirectly).

The following bullets detail when architectural state is and is not updated in response to VM exits:

- If an event causes a VM exit directly, it does not update architectural state as it would have if it had it not caused the VM exit:
 - A debug exception does not update DR6, DR7, or IA32_DEBUGCTL. (Information about the nature of the debug exception is saved in the exit qualification field.)
 - A page fault does not update CR2. (The linear address causing the page fault is saved in the exit-qualification field.)

- An NMI causes subsequent NMIs to be blocked, but only after the VM exit completes.
- An external interrupt does not acknowledge the interrupt controller and the interrupt remains pending, unless the “acknowledge interrupt on exit” VM-exit control is 1. In such a case, the interrupt controller is acknowledged and the interrupt is no longer pending.
- The flags L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
- If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT¹; or the TR register.
- No last-exception record is made if the event that would do so directly causes a VM exit.
- If a machine-check exception causes a VM exit directly, this does not prevent machine-check MSRs from being updated. These are updated by the machine-check event itself and not the resulting machine-check exception.
- If the logical processor is in an inactive state (see Section 27.4.2) 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 28.5.2 and Section 29.7.8) are not blocked in the HLT activity state. If an MTF VM exit occurs in the HLT activity state, the logical processor returns to the active state only after the VM exit completes. MTF VM exits are blocked the shutdown state and the wait-for-SIPI state.

- If an event causes a VM exit indirectly, the event does update architectural state:
 - A debug exception updates DR6, DR7, and the IA32_DEBUGCTL MSR. No debug exceptions are considered pending.
 - A page fault updates CR2.
 - An NMI causes subsequent NMIs to be blocked before the VM exit commences.
 - An external interrupt acknowledges the interrupt controller and the interrupt is no longer pending.
 - If the logical processor had been in an inactive state, it enters the active state and, before the VM exit commences, generates any special bus cycle that is normally generated when the active state is entered from that activity state.
 - There is no blocking by STI or by MOV SS when the VM exit commences.
 - Processor state that is normally updated as part of delivery through the IDT (CS, RIP, SS, RSP, RFLAGS) is not modified. However, the incomplete delivery of the event may write to the stack.
 - The treatment of last-exception records is implementation dependent:
 - Some processors make a last-exception record when beginning the delivery of an event (before it can encounter a nested exception). Such processors perform this update even if the event encounters a nested exception that causes a VM exit (including the case where nested exceptions lead to a triple fault).
 - Other processors delay making a last-exception record until event delivery has reached some event handler successfully (perhaps after one or more nested exceptions). Such processors do not update the last-exception record if a VM exit or triple fault occurs before an event handler is reached.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

2. If a VM exit takes the processor from an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

3. An exception is made if the logical processor had been inactive due to execution of MWAIT; in this case, it is considered to have become active before the VM exit.

- If the “virtual NMIs” VM-execution control is 1, VM entry injects an NMI, and delivery of the NMI causes a nested exception, double fault, task switch, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event, or APIC access that causes a VM exit, virtual-NMI blocking is in effect before the VM exit commences.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered during execution of IRET and the “NMI exiting” VM-execution control is 0, any blocking by NMI is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the exiting-event identification field; see Section 30.2.3.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered during execution of IRET and the “virtual NMIs” VM-execution control is 1, virtual-NMI blocking is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the exiting-event identification field; see Section 30.2.3.
- Suppose that a VM exit is caused directly by an x87 FPU Floating-Point Error (#MF) or by any of the following events if the event was unblocked due to (and given priority over) an x87 FPU Floating-Point Error: an INIT signal, an external interrupt, an NMI, an SMI; or a machine-check exception. In these cases, there is no blocking by STI or by MOV SS when the VM exit commences.
- Normally, a last-branch record may be made when an event is delivered. However, if such an event results in a VM exit before delivery is complete, no last-branch record is made.
- If machine-check exception results in a VM exit, processor state is suspect and may result in suspect state being saved to the guest-state area. A VM monitor should consult the RIPV and EIPV bits in the IA32_MC_G_STATUS MSR before resuming a guest that caused a VM exit resulting from a machine-check exception.
- If a VM exit results from a fault, APIC access (see Section 32.4), EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered while executing an instruction, data breakpoints due to that instruction may have been recognized and information about them may be saved in the pending debug exceptions field (unless the VM exit clears that field; see Section 30.3.4).
- The following VM exits are considered to happen after an instruction is executed:
 - VM exits resulting from debug traps (single-step, I/O breakpoints, and data breakpoints).
 - VM exits resulting from debug exceptions (data breakpoints) whose recognition was delayed by blocking by MOV SS.
 - VM exits resulting from some machine-check exceptions.
 - Trap-like VM exits due to execution of MOV to CR8 when the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1 (see Section 32.3). (Such VM exits can occur only from 64-bit mode and thus only on processors that support Intel 64 architecture.)
 - Trap-like VM exits due to execution of WRMSR when the “use MSR bitmaps” VM-execution control is 1; the value of ECX is in the range 800H–8FFH; and the bit corresponding to the ECX value in write bitmap for low MSRs is 0; and the “virtualize x2APIC mode” VM-execution control is 1. See Section 32.5.
 - VM exits caused by APIC-write emulation (see Section 32.4.3.2) that result from APIC accesses as part of instruction execution.

For these VM exits, the instruction’s modifications to architectural state complete before the VM exit occurs. Such modifications include those to the logical processor’s interruptibility state (see Table 27-3). If there had been blocking by MOV SS, POP SS, or STI before the instruction executed, such blocking is no longer in effect.

A VM exit that occurs in enclave mode sets bit 27 of the exit-reason field and bit 4 of the guest interruptibility-state field. Before such a VM exit is delivered, an Asynchronous Enclave Exit (AEX) occurs (see Chapter 40, “Enclave Exiting Events”). An AEX modifies architectural state (Section 40.3). In particular, the processor establishes the following architectural state as indicated:

- The following bits in RFLAGS are cleared: CF, PF, AF, ZF, SF, OF, and RF.
- FS and GS are restored to the values they had prior to the most recent enclave entry.
- RIP is loaded with the AEP of interrupted enclave thread.
- RSP is loaded from the URSP field in the enclave’s state-save area (SSA).

30.2 RECORDING VM-EXIT INFORMATION AND UPDATING VM-ENTRY CONTROL FIELDS

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. Section 30.2.1 to Section 30.2.5 detail the use of these fields.

In addition to updating the VM-exit information fields, the valid bit (bit 31) is cleared in the injected-event identification field. If bit 5 of the IA32_VMX_MISC MSR (index 485H) is read as 1 (see Appendix A.6), the value of IA32_EFER.LMA is stored into the “IA-32e mode guest” VM-entry control.¹

30.2.1 Basic VM-Exit Information

Section 27.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix C lists the numbers used and their meaning.

Future processors may introduce new VM exits with new basic exit reasons. Any such VM exit will occur only with VM-execution control settings introduced by the same processor or due to execution of an instruction that would cause an invalid-opcode exception (#UD) on previous processors. Software written for a particular processor generation should be aware that, if run on a later processor generation, it may experience VM exits with new basic exit reasons; it should also be aware that such a VM exit would result only from execution of an instruction that would have caused #UD on the earlier processor generation.
 - Bit 25 is set if the “prematurely busy shadow stack” VM-exit control is 1 and the VM exit caused a shadow stack become prematurely busy (see Section 28.4.3). Otherwise, the bit is cleared.
 - Bit 26 of this field is set to 1 if the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1. Such VM exits include those that occur due to the 1-setting of that control as well as others that might occur during execution of an instruction that asserted a bus lock.
 - Bit 27 of this field is set to 1 if the VM exit occurred while the logical processor was in enclave mode.

Such VM exits include those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field indicates an enclave interruption (bit 4 of the field was 1); or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.
 - The remainder of the field (bits 31:28 and bits 24:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 34.15.2.3).²
- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPs); system-management interrupts (SMIs) that arrive immediately after the execution of I/O instructions; task switches; INVEPT; INVLPG; INVPCID; INVVPID; LGDT; LIDT; LLDT; LTR; RDMSRLIST; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; WBINVD; WBNOINVD; WRMSR; WRMSRLIST; WRMSRNS; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 32.4); EPT violations (see Section 31.3.3.2); EOI virtualization (see Section 32.1.4); APIC-write emulation (see Section 32.4.3.3); page-modification log full (see Section 31.3.6); SPP-related events (see Section 31.3.4); and instruction timeout (see Section 28.2). For all other VM exits, this field is cleared. The following items provide details:

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.

2. Bit 31 of this field is set on certain VM-entry failures; see Section 29.8.

- For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 30-1.

Table 30-1. Exit Qualification for Debug Exceptions

Bit Position(s)	Contents
3:0	B3 – B0. When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if its corresponding enabling bit in DR7 is not set.
10:4	Not currently defined.
11	BLD. When set, this bit indicates that a bus lock was asserted while OS bus-lock detection was enabled and CPL > 0 (see Section 20.3.1.6 (“OS Bus-Lock Detection”)). ¹
12	Not currently defined.
13	BD. When set, this bit indicates that the cause of the debug exception is “debug register access detected.”
14	BS. When set, this bit indicates that the cause of the debug exception is either the execution of a single instruction (if RFLAGS.TF = 1 and IA32_DEBUGCTL.BTF = 0) or a taken branch (if RFLAGS.TF = DEBUGCTL.BTF = 1).
15	Not currently defined.
16	RTM. When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). ²
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 11 to indicate detection of a bus lock, while this field **sets** the bit to indicate that condition.
2. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- For a page-fault exception, the exit qualification contains the linear address that caused the page fault. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata. On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the page-fault exception occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

- For a start-up IPI (SIPI), the exit qualification contains the SIPI vector information in bits 7:0. Bits 63:8 of the exit qualification are cleared to 0.
- For a task switch, the exit qualification contains details about the task switch, encoded as shown in Table 30-2.
- For INVLPG, the exit qualification contains the linear-address operand of the instruction.
 - On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
 - If the INVLPG source operand specifies an unusable segment, the linear address specified in the exit qualification will match the linear address that the INVLPG would have used if no VM exit occurred. This address is not architecturally defined and may be implementation-specific.
- For INVEPT, INVPCID, INVVPID, LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, and XSAVES, the exit qualification receives the value of the instruction’s displacement field, which is sign-extended to 64 bits if necessary (32 bits on processors

Table 30-2. Exit Qualification for Task Switches

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Not currently defined
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

that do not support Intel 64 architecture). If the instruction has no displacement (for example, has a register operand), zero is stored into the exit qualification.

On processors that support Intel 64 architecture, an exception is made for RIP-relative addressing (used only in 64-bit mode). Such addressing causes an instruction to use an address that is the sum of the displacement field and the value of RIP that references the following instruction. In this case, the exit qualification is loaded with the sum of the displacement field and the appropriate RIP value.

In all cases, bits of this field beyond the instruction's address size are undefined. For example, suppose that the address-size field in the VM-exit instruction-information field (see Section 27.9.4 and Section 30.2.5) reports an n -bit address size. Then bits 63: n (bits 31: n on processors that do not support Intel 64 architecture) of the instruction displacement are undefined.

- For a control-register access, the exit qualification contains information about the access and has the format given in Table 30-3.
- For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 30-4.
- For an I/O instruction, the exit qualification contains information about the instruction and has the format given in Table 30-5.
- For MWAIT, the exit qualification contains a value that indicates whether address-range monitoring hardware was armed. The exit qualification is set either to 0 (if address-range monitoring hardware is not armed) or to 1 (if address-range monitoring hardware is armed).
- For RDMSRLIST and WRMSRLIST, the exit qualification depends on the setting of the “use MSR bitmaps” VM-execution control. If the control is 0, the exit qualification is zero. If the control is 1, the exit qualification is the index of the MSR whose access caused the VM exit (see Section 28.1.3).
- WBINVD and WBNOINVD use the same basic exit reason (see Appendix C). For WBINVD, the exit qualification is 0, while for WBNOINVD it is 1.
- WRMSR and WRMSRNS use the same basic exit reason (see Appendix C). For WRMSR, the exit qualification is 0, while for WRMSRNS it is 1.
- For an APIC-access VM exit resulting from a linear access or a guest-physical access to the APIC-access page (see Section 32.4), the exit qualification contains information about the access and has the format given in Table 30-6.¹

If the access to the APIC-access page occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

An APIC-access VM exit that sets the access type (bits 15:12) of the exit qualification to 0 (data read during instruction execution) or 1 (data write during instruction execution) chooses between those two values in a manner similar to the setting of bit 1 (W/R) of the page-fault error code. This implies the following:

1. The exit qualification is undefined if the access was part of the logging of a branch record or a legacy 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.

- For an APIC-access VM exit caused by the CLFLUSH and CLFLUSHOPT instructions, the access type is “data read during instruction execution.”
- For an APIC-access VM exit caused by the ENTER instruction, the access type is “data write during instruction execution.”

Table 30-3. Exit Qualification for Control-Register Accesses

Bit Positions	Contents
3:0	Number of control register (0 for CLTS and LMSW). Bit 3 is always 0 on processors that do not support Intel 64 architecture as they do not support CR8.
5:4	Access type: 0 = MOV to CR 1 = MOV from CR 2 = CLTS 3 = LMSW
6	LMSW operand type: 0 = register 1 = memory For CLTS and MOV CR, cleared to 0
7	Not currently defined
11:8	For MOV CR, the general-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) For CLTS and LMSW, cleared to 0
15:12	Not currently defined
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

- For an APIC-access VM exit caused by the MASKMOVQ instruction or the MASKMOVDQU instruction, the access type is “data write during instruction execution.”
- For an APIC-access VM exit caused by the MONITOR instruction, the access type is “data read during instruction execution.”

The following items specify how the access type is determined for certain accesses resulting from instruction execution:

- For an APIC-access VM exit caused directly by an access to a linear address in the DS save area (BTS or legacy PEBS), the access type is “linear access [asynchronous to instruction execution](#).”
- 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 [asynchronous to instruction execution](#).”

- 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 asynchronous to instruction execution.”
- For an APIC-access VM exit caused by architectural PEBS when the “PEBS uses guest physical addresses” VM-execution control is 1, the access type is “guest-physical access asynchronous to instruction execution.”

Such a VM exit stores 1 for bit 31 in the original-event identification field (see Section 30.2.4) if and only if it sets bits 15:12 of the exit qualification to 0011b (linear access during event delivery) or 1010b (guest-physical access during event delivery).

See Section 32.4.4 for further discussion of these instructions and APIC-access VM exits.

For APIC-access VM exits resulting from physical accesses to the APIC-access page (see Section 32.4.6), the exit qualification is undefined.

- For an EPT violation, the exit qualification contains information about the access causing the EPT violation and has the format given in Table 30-7.

As noted in that table, the format and meaning of the exit qualification depends on the setting of the “mode-based execute control for EPT” VM-execution control and whether the processor supports advanced VM-exit information for EPT violations.¹

An EPT violation that occurs during as a result of execution of a read-modify-write operation sets bit 1 (data write). Whether it also sets bit 0 (data read) is implementation-specific and, for a given implementation, may differ for different kinds of read-modify-write operations.

Table 30-4. Exit Qualification for MOV DR

Bit Position(s)	Contents
2:0	Number of debug register
3	Not currently defined
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Not currently defined
11:8	General-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8 - 15 = R8 - R15, respectively
63:12	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

1. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

Table 30-5. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Not currently defined
31:16	Port number (as specified in DX or in an immediate operand)
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

Table 30-6. Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	- If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page. - Undefined if the APIC-access VM exit is due a guest-physical access
15:12	Access type: 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 4 = linear access <i>asynchronous to instruction execution</i> 10 = guest-physical access during event delivery 11 = guest-physical access <i>asynchronous to instruction execution</i> 15 = guest-physical access for an instruction fetch or during instruction execution Other values not used
16	This bit is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These includes guest-physical accesses related to trace output by Intel PT (see Section 28.5.4), accesses related to <i>legacy</i> PEBS on processors with the “EPT-friendly” enhancement (see Section 22.9.5), <i>accesses related to architectural PEBS (see Section 28.5.5)</i> , and accesses that occur during user-interrupt delivery (see Section 9.4.2).
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

Bit 12 reports “NMI unblocking due to IRET”; see Section 30.2.3.

Bit 16 is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These include trace-address pre-translation (TAPT) for Intel PT (see Section 28.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 22.9.5), *accesses related to architectural PEBS (see Section 28.5.5)*, and accesses as part of user-interrupt delivery (see Section 9.4.2).

- For VM exits caused as part of EOI virtualization (Section 32.1.4), bits 7:0 of the exit qualification are set to vector of the virtual interrupt that was dismissed by the EOI virtualization. Bits above bit 7 are cleared.
- For APIC-write VM exits (Section 32.4.3.3), bits 11:0 of the exit qualification are set to the page offset of the write access that caused the VM exit.¹ Bits above bit 11 are cleared.
- For a VM exit due to a page-modification log-full event (Section 31.3.6), bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 30.2.3). Bit 16 is set if the VM exit occurs during TAPT, EPT-friendly PEBS, *due to a write for architectural PEBS*, or *during* user-interrupt delivery. All other bits of the exit qualification are undefined.
- For a VM exit due to an SPP-related event (Section 31.3.4), bit 11 of the exit qualification indicates the type of event: 0 indicates an SPP misconfiguration and 1 indicates an SPP miss. Bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 30.2.3). Bit 16 is set if the VM exit occurs during TAPT, EPT-friendly PEBS, *due to a write for architectural PEBS*, or *during* user-interrupt delivery. All other bits of the exit qualification are undefined.
- If the “PASID translation” VM-execution control, PASID translation is performed for executions of the ENQCMD and ENQCMLS instructions (see Section 28.5.9). PASID translation may fail, resulting in a VM exit. Such a VM exit saves an exit qualification specified in the following items:
 - For ENQCMD, the exit qualification is IA32_PASID[19:0].
 - For ENQCMLS, the exit qualification contains the low 32 bits of the instruction’s source operand (which had been read from memory prior to PASID translation).
- For a VM exit due to an instruction timeout (Section 28.2), bit 0 indicates (if set) that the context of the virtual machine is invalid and that the VM should not be resumed. Bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 30.2.3). All other bits of the exit qualification are undefined.
- **Guest linear address.** For some VM exits, this field receives a linear address that pertains to the VM exit. The field is set for different VM exits as follows:
 - VM exits due to attempts to execute LMSW with a memory operand. In these cases, this field receives the linear address of that operand. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 - VM exits due to attempts to execute INS or OUTS for which the relevant segment is usable (if the relevant segment is not usable, the value is undefined). (ES is always the relevant segment for INS; for OUTS, the relevant segment is DS unless overridden by an instruction prefix.) The linear address is the base address of relevant segment plus (E)DI (for INS) or (E)SI (for OUTS). Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded is the original address before any masking (and may thus contain any metadata).
 - VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 30-7; these are all EPT violations except those resulting from an attempt to load the PDPTes as of execution of the MOV CR instruction and those due to TAPT). The linear address may translate to the guest-physical address whose access caused the EPT violation. Alternatively, translation of the linear address may reference a paging-structure entry whose access caused the EPT violation. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 If the EPT violation occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.
 - VM exits due to SPP-related events. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 - If the “prematurely busy shadow stack” VM-exit control is 1, certain VM exits (besides those noted above) save the linear address that pertains to the VM exit if the VM exit caused a shadow stack to become prematurely busy (see Section 28.4.3). This is true for VM exits due for these reasons: EPT misconfiguration, page-modification log-full event, and instruction timeout. (A VM exit due to instruction timeout that

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.

sets bit 0 of the exit qualification, indicating that VM context is invalid, does not save a valid linear address.) If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.

— For all other VM exits, the field is undefined.

- **Guest-physical address.** For a VM exit due to an EPT violation, an EPT misconfiguration, or an SPP-related event, this field receives the guest-physical address that caused the EPT violation or EPT misconfiguration. For all other VM exits, the field is undefined.

If the EPT violation or EPT misconfiguration occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

Table 30-7. Exit Qualification for EPT Violations

Bit Position(s)	Contents
0	Set if the access causing the EPT violation was a data read. ¹
1	Set if the access causing the EPT violation was a data write. ¹
2	Set if the access causing the EPT violation was an instruction fetch.
3	The logical-AND of bit 0 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was readable). ²
4	The logical-AND of bit 1 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was writeable). ²
5	The logical-AND of bit 2 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. ² If the “mode-based execute control for EPT” VM-execution control is 0, this indicates whether the guest-physical address was executable. If that control is 1, this indicates whether the guest-physical address was executable for supervisor-mode linear addresses.
6	If the “mode-based execute control” VM-execution control is 0, the value of this bit is undefined. If that control is 1, this bit is the logical-AND of bit 10 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. In this case, it indicates whether the guest-physical address was executable for user-mode linear addresses. ³
7	Set if the guest linear-address field is valid. The guest linear-address field is valid for all EPT violations except those resulting from an attempt to load the guest PDPTes as part of the execution of the MOV CR instruction, those due to trace-address pre-translation (TAPT; Section 28.5.4), and those due to architectural PEBS (Section 28.5.5).
8	If bit 7 is 1: ▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address. ▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit. Reserved if bit 7 is 0 (cleared to 0).
9	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if the linear address is a supervisor-mode linear address and 1 if it is a user-mode linear address. (If CR0.PG = 0, the translation of every linear address is a user-mode linear address and thus this bit will be 1.) Otherwise, this bit is undefined.
10	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if paging translates the linear address to a read-only page and 1 if it translates to a read/write page. (If CR0.PG = 0, every linear address is read/write and thus this bit will be 1.) Otherwise, this bit is undefined.

Table 30-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
11	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if paging translates the linear address to an executable page and 1 if it translates to an execute-disable page. (If CR0.PG = 0, CR4.PAE = 0, or IA32_EFER.NXE = 0, every linear address is executable and thus this bit will be 0.) Otherwise, this bit is undefined.
12	NMI unblocking due to IRET (see Section 30.2.3).
13	Set if the access causing the EPT violation was a shadow-stack access.
14	If supervisor shadow-stack control is enabled (by setting bit 7 of EPTP), this bit is the same as bit 60 in the EPT paging-structure entry that maps the page of the guest-physical address of the access causing the EPT violation. Otherwise (or if translation of the guest-physical address terminates before reaching an EPT paging-structure entry that maps a page), this bit is undefined.
15	This bit is set if the EPT violation was caused as a result of guest-paging verification. See Section 31.3.3.2.
16	This bit is set if the access was asynchronous to instruction execution not the result of event delivery. The bit is set if the access is related to trace output by Intel PT (see Section 28.5.4), accesses related to legacy PEBS on processors with the “EPT-friendly” enhancement (see Section 22.9.5), accesses related to architectural PEBS (see Section 28.5.5), or to user-interrupt delivery (see Section 9.4.2). Otherwise, this bit is cleared.
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. If accessed and dirty flags for EPT are enabled, processor accesses to guest paging-structure entries are treated as writes with regard to EPT violations (see Section 31.3.3.2). If such an access causes an EPT violation, the processor sets both bit 0 and bit 1 of the exit qualification.
2. Bits 5:3 are cleared to 0 if either (1) any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present; or (2) 4-level EPT is in use and the guest-physical address sets any bits in the range 51:48 (see Section 31.3.2).
3. Bit 6 is cleared to 0 if (1) the “mode-based execute control” VM-execution control is 1; and (2) either (a) any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present; or (b) 4-level EPT is in use and the guest-physical address sets any bits in the range 51:48 (see Section 31.3.2).
4. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

30.2.2 Information for VM Exits Due to Vectored Events

Section 27.9.2 defines fields containing information for VM exits due to the following events: exceptions (including those generated by the instructions INT1, INT3, INTO, BOUND, UD1, UD2, and UDB); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs).¹ Such VM exits include those that occur on an attempt at a task switch that causes an exception before generating the VM exit due to the task switch that causes the VM exit.

The following items detail the use of these fields:

- **Exiting-event identification** (format given in Table 27-20). The following items detail how this field is established for VM exits due to these events:
 - For an exception, bits 7:0 receive the exception vector (at most 31). For an NMI, bits 7:0 are set to 2. For an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 5 (privileged software exception), or 6 (software exception). Hardware exceptions comprise all exceptions except the following:

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*.

- Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)
- Breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.)

BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD0, UD1, UD2, and UDB are hardware exceptions.

- Bit 11 is set to 1 if the VM exit is caused by a hardware exception that would have delivered an error code on the stack; otherwise, it is 0.¹ This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).² If bit 11 is set to 1, the error code is placed in the exiting-event error code (see below).
- Bit 12 reports “NMI unblocking due to IRET”; see Section 30.2.3. The value of this bit is undefined if the VM exit is due to a double fault (the event type is hardware exception and the vector is 8).
- Bit 13 (nested exception) is set to 1 if the VM exit was caused by a hardware exception that was encountered during FRED event delivery of some other event; it is not set for VM exits due to #DF or to those encountered during IDT event delivery. Other VM exits for which the field is valid (including VM exits due to #DF) save bit 13 as 0.
- Bits 30:14 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- **Exiting-event error code.**

- For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the exiting-event identification field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the original-event identification field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).
- For other VM exits, the value of this field is undefined.

30.2.3 Information About NMI Unblocking Due to IRET

A VM exit may occur during execution of the IRET instruction for reasons including the following: faults, EPT violations, page-modification log-full events, SPP-related events, or instruction timeouts.

An execution of IRET that commences while non-maskable interrupts (NMIs) are blocked will unblock NMIs even if a fault or VM exit occurs; the state saved by such a VM exit will indicate that NMIs were not blocked.

VM exits for the reasons enumerated above provide more information to software by saving a bit called “NMI unblocking due to IRET.” This bit is defined if (1) either the “NMI exiting” VM-execution control is 0 or the “virtual NMIs” VM-execution control is 1; (2) the VM exit does not set the valid bit in the original-event identification field (see Section 30.2.4); and (3) the VM exit is not due to a double fault. In these cases, the bit is defined as follows:

- The bit is 1 if the VM exit resulted from a memory access as part of execution of the IRET instruction and one of the following holds:

1. FRED event delivery always pushes an error code on the stack (it pushes zero if no error code is defined). Nevertheless, bit 11 here is set only for hardware exceptions that would have pushed an error code with IDT event delivery.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- The “virtual NMIs” VM-execution control is 0 and blocking by NMI (see Table 27-3) was in effect before execution of IRET.
- The “virtual NMIs” VM-execution control is 1 and virtual-NMI blocking was in effect before execution of IRET.
- The bit is 0 for all other relevant VM exits.

For VM exits due to faults, NMI unblocking due to IRET is saved in bit 12 of the exiting-event identification field (Section 30.2.2). For VM exits due to EPT violations, page-modification log-full events, SPP-related events, and instruction timeouts, NMI unblocking due to IRET is saved in bit 12 of the exit qualification (Section 30.2.1).

(Executions of IRET may also incur VM exits due to APIC accesses and EPT misconfigurations. These VM exits do not report information about NMI unblocking due to IRET.)

30.2.4 Information for VM Exits During Event Delivery

Section 27.9.3 defined **original-event fields** containing information for VM exits that occur while delivering an event and as a result of any of the following cases:¹

- An exception occurs during event delivery and causes a VM exit (because the bit associated with the exception is set to 1 in the exception bitmap).
- A task switch is invoked through a task gate in the IDT. The VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 28.4.2).
- Event delivery causes an APIC-access VM exit (see Section 32.4).
- An EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that occurs during event delivery.
- Any of the above VM exits that occur during user-interrupt notification processing (see Section 9.5.2). Such VM exits will be treated as if they occurred during delivery of an external interrupt with the vector UINV.

NOTE

When FRED transitions are enabled, SYSCALL and SYSENTER are delivered using FRED event delivery. Execution of these instructions may result in the VM exits enumerated above (except task switches and user-interrupt notification processing). For that reason, the original-event fields may record information about SYSCALL and SYSENTER when FRED transitions are enabled.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 29.6.1.2).

A VM exit is not considered to occur during event delivery in any of the following circumstances:

- The original event causes the VM exit directly (for example, because the original event is a non-maskable interrupt (NMI) and the “NMI exiting” VM-execution control is 1).
- The original event results in a double-fault exception that causes the VM exit directly.
- The VM exit occurred as a result of fetching the first instruction of the handler invoked by the event delivery.
- The VM exit is caused by a triple fault.
- The original event was a software interrupt (INT n) executed in virtual-8086 mode with EFLAGS.IOPL < 3 and the VM exit was due to a general-protection exception (#GP) that occurred because either CR4.VME = 0 or bit n of the software interrupt redirection bit map in the TSS is set.

The following items detail the use of these fields:

- Original-event identification (format given in Table 27-21). The following items detail how this field is established for VM exits that occur during event delivery:

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).

- 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:¹

- Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)
- Breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.)

BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD0, UD1, UD2, and UDB are hardware exceptions.

- Bit 11 is set to 1 if the VM exit occurred during delivery of a hardware exception that would have delivered an error code on the stack; otherwise, it is 0.² 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 original-event error code (see below).
- Bit 12 is undefined.
- Bit 13 (nested exception) is set to 1 if the VM exit occurred during delivery of a hardware exception that was encountered during FRED event delivery of some other event; it is not set for VM exits due to #DF or to those encountered during IDT event delivery. Other VM exits for which the original-event identification field is valid (including a VM exits that occurred during delivery of #DF) save bit 13 as 0.
- Bits 30:14 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- Original-event error code.
 - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the original-event identification field, this field receives the error code that would have been pushed on the stack by the event that was being delivered at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
 - For other VM exits, the value of this field is undefined.
- Original-event data.
 - For VM exits that occur during FRED event delivery, this field receives the event data that would have been pushed on the stack by the event that was being delivered at the time of the VM exit. This is done for all events (if FRED event delivery would have saved zero for event data, that value is saved).
 - For other VM exits (including those that occur when FRED transitions are not enabled), the value of this field is undefined.

1. 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*.

2. FRED event delivery always pushes an error code on the stack (it pushes zero if no error code is defined). Nevertheless, bit 11 here is set only for hardware exceptions that would have pushed an error code with IDT event delivery.

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.

30.2.5 Information for VM Exits Due to Instruction Execution

Section 27.9.4 defined fields containing information for VM exits that occur due to instruction execution. (The VM-exit instruction length is also used for VM exits that occur during the delivery of a software interrupt or software exception.) The following items detail their use.

- **VM-exit instruction length.** This field is used in the following cases:
 - For fault-like VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 28.1.2) or based on the settings of VM-execution controls (see Section 28.1.3): CLTS, CPUID, ENCLS, GETSEC, HLT, IN, INS, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LOADIWKEY, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, PCONFIG, RDMSR, RDPMSR, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, SEAMCALL, SGDT, SIDT, SLDT, STR, TDCALL, SGDT, SIDT, SLDT, STR, TPAUSE, UMWAIT, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WBNOINVD, WRMSR, XRSTORS, XSETBV, and XSAVES.¹
 - For VM exits due to software exceptions (those generated by executions of INT3 or INTO) or privileged software exceptions (those generated by executions of INT1).
 - For VM exits due to faults encountered during delivery of a software interrupt, privileged software exception, software exception, or (if FRED transitions are enabled) SYSCALL or SYSENTER.
 - For VM exits due to attempts to effect a task switch via instruction execution. These are VM exits that produce an exit reason indicating task switch and either of the following:
 - An exit qualification indicating execution of CALL, IRET, or JMP instruction.
 - An exit qualification indicating a task gate in the IDT and an original-event identification field indicating that the task gate was encountered during delivery of a software interrupt, privileged software exception, or software exception.
 - For APIC-access VM exits and for VM exits caused by EPT violations, page-modification log-full events, and SPP-related events encountered during delivery of a software interrupt, privileged software exception, or software exception.²
 - For VM exits due executions of VMFUNC that fail because one of the following is true:
 - EAX indicates a VM function that is not enabled (the bit at position EAX is 0 in the VM-function controls; see Section 28.5.7.2).
 - EAX = 0 and either ECX ≥ 512 or the value of ECX selects an invalid tentative EPTP value (see Section 28.5.7.3).

In all the above cases, this field receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit (see the next paragraph for one exception).

The cases of VM exits encountered during delivery of a software interrupt, privileged software exception, or software exception include those encountered during delivery of events injected as part of VM entry (see Section 29.6.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

If the VM exit occurred in enclave mode, this field is cleared (none of the previous items apply).

Table 30-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS

Bit Position(s)	Content
6:0	Undefined.

1. This item applies only to fault-like VM exits. It does not apply to trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1 or to those following executions of the WRMSR instruction when the “virtualize x2APIC mode” VM-execution control is 1.
2. The VM-exit instruction-length field is not defined following APIC-access VM exits resulting from physical accesses (see Section 32.4.6) even if encountered during delivery of a software interrupt, privileged software exception, software exception, SYSCALL, or SYSENTER.

Table 30-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS (Contd.)

Bit Position(s)	Content
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, TPAUSE, UMWAIT, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, or XSAVES, this field receives information about the instruction that caused the VM exit. The format of the field depends on the identity of the instruction causing the VM exit:
 - For VM exits due to attempts to execute INS or OUTS, the field has the format is given in Table 30-8.¹
 - For VM exits due to attempts to execute INVEPT, INVPCID, or INVVPID, the field has the format is given in Table 30-9.
 - For VM exits due to attempts to execute LIDT, LGDT, SIDT, or SGDT, the field has the format is given in Table 30-10.
 - For VM exits due to attempts to execute LLDT, LTR, SLDT, or STR, the field has the format is given in Table 30-11.
 - For VM exits due to attempts to execute RDRAND or RDSEED, the field has the format is given in Table 30-12.
 - For VM exits due to attempts to execute TPAUSE or UMWAIT, the field has the format is given in Table 30-13.
 - For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, or XSAVES, the field has the format is given in Table 30-14.
 - For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 30-15.
 - For VM exits due to attempts to execute LOADIWKEY, the field has the format is given in Table 30-16.

For all other VM exits, the field is undefined, unless the VM exit occurred in enclave mode, in which case the field is cleared.
- I/O RCX, I/O RSI, I/O RDI, I/O RIP.** These fields are undefined except for SMM VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions. See Section 34.15.2.3. Note that, if the VM exit occurred in enclave mode, these fields are all cleared.

1. The format of the field was undefined for these VM exits on the first processors to support the virtual-machine extensions. Software can determine whether the format specified in Table 30-8 is used by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

- **MSR data.** An execution of WRMSRLIST may cause a VM exit if it would write to an MSR for which the MSR bitmaps do not allow writes (see Section 28.1.3). Such VM exits save the 64-bit data that would have been written to the MSR into this field in the VMCS.

Table 30-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for memory instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Reg2 (same encoding as IndexReg above)

Table 30-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
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 30-10. 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 30-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).

Table 30-11. 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 30-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND and RDSEED

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (destination register): 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture)
10:7	Undefined.
12:11	Operand size: 0: 16-bit 1: 32-bit 2: 64-bit The value 3 is not used.
31:13	Undefined.

Table 30-13. Format of the VM-Exit Instruction-Information Field as Used for TPAUSE and UMWAIT

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (source register): 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture)
31:7	Undefined.

Table 30-14. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Undefined.

30.3 SAVING GUEST STATE

VM exits save certain components of processor state into corresponding fields in the guest-state area of the VMCS (see Section 27.4). On processors that support Intel 64 architecture, the full value of each natural-width field (see Section 27.11.2) is saved regardless of the mode of the logical processor before and after the VM exit.

Table 30-15. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
31:28	Reg2 (same encoding as Reg1 above)

In general, the state saved is that which was in the logical processor at the time the VM exit commences. See Section 30.1 for a discussion of which architectural updates occur at that time.

Table 30-16. Format of the VM-Exit Instruction-Information Field as Used for `LOADIWKEY`

Bit Position(s)	Content
2:0	Undefined.
6:3	Reg1: identifies the first XMM register operand (XMM0-XMM15; values 8-15 are used only on processors that support Intel 64 architecture).
30:7	Undefined.
31:28	Reg2: identifies the second XMM register operand (see above).

Section 30.3.1 through Section 30.3.4 provide details for how various components of processor state are saved. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

30.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs are saved as follows:

- The contents of CR0, CR3, CR4, and the `IA32_SYSENTER_CS`, `IA32_SYSENTER_ESP`, and `IA32_SYSENTER_EIP` MSRs are saved into the corresponding fields. Bits 63:32 of the `IA32_SYSENTER_CS` MSR are not saved. On processors that do not support Intel 64 architecture, bits 63:32 of the `IA32_SYSENTER_ESP` and `IA32_SYSENTER_EIP` MSRs are not saved.
- If the “save debug controls” VM-exit control is 1, the contents of DR7 and the `IA32_DEBUGCTL` MSR are saved into the corresponding fields. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always saved data into these fields.
- If the “save `IA32_PAT`” VM-exit control is 1, the contents of the `IA32_PAT` MSR are saved into the corresponding field.
- If the “save `IA32_EFER`” VM-exit control is 1, the contents of the `IA32_EFER` MSR are saved into the corresponding field.
- If the processor supports either the 1-setting of the “load `IA32_BNDCFGS`” VM-entry control or that of the “clear `IA32_BNDCFGS`” VM-exit control, the contents of the `IA32_BNDCFGS` MSR are saved into the corresponding field.
- If the processor supports either the 1-setting of the “load `IA32_RTIT_CTL`” VM-entry control or that of the “clear `IA32_RTIT_CTL`” VM-exit control, the contents of the `IA32_RTIT_CTL` MSR are saved into the corresponding field.
- If the processor supports the 1-setting of the “load CET” VM-entry control, the contents of the `IA32_S_CET` and `IA32_INTERRUPT_SSP_TABLE_ADDR` MSRs are saved into the corresponding fields. On processors that do not support Intel 64 architecture, bits 63:32 of these MSRs are not saved.
- If the processor supports either the 1-setting of the “load guest `IA32_LBR_CTL`” VM-entry control or that of the “clear `IA32_LBR_CTL`” VM-exit control, the contents of the `IA32_LBR_CTL` MSR are saved into the corresponding field.
- If the processor supports the 1-setting of the “load PKRS” VM-entry control, the contents of the `IA32_PKRS` MSR are saved into the corresponding field.
- If a processor supports user interrupts, every VM exit saves `UINV` into the guest `UINV` field in the VMCS (bits 15:8 of the field are cleared).
- If the “save `IA32_PERF_GLOBAL_CTL`” VM-exit control is 1, the contents of the `IA32_PERF_GLOBAL_CTL` MSR are saved into the corresponding field.
- If the “save FRED” VM-exit control is 1, the contents of the following MSRs are saved into the corresponding fields: `IA32_FRED_CONFIG`, `IA32_FRED_RSP1`, `IA32_FRED_RSP2`, `IA32_FRED_RSP3`, `IA32_FRED_STKLVLS`, `IA32_FRED_SSP1`, `IA32_FRED_SSP2`, and `IA32_FRED_SSP3`.
- The value of the `SMBASE` field is undefined after all VM exits except SMM VM exits. See Section 34.15.2.

30.3.2 Saving Segment Registers and Descriptor-Table Registers

For each segment register (CS, SS, DS, ES, FS, GS, LDTR, or TR), the values saved for the base-address, segment-limit, and access rights are based on whether the register was unusable (see Section 27.4.1) before the VM exit:

- If the register was unusable, the values saved into the following fields are undefined: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in the access-rights field. The following exceptions apply:
 - CS.
 - The base-address and segment-limit fields are saved.
 - The L, D, and G bits are saved in the access-rights field.
 - SS.
 - DPL is saved in the access-rights field.
 - On processors that support Intel 64 architecture, bits 63:32 of the value saved for the base address are always zero.
 - DS and ES. On processors that support Intel 64 architecture, bits 63:32 of the values saved for the base addresses are always zero.
 - FS and GS. The base-address field is saved.
- If the register was not unusable, the values saved into the following fields are those which were in the register before the VM exit: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in access rights.
- Bits 31:17 and 11:8 in the access-rights field are always cleared. Bit 16 is set to 1 if and only if the segment is unusable.

The contents of the GDTR and IDTR registers are saved into the corresponding base-address and limit fields.

30.3.3 Saving RIP, RSP, RFLAGS, and SSP

The contents of the RIP, RSP, RFLAGS, and SSP (shadow-stack pointer) registers are saved as follows:

- The value saved in the RIP field is determined by the nature and cause of the VM exit:
 - If the VM exit occurred in enclave mode, the value saved is the AEP of interrupted enclave thread (the remaining items do not apply).
 - If the VM exit occurs due to by an attempt to execute an instruction that causes VM exits unconditionally or that has been configured to cause a VM exit via the VM-execution controls, the value saved references that instruction.
 - If the VM exit is caused by an occurrence of an INIT signal, a start-up IPI (SIPI), or system-management interrupt (SMI), the value saved is that which was in RIP before the event occurred.
 - If the VM exit occurs due to the 1-setting of either the “interrupt-window exiting” VM-execution control or the “NMI-window exiting” VM-execution control, the value saved is that which would be in the register had the VM exit not occurred.
 - If the VM exit is due to an external interrupt, non-maskable interrupt (NMI), or hardware exception (as defined in Section 30.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,¹ or into the old task-state segment had the event been delivered through a task gate).
 - If the VM exit is due to a triple fault, the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate, or into the old task-state segment had the event been delivered through a task gate) had delivery of the double fault not encountered the nested exception that caused the triple fault.

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 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.
- If the VM exit is due to an execution of MOV to CR8 or WRMSR that reduced the value of bits 7:4 of VTPR (see Section 32.1.1) below that of TPR threshold VM-execution control field (see Section 32.1.2), the value saved references the instruction following the MOV to CR8 or WRMSR.
- If the VM exit was caused by APIC-write emulation (see Section 32.4.3.2) that results from an APIC access as part of instruction execution, the value saved references the instruction following the one whose execution caused the APIC-write emulation.
- The contents of the RSP register are saved into the RSP field.
- With the exception of the resume flag (RF; bit 16), the contents of the RFLAGS register is saved into the RFLAGS field. RFLAGS.RF is saved as follows:
 - If the VM exit occurred in enclave mode, the value saved is 0 (the remaining items do not apply).
 - If the VM exit is caused directly by an event that would normally be delivered to software, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate¹ or into the old task-state segment had the event been delivered through a task gate) had the event been delivered to guest software. See below for VM exits due to task switches caused by task gates in the IDT.
 - If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
 - If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.
 - If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.²
 - For APIC-access VM exits and for VM exits caused by EPT violations, EPT misconfigurations, page-modification log-full events, or SPP-related events, the value saved depends on whether the VM exit occurred during delivery of an event:
 - If the VM exit stored 0 for bit 31 for original-event identification field (because the VM exit did not occur during delivery of an event; see Section 30.2.4), the value saved is 1.
 - If the VM exit stored 1 for bit 31 for original-event identification field (because the VM exit did occur during delivery of an event), the value saved is the value that would have appeared in the saved RFLAGS image had the event been delivered (see above).
 - For all other VM exits, the value saved is the value RFLAGS.RF had before the VM exit occurred.
- If the processor supports the 1-setting of the “load CET” VM-entry control, the contents of the SSP register are saved into the SSP field.

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.

30.3.4 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.¹ See Section 30.1 for details of how events leading to a VM exit may affect the activity state. If the VM exit occurred during user-interrupt notification processing (see Section 9.5.2) and the logical processor would have entered the HLT state following user-interrupt notification processing, the saved activity state is "HLT".
- The interruptibility-state field is saved to reflect the logical processor's interruptibility before the VM exit.
 - See Section 30.1 for details of how events leading to a VM exit may affect this state.
 - VM exits that end outside system-management mode (SMM) save bit 2 (blocking by SMI) as 0 regardless of the state of such blocking before the VM exit.
 - Bit 3 (blocking by NMI) is treated specially if the "virtual NMIs" VM-execution control is 1. In this case, the value saved for this field does not indicate the blocking of NMIs but rather the state of virtual-NMI blocking.
 - Bit 4 (enclave interruption) is set to 1 if the VM exit occurred while the logical processor was in enclave mode.

Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field sets this bit; or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.

- The pending debug exceptions field is saved as clear for all VM exits except the following:
 - A VM exit caused by an INIT signal, a machine-check exception, or a system-management interrupt (SMI).
 - A VM exit with basic exit reason "TPR below threshold",² "virtualized EOI", "APIC write", "monitor trap flag," or "bus-lock detected."
 - A VM exit due to trace-address pre-translation (TAPT; see Section 28.5.4), due to accesses related to [legacy PEBS](#) on processors with the "EPT-friendly" enhancement (see Section 22.9.5), or [due to architectural PEBS](#) (see Section 28.5.5). 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 [either form of PEBS](#), these VM exits (with the exception of those due to EPT misconfigurations) set bit 16 of the exit qualification, indicating that they are asynchronous to instruction execution and not part of event delivery.
 - VM exits that are not caused by debug exceptions and that occur while there is MOV-SS blocking of debug exceptions.

For VM exits that do not clear the field, the value saved is determined as follows:

- Each of bits 3:0 may be set if it corresponds to a matched breakpoint. This may be true even if the corresponding breakpoint is not enabled in DR7.
- Suppose that a VM exit is due to an INIT signal, a machine-check exception, or an SMI; or that a VM exit has basic exit reason "TPR below threshold" or "monitor trap flag." In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit.

If the VM exit occurs immediately after VM entry, the value saved may match that which was loaded on VM entry (see Section 29.7.3). Otherwise, the following items apply:

- Bit 12 (enabled breakpoint) is set to 1 in any of the following cases:
 - If there was at least one matched data or I/O breakpoint that was enabled in DR7.

1. If this activity state was an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

2. This item includes VM exits that occur as a result of certain VM entries (Section 29.7.7).

- If it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 29.7.3) and the VM exit occurred before those exceptions were either delivered or lost.
- If the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). (This does not apply to VM exits with basic exit reason “monitor trap flag.”)
- If a bus lock was asserted while $CPL > 0$ and OS bus-lock detection was enabled.

In other cases, bit 12 is cleared to 0.

- Bit 14 (BS) is set if $RFLAGS.TF = 1$ in either of the following cases:
 - $IA32\_DEBUGCTL.BTF = 0$ and the cause of a pending debug exception was the execution of a single instruction.
 - $IA32\_DEBUGCTL.BTF = 1$ and the cause of a pending debug exception was a taken branch.
- Bit 16 (RTM) is set if a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions had been enabled. (This does not apply to VM exits with basic exit reason “monitor trap flag.”)
- Suppose that a VM exit is due to another reason (but not a debug exception) and occurs while there is MOV-SS blocking of debug exceptions. In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit. If the VM exit occurs immediately after VM entry (no instructions were executed in VMX non-root operation), the value saved may match that which was loaded on VM entry (see Section 29.7.3). Otherwise, the following items apply:
 - Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 29.7.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
 - The setting of bit 14 (BS) is implementation-specific. However, it is not set if $RFLAGS.TF = 0$ or $IA32\_DEBUGCTL.BTF = 1$.
- The reserved bits in the field are cleared.
- If the “save VMX-preemption timer value” VM-exit control is 1, the value of timer is saved into the VMX-preemption timer-value field. This is the value loaded from this field on VM entry as subsequently decremented (see Section 28.5.1). VM exits due to timer expiration save the value 0. Other VM exits may also save the value 0 if the timer expired during VM exit. (If the “save VMX-preemption timer value” VM-exit control is 0, VM exit does not modify the value of the VMX-preemption timer-value field.)
- If the logical processor supports the 1-setting of the “enable EPT” VM-execution control, values are saved into the four (4) PDPTE fields as follows:
 - If the “enable EPT” VM-execution control is 1 and the logical processor was using PAE paging at the time of the VM exit, the PDPTE values currently in use are saved:¹
 - The values saved into bits 11:9 of each of the fields is undefined.
 - If the value saved into one of the fields has bit 0 (present) clear, the value saved into bits 63:1 of that field is undefined. That value need not correspond to the value that was loaded by VM entry or to any value that might have been loaded in VMX non-root operation.
 - If the value saved into one of the fields has bit 0 (present) set, the value saved into bits 63:12 of the field is a guest-physical address.
 - If the “enable EPT” VM-execution control is 0 or the logical processor was not using PAE paging at the time of the VM exit, the values saved are undefined.

1. A logical processor uses PAE paging if $CRO.PG = 1$, $CR4.PAE = 1$ and $IA32\_EFER.LMA = 0$. See Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM exit functions as if the “enable EPT” VM-execution control were 0. See Section 27.6.2.

30.4 SAVING MSRS

After processor state is saved to the guest-state area, values of MSRs may be stored into the VM-exit MSR-store area (see Section 27.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-store count) is processed in order by storing the value of the MSR indexed by bits 31:0 (as they would be read by RDMSR) into bits 127:64. Processing of an entry fails in either of the following cases:

- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMBASE is an MSR that can be read only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be saved on VM exits for model-specific reasons. A processor may prevent certain MSRs (based on the value of bits 31:0) from being stored on VM exits, even if they can normally be read by RDMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.
- Bits 63:32 of the entry are not all 0.
- An attempt to read the MSR indexed by bits 31:0 would cause a general-protection exception if executed via RDMSR with CPL = 0.

A VMX abort occurs if processing fails for any entry. See Section 30.7.

30.5 LOADING HOST STATE

Processor state is updated on VM exits in the following ways:

- Some state is loaded from or otherwise determined by the contents of the host-state area.
- Some state is determined by VM-exit controls.
- Some state is established in the same way on every VM exit.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order.

On processors that support Intel 64 architecture, the full values of each 64-bit field loaded (for example, the base address for GDTR) is loaded regardless of the mode of the logical processor before and after the VM exit.

The loading of host state is detailed in Section 30.5.1 to Section 30.5.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

A logical processor is in IA-32e mode after a VM exit only if the “host address-space size” VM-exit control is 1. If the logical processor was in IA-32e mode before the VM exit and this control is 0, a VMX abort occurs. See Section 30.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 30.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 30.6). This loading occurs only after the state loading described in this section.

30.5.1 Loading Host Control Registers, Debug Registers, MSRs

VM exits load new values for controls registers, debug registers, and some MSRs:

- CR0, CR3, and CR4 are loaded from the CR0 field, the CR3 field, and the CR4 field, respectively, with the following exceptions:
 - The following bits are not modified:
 - For CR0, ET, CD, NW; bits 63:32 (on processors that support Intel 64 architecture), 28:19, 17, and 15:6; and any bits that are fixed in VMX operation (see Section 26.8).¹
 - For CR3, bits 63:MAXPHYADDR, where MAXPHYADDR is defined as follows:

- Usually, MAXPHYADDR is the processor's supported physical-address width as enumerated in CPUID.80000008H:EAX[7:0] (this width is at most 52).
- If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.¹

This item applies only to processors that support Intel 64 architecture.

- For CR4, any bits that are fixed in VMX operation (see Section 26.8).
- CR4.PAE is set to 1 if the "host address-space size" VM-exit control is 1.
- CR4.PCIDE is set to 0 if the "host address-space size" VM-exit control is 0.
- CR4.FRED is set to 0 if the "host-address-space size" VM-exit control is 0.
- DR7 is set to 400H.
- If the "clear UINV" VM-exit control is 1, VM exit clears UINV.
- The following MSRs are established as follows:
 - The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
 - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since that field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
 - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP fields, respectively.

If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

If the processor supports the Intel 64 architecture with $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.²

 - The following steps are performed on processors that support Intel 64 architecture:
 - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 30.5.2).
 - The LMA and LME bits in the IA32_EFER MSR are each loaded with the setting of the "host address-space size" VM-exit control.
 - If the "load IA32_PERF_GLOBAL_CTRL" VM-exit control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field. Bits that are reserved in that MSR are maintained with their reserved values.
 - If the "load IA32_PAT" VM-exit control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field. Bits that are reserved in that MSR are maintained with their reserved values.
 - If the "load IA32_EFER" VM-exit control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field. Bits that are reserved in that MSR are maintained with their reserved values.
 - If the "clear IA32_BNDCFGS" VM-exit control is 1, the IA32_BNDCFGS MSR is cleared to 00000000_00000000H; otherwise, it is not modified.
 - If the "clear IA32_RTIT_CTL" VM-exit control is 1, the IA32_RTIT_CTL MSR is cleared to 00000000_00000000H; otherwise, it is not modified.
 - If the "load CET" VM-exit control is 1, the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are loaded from the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR fields, respectively.

1. Bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

2. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

If the processor supports the Intel 64 architecture with $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.

- If the “load PKRS” VM-exit control is 1, the IA32_PKRS MSR is loaded from the IA32_PKRS field. Bits 63:32 of that MSR are maintained with zeroes.
- If the “load FRED” VM-exit control is 1, the following MSRs are loaded from the corresponding fields: IA32_FRED_CONFIG, IA32_FRED_RSP1, IA32_FRED_RSP2, IA32_FRED_RSP3, IA32_FRED_STKLVL, IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-exit MSR-load area. See Section 30.6.

30.5.2 Loading Host Segment and Descriptor-Table Registers

Each of the registers CS, SS, DS, ES, FS, GS, and TR is loaded as follows (see below for the treatment of LDTR):

- The selector is loaded from the selector field. The segment is unusable if its selector is loaded with zero. The checks specified in Section 29.2.3 limit the selector values that may be loaded. In particular, CS and TR are never loaded with zero and are thus never unusable. SS can be loaded with zero only on processors that support Intel 64 architecture and only if the VM exit is to 64-bit mode (64-bit mode allows use of segments marked unusable).
- The base address is set as follows:
 - CS. Cleared to zero.
 - SS, DS, and ES. Undefined if the segment is unusable; otherwise, cleared to zero.
 - FS and GS. Undefined (but, on processors that support Intel 64 architecture, canonical) if the segment is unusable and the VM exit is not to 64-bit mode; otherwise, loaded from the base-address field.

If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.¹ The values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.

 - TR. Loaded from the host-state area. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.
- The segment limit is set as follows:
 - CS. Set to FFFFFFFFH (corresponding to a descriptor limit of FFFFFFFH and a G-bit setting of 1).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to FFFFFFFFH.
 - TR. Set to 00000067H.
- The type field and S bit are set as follows:
 - CS. Type set to 11 and S set to 1 (execute/read, accessed, non-conforming code segment).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, type set to 3 and S set to 1 (read/write, accessed, expand-up data segment).
 - TR. Type set to 11 and S set to 0 (busy 32-bit task-state segment).
- The DPL is set as follows:
 - CS, SS, and TR. Set to 0. The current privilege level (CPL) will be 0 after the VM exit completes.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 0.
- The P bit is set as follows:
 - CS, TR. Set to 1.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

- SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because the value of this control is also loaded into IA32_EFER.LMA (see Section 30.5.1), no VM exit is ever to compatibility mode (which requires IA32_EFER.LMA = 1 and CS.L = 0).
- D/B.
 - CS. Loaded with the inverse of the setting of the “host address-space size” VM-exit control. For example, if that control is 0, indicating a 32-bit guest, CS.D/B is set to 1.
 - SS. Set to 1.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.
- G.
 - CS. Set to 1.
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.

The host-state area does not contain a selector field for LDTR. LDTR is established as follows on all VM exits: the selector is cleared to 0000H, the segment is marked unusable and is otherwise undefined.

The base addresses for GDTR and IDTR are loaded from the GDTR base-address field and the IDTR base-address field, respectively. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N of each base address is set to the value of bit N-1 of that base address. The GDTR and IDTR limits are each set to FFFFH.

30.5.3 Loading Host RIP, RSP, RFLAGS, and SSP

RIP and RSP are loaded from the RIP field and the RSP field, respectively. RFLAGS is cleared, except bit 1, which is always set.

If the “load CET” VM-exit control is 1, SSP (shadow-stack pointer) is loaded from the SSP field.

30.5.4 Checking and Loading Host Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LMA = 0, the logical processor uses **PAE paging**. See Section 5.4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.¹ When in PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTes). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTes and, if they are valid, loads them into the processor (into internal, non-architectural registers).

A VM exit is to a VMM that uses PAE paging if (1) bit 5 (corresponding to CR4.PAE) is set in the CR4 field in the host-state area of the VMCS; and (2) the “host address-space size” VM-exit control is 0. Such a VM exit may check the validity of the PDPTes referenced by the CR3 field in the host-state area of the VMCS. Such a VM exit must check their validity if either (1) PAE paging was not in use before the VM exit; or (2) the value of CR3 is changing as a result of the VM exit. A VM exit to a VMM that does not use PAE paging must not check the validity of the PDPTes.

A VM exit that checks the validity of the PDPTes uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use. If MOV to CR3 would cause a general-protection exception due to the PDPTes that would be loaded (e.g., because a reserved bit is set), a VMX abort occurs (see Section 30.7). If a VM exit to a VMM that uses PAE does not cause a VMX abort, the PDPTes are loaded into the processor as would MOV to CR3, using the value of CR3 being load by the VM exit.

1. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

30.5.5 Updating Non-Register State

VM exits affect the non-register state of a logical processor as follows:

- A logical processor is always in the active state after a VM exit.
- Event blocking is affected as follows:
 - There is no blocking by STI or by MOV SS after a VM exit.
 - VM exits caused directly by non-maskable interrupts (NMIs) cause blocking by NMI (see Table 27-3). Other VM exits do not affect blocking by NMI. (See Section 30.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 31.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM exits invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP).
- VM exits are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

30.5.6 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 11.10.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. VM exits clear any address-range monitoring that may be in effect.

30.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 27.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101H (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 0000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM exits for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.¹

If processing fails for any entry, a VMX abort occurs. See Section 30.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM exit, the logical processor does not use any translations that were cached before the transition.

1. Note the following about processors that support Intel 64 architecture. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CR0.PG is always 1 in VMX operation, the IA32_EFER MSR should not be included in the VM-exit MSR-load area for the purpose of modifying the LME bit.

30.7 VMX ABORTS

A problem encountered during a VM exit leads to a **VMX abort**. A VMX abort takes a logical processor into a shut-down state as described below.

A VMX abort does not modify the VMCS data in the VMCS region of any active VMCS. The contents of these data are thus suspect after the VMX abort.

On a VMX abort, a logical processor saves a nonzero 32-bit VMX-abort indicator field at byte offset 4 in the VMCS region of the VMCS whose misconfiguration caused the failure (see Section 27.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 30.4).
2. Host checking of the page-directory-pointer-table entries (PDPTes) failed (see Section 30.5.4).
3. The current VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the logical processor cannot complete the VM exit properly.
4. There was a failure on loading host MSRs (see Section 30.6).
5. There was a machine-check event during VM exit (see Section 30.8).
6. The logical processor was in IA-32e mode before the VM exit and the “host address-space size” VM-exit control was 0 (see Section 30.5).

Some of these causes correspond to failures during the loading of state from the host-state area. Because the loading of such state may be done in any order (see Section 30.5) a VM exit that might lead to a VMX abort for multiple reasons (for example, the current VMCS may be corrupt and the host PDPTes might not be properly configured). In such cases, the VMX-abort indicator could correspond to any one of those reasons.

A logical processor never reads the VMX-abort indicator in a VMCS region and writes it only with one of the non-zero values mentioned above. The VMX-abort indicator allows software on one logical processor to diagnose the VMX-abort on another. For this reason, it is recommended that software running in VMX root operation zero the VMX-abort indicator in the VMCS region of any VMCS that it uses.

After saving the VMX-abort indicator, operation of a logical processor experiencing a VMX abort depends on whether the logical processor is in SMX operation:¹

- If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000DH, indicating “VMX abort.” See the Intel® Trusted Execution Technology Measured Launched Environment Programming Guide.
- If the logical processor is outside SMX operation, it issues a special bus cycle (to notify the chipset) and enters the **VMX-abort shutdown state**. RESET is the only event that wakes a logical processor from the VMX-abort shutdown state. The following events do not affect a logical processor in this state: machine-check events; INIT signals; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

30.8 MACHINE-CHECK EVENTS DURING VM EXIT

If a machine-check event occurs during VM exit, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM exit:
 - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 7, “Safer Mode Extensions Reference,” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

- If CR4.MCE = 1, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered to the guest normally.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- The machine-check event is handled after VM exit completes:
 - If the VM exit ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM exit ends with CR4.MCE = 1, a machine-check exception (#MC) is delivered to the host normally.
- A VMX abort is generated (see Section 30.7). The logical processor blocks events as done normally in VMX abort. The VMX abort indicator is 5, for “machine-check event during VM exit.”

The first option is not used if the machine-check event occurs after any host state has been loaded. The second option is used only if VM entry is able to load all host state.

30.9 USER-INTERRUPT RECOGNITION AFTER VM EXIT

A VM exit results in recognition of a pending user interrupt if it completes with CR4.UINTR = IA32_EFER.LMA = 1 and with UIRR ≠ 0; otherwise, no pending user interrupt is recognized.

15. Updates to Chapter 31, Volume 3C

Change bars and violet text show changes to Chapter 31 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- In Section 31.3.4.1, "Write Accesses That Are Eligible for Sub-Page Write Permissions," write access performed for architectural PEBS was added to the list of accesses where a sub-page write permission does not apply.

CHAPTER 31

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 31.1 details the architecture of VPIDs. Section 31.3 provides the details of EPT. Section 31.4 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.

31.1 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 31.4 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 34.14.)
 - 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 29.2.1.1.)

VPIDs and PCIDs (see Section 5.10.1) 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.

31.2 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 5.5, which describes the operation of 4-level paging and 5-level paging.

31.3 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 31.3.1 gives an overview of EPT.
- Section 31.3.2 describes operation of EPT-based address translation.
- Section 31.3.3 discusses VM exits that may be caused by EPT.
- Section 31.3.7 describes interactions between EPT and memory typing.

31.3.1 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 31.3.2 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 31.3.3.

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 27.6.2.
2. If HLAT paging is enabled, the processor uses the HLATP VMCS field instead of CR3. See Section 5.5.1. 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 5.4 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 5.4.1 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.

The widths of physical and guest-physical addresses are limited. CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. This width is at most 52 and limits the width of both physical and guest-physical addresses:

- **Physical addresses.** The limit on physical addresses is denoted MAXPHYADDR and is derived from CPUID.80000008H:EAX[7:0]. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.¹
For example, if CPUID.80000008H:EAX[7:0] = 48, IA32_TME_ACTIVATE[39:36] = 4, and TME has been configured, MAXPHYADDR is 44 outside SEAM (but remains 48 in SEAM).
- **Guest-physical addresses.** The width of these addresses is limited by the value enumerated by CPUID.80000008H:EAX[7:0] without any regard to TME or SEAM.

31.3.2 EPT Translation Mechanism

The EPT translation mechanism can be configured in either of two modes: **4-level EPT** or **5-level EPT**. 4-level EPT accesses at most 4 EPT paging-structure entries (an EPT page-walk length of 4) to translate a guest-physical address and uses only bits 47:0 of each guest-physical address. In contrast, 5-level EPT may access up to 5 EPT paging-structure entries (an EPT page-walk length of 5) and uses guest-physical address bits 56:0.²

The EPT page-walk length is configured using the extended-page-table pointer (EPTP), a VM-execution control field (see Table 27-9 in Section 27.6.11). Specifically, bits 5:3 contain a value one less than EPT page-walk length. Thus, a value of 3 configures 4-level EPT, while a value of 4 configures 5-level EPT.³

The remainder of this section describes the translation process used by 4-level EPT and 5-level EPT. Because the processes used by the two EPT modes are similar, they are described together in the following items (with any differences identified):

- With 5-level EPT, 4-KByte naturally aligned EPT PML5 table is located at the physical address specified in bits 51:12 of the EPTP. An EPT PML5 table comprises 512 64-bit entries (EPT PML5Es). An EPT PML5E 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 56:48 of the guest-physical address.⁴
 - Bits 2:0 are all 0.

-
1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.
 2. Physical addresses and guest-physical addresses are architecturally limited to 52 bits (e.g., by paging), so in practice bits 56:52 are zero. See Section 31.3.1
 3. 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.
 4. Bits 56:52 of each guest-physical address are necessarily zero because guest-physical addresses are architecturally limited to 52 bits.

Because an EPT PML5E is identified using bits 56:48 of the guest-physical address, it controls access to a 256-TByte region of the guest-physical-address space. The format of an EPT PML5E is given in Table 31-1.

Table 31-1. Format of an EPT PML5 Entry (PML5E) that References an EPT PML4 Table

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 256-TByte region controlled by this entry.
1	Write access; indicates whether writes are allowed to the 256-TByte 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 256-TByte 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 256-TByte 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 256-TByte region controlled by this entry (see Section 31.3.5). 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 256-TByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
M-1:12	Physical address of 4-KByte aligned EPT PML4 table referenced by this entry. ¹
51:M	Reserved (must be 0).
63:52	Ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 31.3.1.

- With 4-level EPT, bits 51:48 of the guest-physical address must all be zero; otherwise, an EPT violation occurs (see Section 31.3.3).
- A 4-KByte naturally aligned EPT PML4 table is located at the physical address specified in the EPTP (for 4-level EPT) or in the EPT PML5E (for 5-level EPT). 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 (for 4-level EPT) or from bits 51:12 of the EPT PML4E (for 5-level EPT).
 - 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 31-2.

- 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.

Table 31-2. 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 31.3.5). 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.
M-1:12	Physical address of 4-KByte aligned EPT page-directory-pointer table referenced by this entry. ¹
51:M	Reserved (must be 0).
63:52	Ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 31.3.1.

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:¹

- 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.
 - 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 31-3.

- 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 31-4.

1. 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 31-3. 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 31.3.7).
6	Ignore PAT memory type for this 1-GByte page (see Section 31.3.7).
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 31.3.5). 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 31.3.5). 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).
M-1:30	Physical address of the 1-GByte page referenced by this entry. ¹
51:M	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 31.3.3.2). 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 31.3.3.2). 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 31.3.3.2). 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 28.5.8.1). If “EPT-violation #VE” VM-execution control is 0, this bit is ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 31.3.1.

Table 31-4. 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 31.3.5). 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.
M-1:12	Physical address of 4-KByte aligned EPT page directory referenced by this entry. ¹
51:M	Reserved (must be 0).
63:52	Ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 31.3.1.

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 31-5.

- 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 31-6.

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 31-5. 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 31.3.7).
6	Ignore PAT memory type for this 2-MByte page (see Section 31.3.7).
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 31.3.5). 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 31.3.5). 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).
M-1:21	Physical address of the 2-MByte page referenced by this entry. ¹
51:M	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 31.3.3.2). 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 31.3.3.2). 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 31.3.3.2). 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 28.5.8.1). If “EPT-violation #VE” VM-execution control is 0, this bit is ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 31.3.1.

— 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 31-7.

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 31.3.3.2). (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 28.5.8.1. If the “EPT-violation #VE” VM-execution control is 0, this bit is ignored.)

Table 31-6. 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 31.3.5). 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.
M-1:12	Physical address of 4-KByte aligned EPT page table referenced by this entry. ¹
51:M	Reserved (must be 0).
63:52	Ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 31.3.1.

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 31.3.3.
- Interactions between the EPT translation mechanism and memory typing are described in Section 31.3.7.

Figure 31-1 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 31-1 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.)

31.3.3 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 31.3.3.1). 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 31.3.3.2). 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 31.3.6).

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 28.5.8.1.

31.3.3.1 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 31.3.2) 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 PDPTs 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 31-7. 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 31.3.7).
6	Ignore PAT memory type for this 4-KByte page (see Section 31.3.7).
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 31.3.5). 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 31.3.5). 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.
M-1:12	Physical address of the 4-KByte page referenced by this entry. ¹
51:M	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 31.3.3.2). 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 31.3.3.2). 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 31.3.3.2). 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 31.3.4). 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 28.5.8.1). If “EPT-violation #VE” VM-execution control is 0, this bit is ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 31.3.1.

- A reserved bit is set. This includes the setting of a bit in the range 51:12 in position MAXPHYADDR or above (see Section 31.3.1). See Section 31.3.2 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.

31.3.3.2 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 31.3.2).
- 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.

[illegible]

Figure 31-1. Formats of EPTP and EPT Paging-Structure Entries

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 31.3.1.
2. Supervisor shadow-stack control.
3. See Section 27.6.11 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 PDPT registers by the MOV to CR instruction for PAE paging; see Section 5.4.1. Those loads of guest PDPTs 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 31.3.4 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 5, “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 31.3.3.3 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.

31.3.3.3 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 31.3.1). 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 PML5 entry or an EPT PML4 entry):
 - a. If the entry is not present (see Section 31.3.2), an EPT violation occurs.
 - b. If the entry is present but its contents are not configured properly (see Section 31.3.3.1), 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 (or if instead it is an EPT PDPTE or 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 31.3.3.2), 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

-
1. When there is a restart of HLAT paging (see Section 5.5), 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 31.3.2.
 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.

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).
 - 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 31.3.3.2), 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 31.3.3.2 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 31.3.3.2), an EPT violation occurs. Otherwise, memory is accessed using the physical address.

31.3.4 Sub-Page Write Permissions

Section 31.3.3.2 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 31.3.4.1 identifies the data accesses that are eligible for sub-page write permissions. Section 31.3.4.2 explains how the processor determines whether to allow such an access.

31.3.4.1 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 31.3.3.2: 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 31.3.3.2.) 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 performed for architectural PEBS.
- 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 31.3.4.2.

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 32.4), 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.

31.3.4.2 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 2*S* 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 is set to 1. 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 31.3.3.2, 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 31.3.3.2; 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, SPPL3E, or SPPL2E), bits 11:1 are reserved, as are bits 63:MAXPHYADDR (see Section 31.3.1). 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.

31.3.5 Accessed and Dirty Flags for EPT

The Intel 64 architecture supports **accessed and dirty flags** in ordinary paging-structure entries (see Section 5.8). 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 27-9 in Section 27.6.11). 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 31.3.3.2).

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 31.3.3.2). 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 5.4.1. 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 31.4). 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.

31.3.6 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 27-7 in Section 27.6.2). 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 31.3.5). 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 31.3.5. 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).

31.3.7 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 14, “Memory Cache Control,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for details of memory typing in the Intel 64 architecture.) Section 31.3.7.1 explains how the memory type is determined for accesses to the EPT paging structures. Section 31.3.7.2 explains how the memory type is determined for an access using a guest-physical address that is translated using EPT.

31.3.7.1 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 27.6.11). 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.

31.3.7.2 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 14.12.3, “Selecting a Memory Type from the PAT.”²
- 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 31.3.3).
- 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 14-7 in Section 14.5.2.2, 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.

31.4 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 5.10, “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 31.4.1 describes the different kinds of information that may be cached. Section 31.4.2 specifies when such information may be cached and how it may be used. Section 31.4.3 details how software can invalidate cached information.

31.4.1 Information That May Be Cached

Section 5.10, “Caching Translation Information,” in the 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:

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.
2. Table 14-11 in Section 14.12.3, “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.

- **Linear mappings.**¹ 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.
- 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.**² 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.**³ 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 31.3.4.2).

31.4.2 Creating and Using Cached Translation Information

The following items detail the creation of the mappings described in the previous section:⁴

- 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 root

1. Earlier versions of this manual used the term “VPID-tagged” to identify linear mappings.

2. Earlier versions of this manual used the term “EPTP-tagged” to identify guest-physical mappings.

3. Earlier versions of this manual used the term “dual-tagged” to identify combined mappings.

4. 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 5.10.1, “Process-Context Identifiers (PCIDs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

table (a EPT PML4 table with 4-level EPT or a EPT PML5 table with 5-level EPT); the notation **EPTRTA** refers to those 40 bits. Newly created guest-physical mappings are associated with the current EPTRTA.

- Combined mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by the current EPTRTA. 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, the current PCID, and the current EPTRTA.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 31.3.2) or that are misconfigured (see Section 31.3.3.1).
- 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 EPTRTA. It may also use global TLB entries (combined mappings) associated with the current VPID, the current EPTRTA, and any PCID.
 - For accesses using guest-physical addresses, it may use guest-physical mappings associated with the current EPTRTA.
 - No linear mappings are used while EPT is in use.

31.4.3 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.

31.4.3.1 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 EPTRTAs). Linear

1. At any given time, a logical processor may be caching combined mappings for a VPID and a PCID that are associated with different EPTRTAs. Similarly, it may be caching combined mappings for an EPTRTA 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 5.10.4, “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.

mappings for the current VPID are invalidated even if EPT is in use.¹ Combined mappings for the current VPID are invalidated even if EPT is not in use.²

- An EPT violation invalidates any guest-physical mappings (associated with the current EPTRTA) 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 EPTRTA.
- 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 EPTRTAs.
- 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 EPTRTAs. (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 EPTRTAs. (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 EPTRTAs.
 - **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 EPTRTAs. The logical processor is **not** required to invalidate information that was used for **global** translations (although it may do so). See Section 5.10, “Caching Translation Information,” for details regarding global translations. (The instruction may also invalidate mappings associated with other VPIDs.)

See Chapter 33 for details of the INVVPID instruction. See Section 31.4.3.3 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 EPTRTA specified in the INVEPT descriptor. Combined mappings for that EPTRTA are invalidated for all VPIDs and all PCIDs. (The instruction may invalidate mappings associated with other EPTRTAs.)
 - **All-context.** If the INVEPT type is 2, the logical processor invalidates guest-physical mappings and combined mappings associated with all EPTRTAs (and, for combined mappings, for all VPIDs and PCIDs).

See Chapter 33 for details of the INVEPT instruction. See Section 31.4.3.4 for guidelines regarding use of this instruction.

- A power-up or a reset invalidates all linear mappings, guest-physical mappings, and combined mappings.

1. 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.

2. 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.

31.4.3.2 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.

31.4.3.3 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 5.10, “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 31.4.3.1); 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 32.4.5, 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.

31.4.3.4 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 31.3.7.)
 - 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 should use the INVEPT instruction with the “single-context” INVEPT type before a VM entry with an EPTP value X if the logical processor had earlier been in VMX non-root operation with an EPTP value Y such that $Y[5:3] \neq X[5:3]$ (different EPT page-walk length) and $Y[51:12] = X[51:12]$.
- Software may use the INVEPT instruction after modifying a present EPT paging-structure entry (see Section 31.3.2) 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 31.4.3.1), an EPT violation will not recur if the original access is performed again, even if the INVEPT instruction is not executed.

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.

- Because a logical processor does not cache any information derived from EPT paging-structure entries that are not present (see Section 31.3.2) or misconfigured (see Section 31.3.3.1), 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 32.4.5, 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.

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 shutdown.” A discussion of TLB shutdown appears in Section 5.10.5, “Propagation of Paging-Structure Changes to Multiple Processors,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

16. Updates to Chapter 32, Volume 3C

Change bars and violet 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:

- In Section 32.4.6.1, "Guest-Physical Accesses to the APIC-Access Page," writes by architectural PEBS was added to the list of guest-physical accesses when the guest-physical access is being translated using EPT.

CHAPTER 32

APIC VIRTUALIZATION AND VIRTUAL INTERRUPTS

The VMCS includes controls that enable the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC).

When these controls are used, the processor will emulate many accesses to the APIC, track the state of the virtual APIC, and deliver virtual interrupts — all in VMX non-root operation without a VM exit.¹

The processor tracks the state of the virtual APIC using a virtual-APIC page identified by the virtual-machine monitor (VMM). Section 32.1 discusses the virtual-APIC page and how the processor uses it to track the state of the virtual APIC.

The following are the VM-execution controls relevant to APIC virtualization and virtual interrupts (see Section 27.6 for information about the locations of these controls):

- **Virtual-interrupt delivery.** This control enables the evaluation and delivery of pending virtual interrupts (Section 32.2). It also enables the emulation of writes (memory-mapped or MSR-based, as enabled) to the APIC registers that control interrupt prioritization.
- **Use TPR shadow.** This control enables emulation of accesses to the APIC's task-priority register (TPR) via CR8 (Section 32.3) and, if enabled, via the memory-mapped or MSR-based interfaces.
- **Virtualize APIC accesses.** This control enables virtualization of memory-mapped accesses to the APIC (Section 32.4) by causing VM exits on accesses to a VMM-specified APIC-access page. Some of the other controls, if set, may cause some of these accesses to be emulated rather than causing VM exits.
- **Virtualize x2APIC mode.** This control enables virtualization of MSR-based accesses to the APIC (Section 32.5).
- **APIC-register virtualization.** This control allows memory-mapped and MSR-based reads of most APIC registers (as enabled) by satisfying them from the virtual-APIC page. It directs memory-mapped writes to the APIC-access page to the virtual-APIC page, following them by VM exits for VMM emulation.
- **Process posted interrupts.** This control allows software to post virtual interrupts in a data structure and send a notification to another logical processor; upon receipt of the notification, the target processor will process the posted interrupts by copying them into the virtual-APIC page (Section 32.6).
- **IPI virtualization.** This control enables the virtualization of interprocessor interrupts (Section 32.1.6).

"Virtualize APIC accesses", "virtualize x2APIC mode", "virtual-interrupt delivery", and "APIC-register virtualization" are all secondary processor-based VM-execution controls; if bit 31 of the primary processor-based VM-execution controls is 0, the processor operates as if these controls were all 0. "IPI virtualization" is a tertiary processor-based VM-execution control; if bit 17 of the primary processor-based VM-execution controls is 0, the processor operates as if "IPI virtualization" were 0. See Section 27.6.2.

32.1 VIRTUAL APIC STATE

The **virtual-APIC page** is a 4-KByte region of memory that the processor uses to virtualize certain accesses to APIC registers and to manage virtual interrupts. The physical address of the virtual-APIC page is the **virtual-APIC address**, a 64-bit VM-execution control field in the VMCS (see Section 27.6.8).

Depending on the settings of certain VM-execution controls, the processor may virtualize certain fields on the virtual-APIC page with functionality analogous to that performed by the local APIC. Section 32.1.1 identifies and defines these fields. Section 32.1.2, Section 32.1.3, Section 32.1.4, and Section 32.1.5 detail the actions taken to virtualize updates to some of these fields.

With the exception of fields corresponding to virtualized APIC registers (defined in Section 32.1.1), software may modify the virtual-APIC page referenced by the current VMCS of a logical processor in VMX non-root operation. (This is an exception to the general requirement given in Section 27.11.4.)

1. In most cases, it is not necessary for a virtual-machine monitor (VMM) to inject virtual interrupts as part of VM entry.

32.1.1 Virtualized APIC Registers

Depending on the setting of certain VM-execution controls, a logical processor may virtualize certain accesses to APIC registers using the following fields on the virtual-APIC page:

- **Virtual task-priority register (VTPR):** the 32-bit field located at offset 080H on the virtual-APIC page.
- **Virtual processor-priority register (VPPR):** the 32-bit field located at offset 0A0H on the virtual-APIC page.
- **Virtual end-of-interrupt register (VEOI):** the 32-bit field located at offset 0B0H on the virtual-APIC page.
- **Virtual interrupt-service register (VISR):** the 256-bit value comprising eight non-contiguous 32-bit fields at offsets 100H, 110H, 120H, 130H, 140H, 150H, 160H, and 170H on the virtual-APIC page. Bit x of the VISR is at bit position $(x \& 1FH)$ at offset $(100H \mid ((x \& E0H) \gg 1))$. The processor uses only the low 4 bytes of each of the 16-byte fields at offsets 100H, 110H, 120H, 130H, 140H, 150H, 160H, and 170H.
- **Virtual interrupt-request register (VIRR):** the 256-bit value comprising eight non-contiguous 32-bit fields at offsets 200H, 210H, 220H, 230H, 240H, 250H, 260H, and 270H on the virtual-APIC page. Bit x of the VIRR is at bit position $(x \& 1FH)$ at offset $(200H \mid ((x \& E0H) \gg 1))$. The processor uses only the low 4 bytes of each of the 16-Byte fields at offsets 200H, 210H, 220H, 230H, 240H, 250H, 260H, and 270H.
- **Virtual interrupt-command register (VICR_LO):** the 32-bit field located at offset 300H on the virtual-APIC page.
- **Virtual interrupt-command register (VICR_HI):** the 32-bit field located at offset 310H on the virtual-APIC page.

The VTPR field virtualizes the TPR whenever the “use TPR shadow” VM-execution control is 1. The other fields indicated above virtualize the corresponding APIC registers whenever the “virtual-interrupt delivery” VM-execution control is 1. (VICR_LO and VICR_HI also virtualize the ICR when the “IPI virtualization” VM-execution control is 1.)

32.1.2 TPR Virtualization

The processor performs **TPR virtualization** in response to the following operations: (1) virtualization of the MOV to CR8 instruction; (2) virtualization of a write to offset 080H on the APIC-access page; and (3) virtualization of the WRMSR instruction with ECX = 808H. See Section 32.3, Section 32.4.3, and Section 32.5 for details of when TPR virtualization is performed.

The following pseudocode details the behavior of TPR virtualization:

```

IF “virtual-interrupt delivery” is 0
    THEN
        IF VTPR[7:4] < TPR threshold (see Section 27.6.8)
            THEN cause VM exit due to TPR below threshold;
        FI;
    ELSE
        perform PPR virtualization (see Section 32.1.3);
        evaluate pending virtual interrupts (see Section 32.2.1);
    FI;

```

Any VM exit caused by TPR virtualization is trap-like: the instruction causing TPR virtualization completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

32.1.3 PPR Virtualization

The processor performs **PPR virtualization** in response to the following operations: (1) VM entry; (2) TPR virtualization; and (3) EOI virtualization. See Section 29.3.2.5, Section 32.1.2, and Section 32.1.4 for details of when PPR virtualization is performed.

PPR virtualization uses the guest interrupt status (specifically, SVI; see Section 27.4.2) and VTPR. The following pseudocode details the behavior of PPR virtualization:

```

IF VTPR[7:4] ≥ SVI[7:4]

```

```

    THEN VPPR := VTPR & FFH;
    ELSE VPPR := SVI & FOH;
FI;

```

PPR virtualization always clears bytes 3:1 of VPPR.

PPR virtualization is caused only by TPR virtualization, EOI virtualization, and VM entry. Delivery of a virtual interrupt also modifies VPPR, but in a different way (see Section 32.2.2). No other operations modify VPPR, even if they modify SVI, VISR, or VTPR.

32.1.4 EOI Virtualization

The processor performs **EOI virtualization** in response to the following operations: (1) virtualization of a write to offset 0B0H on the APIC-access page; and (2) virtualization of the WRMSR instruction with ECX = 80BH. See Section 32.4.3 and Section 32.5 for details of when EOI virtualization is performed. EOI virtualization occurs only if the “virtual-interrupt delivery” VM-execution control is 1.

EOI virtualization uses and updates the guest interrupt status (specifically, SVI; see Section 27.4.2). The following pseudocode details the behavior of EOI virtualization:

```

Vector := SVI;
VISR[Vector] := 0; (see Section 32.1.1 for definition of VISR)
IF any bits set in VISR
    THEN SVI := highest index of bit set in VISR
    ELSE SVI := 0;
FI;
perform PPR virtualization (see Section 32.1.3);
IF EOI_exit_bitmap[Vector] = 1 (see Section 27.6.8 for definition of EOI_exit_bitmap)
    THEN cause EOI-induced VM exit with Vector as exit qualification;
    ELSE evaluate pending virtual interrupts; (see Section 32.2.1)
FI;

```

Any VM exit caused by EOI virtualization is trap-like: the instruction causing EOI virtualization completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

32.1.5 Self-IPI Virtualization

The processor performs **self-IPI virtualization** in response to the following operations: (1) virtualization of a write to offset 300H on the APIC-access page; and (2) virtualization of the WRMSR instruction with ECX = 83FH. See Section 32.4.3 and Section 32.5 for details of when self-IPI virtualization is performed. Self-IPI virtualization occurs only if the “virtual-interrupt delivery” VM-execution control is 1.

Each operation that leads to self-IPI virtualization provides an 8-bit vector (see Section 32.4.3 and Section 32.5). Self-IPI virtualization updates the guest interrupt status (specifically, RVI; see Section 27.4.2). The following pseudocode details the behavior of self-IPI virtualization:

```

VIRR[Vector] := 1; (see Section 32.1.1 for definition of VIRR)
RVI := max{RVI, Vector};
evaluate pending virtual interrupts; (see Section 32.2.1)

```

32.1.6 IPI Virtualization

The processor performs **IPI virtualization** in response to the following operations: (1) virtualization of a write to offset 300H on the APIC-access page (Section 32.4.3); (2) virtualization of the WRMSR instruction with ECX = 830H (Section 32.5); and (3) virtualization of some executions of SENDUIPI (Section 32.7). IPI virtualization occurs only if the “IPI virtualization” VM-execution control is 1.

IPI virtualization uses virtual-interrupt posting, which is described in Section 32.1.6.1. Section 32.1.6.2 gives the details of the operation of IPI virtualization.

32.1.6.1 Virtual-Interrupt Posting

IPI virtualization is based on **virtual-interrupt posting**, a process that can direct virtual interrupts to a specific virtual processor. With virtual-interrupt posting, a hardware or software agent “posts” a virtual interrupt in a data structure (**posted-interrupt descriptor** or **PID**) and then sends an interrupt (notification) to the logical processor on which the target virtual processor is operating. When that logical processor receives the notification, it uses information in the PID to deliver the virtual interrupt to the virtual processor (see Section 32.6).

A PID is a 64-byte data structure. In an expected usage, there is one PID for each virtual processor; the virtual processor’s VMCS contains a pointer to its PID. A PID has the format shown in Table 32-1.

Table 32-1. Format of Posted-Interrupt Descriptor (PID)

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests (PIR)	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1.
256	Outstanding notification (ON)	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0.
257	Suppress notify (SN)	Setting this bit directs agents not to send notifications.
271:258	Reserved	Reserved.
279:272	Notify vector (NV)	Notifications will use this vector.
287:280	Reserved	Reserved.
319:288	Notify destination (NDST)	Notifications will be directed to this physical APIC ID.
511:320	Reserved	Reserved.

A hardware or software agent posts a virtual interrupt to a virtual processor with following steps:

1. Read the PIR field in the virtual processor’s PID and write it back atomically, setting the bit that corresponds to the virtual interrupt’s vector.
2. Read the notification-information field in the PID and write it back atomically, setting the ON bit if the ON and SN bits were both 0 in the value read. (Step #2 may be done atomically with step #1.)
3. If step #2 changed the ON bit from 0 to 1, send a notification. The notification is an ordinary interrupt sent to the physical APIC ID NDST with vector NV.

A processor’s response to the delivery of a notification is detailed in Section 32.6.

The use of virtual-interrupt posting for IPI virtualization is explained in Section 32.1.6.2.

32.1.6.2 IPI Virtualization Using Virtual-Interrupt Posting

Each operation that leads to IPI virtualization provides an 8-bit virtual vector V and a 32-bit virtual APIC ID T. IPI virtualization uses those values to initiate the indicated virtual IPI using the **PID-pointer table**.

The PID-pointer table is a data structure referenced by the PID-pointer table address, a field in the VMCS. Each entry in the PID-pointer table contains the following information:

- Bits 63:6 contain bits 63:6 of the 64-bit physical address of a PID (see Section 32.1.6.1).
- Bits 5:1 are reserved and must be 0.
- Bit 0 is a valid bit.

Each such address must be 64-byte aligned. The index of the last entry in the table is also a field in the VMCS.

When virtualizing an IPI, the CPU uses the virtual APIC ID T to select an entry from the PID-pointer table. It uses the address in that entry to locate a posted-interrupt descriptor (PID) and then posts a virtual interrupt with vector V in that PID. The following pseudocode details the behavior of IPI virtualization:

```

IF V < 16
    THEN APIC-write VM exit;           // illegal vector
ELSE IF T ≤ last PID-pointer index    // a field in the VMCS

```

```

THEN
    PID_ADDR := 8 bytes at (PID-pointer table address + (T << 3));
    IF PID_ADDR sets bits beyond the processor's physical-address width1 OR
        PID_ADDR[5:0] ≠ 000001b // PID pointer not valid or reserved bits set
    THEN APIC-write VM exit; // See Section 32.4.3.3
    ELSE
        PID_ADDR[0] := 0; // clear the valid bit before using as an address
        PIR := 32 bytes at PID_ADDR; // under lock
        PIR[V] := 1;
        store PIR at PID_ADDR; // release lock; corresponds to step #1 in Section 32.1.6.1
        NotifyInfo := 8 bytes at PID_ADDR + 32; // under lock
        IF NotifyInfo.ON = 0 AND NotifyInfo.SN = 0
            THEN
                NotifyInfo.ON := 1;
                SendNotify := 1;
            ELSE SendNotify := 0;
        FI;
        store NotifyInfo at PID_ADDR + 32; // release lock; corresponds to step #2 in Section 32.1.6.1
        IF SendNotify = 1
            THEN send an IPI specified by NotifyInfo.NDST and NotifyInfo.NV; // step #3 in Section 32.1.6.1
        FI;
    FI;
ELSE APIC-write VM exit; // virtual APIC ID beyond end of tables
FI;

```

The sending of the notification IPI is indicated by fields in the selected PID: NDST (PID[319:288]) and NV (PID[279:272]):

- If the local APIC is in xAPIC mode, this is the IPI that would be generated by writing NDST[15:8] (PID[303:296]) to ICR_HI[31:24] (offset 310H from IA32_APIC_BASE) and then writing NV to ICR_LO (offset 300H from IA32_APIC_BASE).
- If the local APIC is in x2APIC mode, this is the IPI that would be generated by executing WRMSR with ECX = 830H (ICR), EAX = NV, and EDX = NDST.

If the pseudocode specifies an APIC-write VM exit, this VM exit occurs as if there had been a write access to page offset 300H on the APIC-access page (see Section 32.4.3.3).

32.2 EVALUATION AND DELIVERY OF VIRTUAL INTERRUPTS

If the “virtual-interrupt delivery” VM-execution control is 1, certain actions in VMX non-root operation or during VM entry cause the processor to evaluate and deliver virtual interrupts.

Evaluation of virtual interrupts is triggered by certain actions change the state of the virtual-APIC page and is described in Section 32.2.1. This evaluation may result in recognition of a virtual interrupt. Once a virtual interrupt is recognized, the processor may deliver it within VMX non-root operation without a VM exit. Virtual-interrupt delivery is described in Section 32.2.2.

32.2.1 Evaluation of Pending Virtual Interrupts

If the “virtual-interrupt delivery” VM-execution control is 1, certain actions cause a logical processor to **evaluate pending virtual interrupts**.

The following actions cause the evaluation of pending virtual interrupts: VM entry; TPR virtualization; EOI virtualization; self-IPI virtualization; and posted-interrupt processing. See Section 29.3.2.5, Section 32.1.2, Section 32.1.4, Section 32.1.5, and Section 32.6 for details of when evaluation of pending virtual interrupts is performed. No other operations cause the evaluation of pending virtual interrupts, even if they modify RVI or VPPR.

1. Usually, the processor's physical-address width is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), the width is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

Evaluation of pending virtual interrupts uses the guest interrupt status (specifically, RVI; see Section 27.4.2). The following pseudocode details the evaluation of pending virtual interrupts:

```

IF logical processor in SEAM and RVI < 31
    THEN RVI := 0;
FI;
IF "interrupt-window exiting" is 0 AND
RVI[7:4] > VPPR[7:4] (see Section 32.1.1 for definition of VPPR)
    THEN recognize a pending virtual interrupt;
ELSE
    do not recognize a pending virtual interrupt;
FI;

```

Once recognized, a virtual interrupt may be delivered in VMX non-root operation; see Section 32.2.2. Note that, in SEAM, no virtual interrupt will be delivered with a vector lower than 31.

Evaluation of pending virtual interrupts is caused only by VM entry, TPR virtualization, EOI virtualization, self-IPI virtualization, and posted-interrupt processing. No other operations do so, even if they modify RVI or VPPR. The logical processor ceases recognition of a pending virtual interrupt following the delivery of a virtual interrupt.

32.2.2 Virtual-Interrupt Delivery

If a virtual interrupt has been recognized (see Section 32.2.1), it is delivered at an instruction boundary when the following conditions all hold: (1) RFLAGS.IF = 1; (2) there is no blocking by STI; (3) there is no blocking by MOV SS or by POP SS; and (4) the "interrupt-window exiting" VM-execution control is 0.

Virtual-interrupt delivery has the same priority as that of VM exits due to the 1-setting of the "interrupt-window exiting" VM-execution control.¹ Thus, non-maskable interrupts (NMIs) and higher priority events take priority over delivery of a virtual interrupt; delivery of a virtual interrupt takes priority over external interrupts and lower priority events.

Virtual-interrupt delivery wakes a logical processor from the same inactive activity states as would an external interrupt. Specifically, it wakes a logical processor from the states entered using the HLT and MWAIT instructions. It does not wake a logical processor in the shutdown state or in the wait-for-SIPI state.

Virtual-interrupt delivery updates the guest interrupt status (both RVI and SVI; see Section 27.4.2) and delivers an event within VMX non-root operation without a VM exit. The following pseudocode details the behavior of virtual-interrupt delivery (see Section 32.1.1 for definition of VISR, VIRR, and VPPR):

```

Vector := RVI;
VISR[Vector] := 1;
SVI := Vector;
VPPR := Vector & FOH;
VIRR[Vector] := 0;
IF any bits set in VIRR
    THEN RVI := highest index of bit set in VIRR
    ELSE RVI := 0;
FI;
IF logical processor is in SEAM and RVI < 31
    THEN RVI := 0;
FI;
cease recognition of any pending virtual interrupt;
IF transactional execution is in effect
    THEN abort transactional execution and transition to a non-transactional execution;
FI;
IF logical processor is in enclave mode

```

1. A logical processor never recognizes or delivers a virtual interrupt if the "interrupt-window exiting" VM-execution control is 1. Because of this, the relative priority of virtual-interrupt delivery and VM exits due to the 1-setting of that control is not defined.

```

    THEN cause an Asynchronous Enclave Exit (AEX) (see Chapter 40, "Enclave Exiting Events")
FI;
IF CR4.UINTR = 1 AND IA32_EFER.LMA = 1 AND Vector = UINV
    THEN virtualize user-interrupt notification identification and processing (see Section 32.2.3)
    ELSE deliver interrupt with Vector to guest software;
FI;

```

32.2.3 Virtualizing User-Interrupt Notifications

Section 9.5 describes the process of user-interrupt notification identification and processing. If the "virtual-interrupt delivery" VM-execution control is 1, this process is modified as described in the following paragraphs.

The virtualized form of user-interrupt notification identification begins as described in Section 32.2.2. Following this, instead of writing zero to the EOI register in the local APIC, the logical processor performs the initial steps of EOI virtualization:

```

VISR[V] := 0;
IF any bit is set in VISR
    THEN SVI := highest index of bit set in VISR
    ELSE SVI := 0;
FI;
perform PPR virtualization (Section 32.1.3);

```

Unlike EOI virtualization resulting from a guest write to the EOI register (as defined for virtual-interrupt delivery), the logical processor does not check the EOI-exit bitmap as part of this modified form of user-interrupt notification identification, and the corresponding VM exits cannot occur.

Following this modified form of user-interrupt notification identification, the logical processor then performs user-interrupt notification processing as specified in Section 9.5.2.

A logical processor is not interruptible during this modified form of user-interrupt notification identification or between it and any subsequent user-interrupt notification processing.

If the user-interrupt notification identification that precedes user-interrupt notification processing occurred while the logical processor was in the HLT state, the logical processor returns to the HLT state following user-interrupt notification processing.

32.3 VIRTUALIZING CR8-BASED TPR ACCESSES

In 64-bit mode, software can access the local APIC's task-priority register (TPR) through CR8. Specifically, software uses the MOV from CR8 and MOV to CR8 instructions (see Section 13.8.6, "Task Priority in IA-32e Mode"). This section describes how these accesses can be virtualized.

A virtual-machine monitor can virtualize these CR8-based APIC accesses by setting the "CR8-load exiting" and "CR8-store exiting" VM-execution controls, ensuring that the accesses cause VM exits (see Section 28.1.3). Alternatively, there are methods for virtualizing some CR8-based APIC accesses without VM exits.

Normally, an execution of MOV from CR8 or MOV to CR8 that does not fault or cause a VM exit accesses the APIC's TPR. However, such an execution are treated specially if the "use TPR shadow" VM-execution control is 1. The following items provide details:

- **MOV from CR8.** The instruction loads bits 3:0 of its destination operand with bits 7:4 of VTPR (see Section 32.1.1). Bits 63:4 of the destination operand are cleared.
- **MOV to CR8.** The instruction stores bits 3:0 of its source operand into bits 7:4 of VTPR; the remainder of VTPR (bits 3:0 and bits 31:8) are cleared. Following this, the processor performs TPR virtualization (see Section 32.1.2).

32.4 VIRTUALIZING MEMORY-MAPPED APIC ACCESSSES

When the local APIC is in xAPIC mode, software accesses the local APIC's control registers using a memory-mapped interface. Specifically, software uses linear addresses that translate to physical addresses on page frame indicated by the base address in the IA32_APIC_BASE MSR (see Section 13.4.4, "Local APIC Status and Location"). This section describes how these accesses can be virtualized.

A virtual-machine monitor (VMM) can virtualize these memory-mapped APIC accesses by ensuring that any access to a linear address that would access the local APIC instead causes a VM exit. This could be done using paging or the extended page-table mechanism (EPT). Another way is by using the 1-setting of the "virtualize APIC accesses" VM-execution control.

If the "virtualize APIC accesses" VM-execution control is 1, the logical processor treats specially memory accesses using linear addresses that translate to physical addresses in the 4-KByte **APIC-access page**.^{1,2} (The APIC-access page is identified by the **APIC-access address**, a field in the VMCS; see Section 27.6.8.)

In general, an access to the APIC-access page causes an **APIC-access VM exit**. APIC-access VM exits provide a VMM with information about the access causing the VM exit. Section 32.4.1 discusses the priority of APIC-access VM exits.

Certain VM-execution controls enable the processor to virtualize certain accesses to the APIC-access page without a VM exit. In general, this virtualization causes these accesses to be made to the virtual-APIC page instead of the APIC-access page.

NOTES

Unless stated otherwise, this section characterizes only linear accesses to the APIC-access page; an access to the APIC-access page is a linear access if (1) it results from a memory access using a linear address; and (2) the access's physical address is the translation of that linear address. Section 32.4.6 discusses accesses to the APIC-access page that are not linear accesses.

The distinction between the APIC-access page and the virtual-APIC page allows a VMM to share paging structures or EPT paging structures among the virtual processors of a virtual machine (the shared paging structures referencing the same APIC-access address, which appears in the VMCS of all the virtual processors) while giving each virtual processor its own virtual APIC (the VMCS of each virtual processor will have a unique virtual-APIC address).

Section 32.4.2 discusses when and how the processor may virtualize read accesses from the APIC-access page. Section 32.4.3 does the same for write accesses. When virtualizing a write to the APIC-access page, the processor typically takes actions in addition to passing the write through to the virtual-APIC page.

The discussion in those sections uses the concept of an **operation** within which these memory accesses may occur. For those discussions, an "operation" can be an iteration of a REP-prefixed string instruction, an execution of any other instruction, or delivery of an event through the IDT or with FRED.

The 1-setting of the "virtualize APIC accesses" VM-execution control may also affect accesses to the APIC-access page that do not result directly from linear addresses. This is discussed in Section 32.4.6.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 42.5.3 for details.

-
1. Even when addresses are translated using EPT (see Section 31.3), the determination of whether an APIC-access VM exit occurs depends on an access's physical address, not its guest-physical address. Even when CR0.PG = 0, ordinary memory accesses by software use linear addresses; the fact that CR0.PG = 0 means only that the identity translation is used to convert linear addresses to physical (or guest-physical) addresses.
 2. If EPT is enabled and there is write to a guest-physical address that translates to an address on the APIC-access page that is eligible for sub-page write permissions (see Section 31.3.4.1), the processor may treat the write as if the "virtualize APIC accesses" VM-execution control were 0 (and not apply the treatment specified in this section). For that reason, it is recommended that software not configure any guest-physical address that translates to an address on the APIC-access page to be eligible for sub-page write permissions.

32.4.1 Priority of APIC-Access VM Exits

The following items specify the priority of APIC-access VM exits relative to other events.

- The priority of an APIC-access VM exit due to a memory access is below that of any page fault or EPT violation that that access may incur. That is, an access does not cause an APIC-access VM exit if it would cause a page fault or an EPT violation.
- A memory access does not cause an APIC-access VM exit until after the accessed flags are set in the paging structures (including EPT paging structures, if enabled).
- A write access does not cause an APIC-access VM exit until after the dirty flags are set in the appropriate paging structure and EPT paging structure (if enabled).
- With respect to all other events, any APIC-access VM exit due to a memory access has the same priority as any page fault or EPT violation that the access could cause. (This item applies to other events that the access may generate as well as events that may be generated by other accesses by the same operation.)

These principles imply, among other things, that an APIC-access VM exit may occur during the execution of a repeated string instruction (including `INS` and `OUTS`). Suppose, for example, that the first n iterations (n may be 0) of such an instruction do not access the APIC-access page and that the next iteration does access that page. As a result, the first n iterations may complete and be followed by an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers reflect the completion of n iterations.

32.4.2 Virtualizing Reads from the APIC-Access Page

A read access from the APIC-access page causes an APIC-access VM exit if any of the following are true:

- The “use TPR shadow” VM-execution control is 0.
- The access is for an instruction fetch.
- The access is more than 32 bits in size.
- The access is part of an operation for which the processor has already virtualized a write to the APIC-access page.
- The access is not entirely contained within the low 4 bytes of a naturally aligned 16-byte region. That is, bits 3:2 of the access’s address are 0, and the same is true of the address of the highest byte accessed.

If none of the above are true, whether a read access is virtualized depends on the setting of the “APIC-register virtualization” and “virtual-interrupt delivery” VM-execution controls:

- A read access is virtualized if its page offset is 080H (task priority) regardless of the settings of the “APIC-register virtualization” and “virtual-interrupt delivery” VM-execution controls.
- If the “virtual-interrupt delivery” VM-execution control is 1, a read access is virtualized if its page offset is 0B0H (end of interrupt) or 300H (interrupt command — low).
- If “APIC-register virtualization” is 1, a read access is virtualized if it is entirely within one of the following ranges of offsets:
 - 020H–023H (local APIC ID);
 - 030H–033H (local APIC version);
 - 080H–083H (task priority);
 - 0B0H–0B3H (end of interrupt);
 - 0D0H–0D3H (logical destination);
 - 0E0H–0E3H (destination format);
 - 0F0H–0F3H (spurious-interrupt vector);
 - 100H–103H, 110H–113H, 120H–123H, 130H–133H, 140H–143H, 150H–153H, 160H–163H, or 170H–173H (in-service);
 - 180H–183H, 190H–193H, 1A0H–1A3H, 1B0H–1B3H, 1C0H–1C3H, 1D0H–1D3H, 1E0H–1E3H, or 1F0H–1F3H (trigger mode);

- 200H–203H, 210H–213H, 220H–223H, 230H–233H, 240H–243H, 250H–253H, 260H–263H, or 270H–273H (interrupt request);
- 280H–283H (error status);
- 300H–303H or 310H–313H (interrupt command);
- 320H–323H, 330H–333H, 340H–343H, 350H–353H, 360H–363H, or 370H–373H (LVT entries);
- 380H–383H (initial count); or
- 3E0H–3E3H (divide configuration).

In all other cases, the access causes an APIC-access VM exit.

A read access from the APIC-access page that is virtualized returns data from the corresponding page offset on the virtual-APIC page.¹

32.4.3 Virtualizing Writes to the APIC-Access Page

Whether a write access to the APIC-access page is virtualized depends on the settings of the VM-execution controls and the page offset of the access. Section 32.4.3.1 details when APIC-write virtualization occurs.

Unlike reads, writes to the local APIC have side effects; because of this, virtualization of writes to the APIC-access page may require emulation specific to the access's page offset (which identifies the APIC register being accessed). Section 32.4.3.2 describes this **APIC-write emulation**.

For some page offsets, it is necessary for software to complete the virtualization after a write completes. In these cases, the processor causes an **APIC-write VM exit** to invoke VMM software. Section 32.4.3.3 discusses APIC-write VM exits.

32.4.3.1 Determining Whether a Write Access is Virtualized

A write access to the APIC-access page causes an APIC-access VM exit if any of the following are true:

- The "use TPR shadow" VM-execution control is 0.
- The access is more than 32 bits in size.
- The access is part of an operation for which the processor has already virtualized a write (with a different page offset or a different size) to the APIC-access page.
- The access is not entirely contained within the low 4 bytes of a naturally aligned 16-byte region. That is, bits 3:2 of the access's address are 0, and the same is true of the address of the highest byte accessed.

If none of the above are true, whether a write access is virtualized depends on the settings of the "APIC-register virtualization", "virtual-interrupt delivery", and "IPI virtualization" VM-execution controls:

- A write access is virtualized if its page offset is 080H (task priority) regardless of the settings of the "APIC-register virtualization" and "virtual-interrupt delivery" VM-execution controls.
- If the "virtual-interrupt delivery" VM-execution control is 1, a write access is virtualized if its page offset is 0B0H (end of interrupt) or 300H (interrupt command — low).
- If the "IPI virtualization" VM-execution control is 1, a write access is virtualized if its page offset is 300H.
- If the "APIC-register virtualization" VM-execution control is 1, a write access is virtualized if it is entirely within one the following ranges of offsets:
 - 020H–023H (local APIC ID);
 - 080H–083H (task priority);
 - 0B0H–0B3H (end of interrupt);
 - 0D0H–0D3H (logical destination);
 - 0E0H–0E3H (destination format);

1. The memory type used for accesses that read from the virtual-APIC page is reported in bits 53:50 of the IA32_VMX_BASIC MSR (see Appendix A.1).

- 0F0H–0F3H (spurious-interrupt vector);
- 280H–283H (error status);
- 300H–303H or 310H–313H (interrupt command);
- 320H–323H, 330H–333H, 340H–343H, 350H–353H, 360H–363H, or 370H–373H (LVT entries);
- 380H–383H (initial count); or
- 3E0H–3E3H (divide configuration).

In all other cases, the access causes an APIC-access VM exit.

The processor virtualizes a write access to the APIC-access page by writing data to the corresponding page offset on the virtual-APIC page.¹ Following this, the processor performs certain actions after completion of the operation of which the access was a part.² APIC-write emulation is described in Section 32.4.3.2.

32.4.3.2 APIC-Write Emulation

If the processor virtualizes a write access to the APIC-access page, it performs additional actions after completion of an operation of which the access was a part. These actions are called **APIC-write emulation**.

The details of APIC-write emulation depend upon the page offset of the virtualized write access:³

- 080H (task priority). The processor clears bytes 3:1 of VTPR and then causes TPR virtualization (Section 32.1.2).
- 0B0H (end of interrupt). If the “virtual-interrupt delivery” VM-execution control is 1, the processor clears VEOI and then causes EOI virtualization (Section 32.1.4); otherwise, the processor causes an APIC-write VM exit (Section 32.4.3.3).
- 300H (interrupt command — low). If the “virtual-interrupt delivery” VM-execution control is 1, the processor checks the value of VICR_LO to determine whether the following are all true:
 - Reserved bits (31:20, 17:16, 13) and bit 12 (delivery status) are all 0.
 - Bits 19:18 (destination shorthand) are 01B (self).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bits 10:8 (delivery mode) are 000B (fixed).
 - Bits 7:4 (the upper half of the vector) are **not** 0000B.

If all of the items above are true, the processor performs self-IPI virtualization using the 8-bit vector in byte 0 of VICR_LO (Section 32.1.5).

If the “virtual-interrupt delivery” VM-execution control is 0, or if any of the items above are false, behavior depends on the setting of the “IPI virtualization” VM-execution control:

- If the “IPI virtualization” VM-execution control is 1, the processor checks the value of VICR_LO to determine whether the following are all true:
 - Reserved bits (31:20, 17:16, 13) and bit 12 (delivery status) are all 0.
 - Bits 19:18 (destination shorthand) are 00B (no shorthand).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bit 11 (destination mode) is 0 (physical).
 - Bits 10:8 (delivery mode) are 000B (fixed).

1. The memory type used for accesses that write to the virtual-APIC page is reported in bits 53:50 of the IA32_VMX_BASIC MSR (see Appendix A.1).

2. Recall that, for the purposes of this discussion, an operation is an iteration of a REP-prefixed string instruction, an execution of any other instruction, or delivery of an event through the IDT or with FRED.

3. For any operation, there can be only one page offset for which a write access was virtualized. This is because a write access is not virtualized if the processor has already virtualized a write access for the same operation with a different page offset.

If all of the items above are true, the processor performs IPI virtualization using the 8-bit vector in byte 0 of VICR_LO and the 8-bit APIC ID in VICR_HI[31:24] (Section 32.1.6); otherwise, the processor causes an APIC-write VM exit.

- If the “IPI virtualization” VM-execution control is 0, the processor causes an APIC-write VM exit.
- 310H–313H (interrupt command — high). The processor clears bytes 2:0 of VICR_HI. No other virtualization or VM exit occurs.
- Any other page offset. The processor causes an APIC-write VM exit.

APIC-write emulation takes priority over system-management interrupts (SMIs), INIT signals, and lower priority events. APIC-write emulation is not blocked if RFLAGS.IF = 0 or by the MOV SS, POP SS, or STI instructions.

If an operation causes a fault after a write access to the APIC-access page and before APIC-write emulation, and that fault is delivered without a VM exit, APIC-write emulation occurs after the fault is delivered and before the fault handler can execute. If an operation causes a VM exit (perhaps due to a fault) after a write access to the APIC-access page and before APIC-write emulation, the APIC-write emulation does not occur.

32.4.3.3 APIC-Write VM Exits

In certain cases, VMM software must be invoked to complete the virtualization of a write access to the APIC-access page. In this case, APIC-write emulation causes an **APIC-write VM exit**. (Section 32.4.3.2 details the cases that causes APIC-write VM exits.)

APIC-write VM exits are invoked by APIC-write emulation, and APIC-write emulation occurs after an operation that performs a write access to the APIC-access page. Because of this, every APIC-write VM exit is trap-like: it occurs after completion of the operation containing the write access that caused the VM exit (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

The basic exit reason for an APIC-write VM exit is “APIC write.” The exit qualification is the page offset of the write access that led to the VM exit.

As noted in Section 32.5, execution of WRMSR with ECX = 83FH (self-IPI MSR) can lead to an APIC-write VM exit if the “virtual-interrupt delivery” VM-execution control is 1; the exit qualification for the APIC-write VM exit is 3F0H. As noted in Section 32.1.6 and in Section 32.7, IPI virtualization and execution of SENDUIPI may lead to APIC-write VM exits; these VM exits produce an exit qualification of 300H.

32.4.4 Instruction-Specific Considerations

Certain instructions that use linear address may cause page faults even though they do not use those addresses to access memory. The APIC-virtualization features may affect these instructions as well:

- **CLFLUSH, CLFLUSHOPT.** With regard to faulting, the processor operates as if each of these instructions reads from the linear address in its source operand. If that address translates to one on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will flush the corresponding cache line on the virtual-APIC page instead of the APIC-access page.
- **ENTER.** With regard to faulting, the processor operates if ENTER writes to the byte referenced by the final value of the stack pointer (even though it does not if its size operand is non-zero). If that value translates to an address on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will cause the APIC-write emulation appropriate to the address’s page offset.
- **MASKMOVQ and MASKMOVDQU.** Even if the instruction’s mask is zero, the processor may operate with regard to faulting as if MASKMOVQ or MASKMOVDQU writes to memory (the behavior is implementation-specific). In such a situation, an APIC-access VM exit may occur.
- **MONITOR.** With regard to faulting, the processor operates as if MONITOR reads from the effective address in RAX. If the resulting linear address translates to one on the APIC-access page, the instruction may cause an

APIC-access VM exit.¹ If it does not, it will monitor the corresponding address on the virtual-APIC page instead of the APIC-access page.

- **PREFETCH.** An execution of the PREFETCH instruction that would result in an access to the APIC-access page does not cause an APIC-access VM exit. Such an access may prefetch data; if so, it is from the corresponding address on the virtual-APIC page.

Virtualization of accesses to the APIC-access page is principally intended for basic instructions such as AND, MOV, OR, TEST, XCHG, and XOR. Use of an instruction that normally operates on floating-point, SSE, AVX, or AVX-512 registers may cause an APIC-access VM exit unconditionally regardless of the page offset it accesses on the APIC-access page.

32.4.5 Issues Pertaining to Page Size and TLB Management

The 1-setting of the “virtualize APIC accesses” VM-execution is guaranteed to apply only if translations to the APIC-access address use a 4-KByte page. The following items provide details:

- If EPT is not in use, any linear address that translates to an address on the APIC-access page should use a 4-KByte page. Any access to a linear address that translates to the APIC-access page using a larger page may operate as if the “virtualize APIC accesses” VM-execution control were 0.
- If EPT is in use, any guest-physical address that translates to an address on the APIC-access page should use a 4-KByte page. Any access to a linear address that translates to a guest-physical address that in turn translates to the APIC-access page using a larger page may operate as if the “virtualize APIC accesses” VM-execution control were 0. (This is true also for guest-physical accesses to the APIC-access page; see Section 32.4.6.1.)

In addition, software should perform appropriate TLB invalidation when making changes that may affect APIC-virtualization. The specifics depend on whether VPIDs or EPT is being used:

- **VPIDs being used but EPT not being used.** Suppose that there is a VPID that has been used before and that software has since made either of the following changes: (1) set the “virtualize APIC accesses” VM-execution control when it had previously been 0; or (2) changed the paging structures so that some linear address translates to the APIC-access address when it previously did not. In that case, software should execute INVVPID (see “INVVPID— Invalidate Translations Based on VPID” in Section 33.3) before performing on the same logical processor and with the same VPID.²
- **EPT being used.** Suppose that there is an EPTP value that has been used before and that software has since made either of the following changes: (1) set the “virtualize APIC accesses” VM-execution control when it had previously been 0; or (2) changed the EPT paging structures so that some guest-physical address translates to the APIC-access address when it previously did not. In that case, software should execute INVEPT (see “INVEPT— Invalidate Translations Derived from EPT” in Section 33.3) before performing on the same logical processor and with the same EPTP value.³
- **Neither VPIDs nor EPT being used.** No invalidation is required.

Failure to perform the appropriate TLB invalidation may result in the logical processor operating as if the “virtualize APIC accesses” VM-execution control were 0 in responses to accesses to the affected address. (No invalidation is necessary if neither VPIDs nor EPT is being used.)

32.4.6 APIC Accesses Not Directly Resulting From Linear Addresses

Section 32.4 has described the treatment of accesses that use linear addresses that translate to addresses on the APIC-access page. This section considers memory accesses that do not result directly from linear addresses.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.
2. INVVPID should use either (1) the all-contexts INVVPID type; (2) the single-context INVVPID type with the VPID in the INVVPID descriptor; or (3) the individual-address INVVPID type with the linear address and the VPID in the INVVPID descriptor.
3. INVEPT should use either (1) the global INVEPT type; or (2) the single-context INVEPT type with the EPTP value in the INVEPT descriptor.

- An access is called a **guest-physical access** if (1) CR0.PG = 1;¹ (2) the “enable EPT” VM-execution control is 1;² (3) the access’s physical address is the result of an EPT translation; and (4) either (a) the access was not generated by a linear address; or (b) the access’s guest-physical address is not the translation of the access’s linear address. Section 32.4.6.1 discusses the treatment of guest-physical accesses to the APIC-access page.
- An access is called a **physical access** if (1) either (a) the “enable EPT” VM-execution control is 0; or (b) the access’s physical address is not the result of a translation through the EPT paging structures; and (2) either (a) the access is not generated by a linear address; or (b) the access’s physical address is not the translation of its linear address. Section 32.4.6.2 discusses the treatment of physical accesses to the APIC-access page.

32.4.6.1 Guest-Physical Accesses to the APIC-Access Page

Guest-physical accesses include the following when guest-physical addresses are being translated using EPT:

- Reads from the guest paging structures when translating a linear address (such an access uses a guest-physical address that is not the translation of that linear address).
- Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging (see Section 5.4).
- Updates to the accessed and dirty flags in the guest paging structures when using a linear address (such an access uses a guest-physical address that is not the translation of that linear address).
- Memory accesses by Intel Processor Trace when the “Intel PT uses guest physical addresses” VM-execution control is 1 (see Section 28.5.4).
- **Writes by architectural PEBS when the “PEBS uses guest physical addresses” VM-execution control is 1 (see Section 28.5.5).**

Every guest-physical access using a guest-physical address that translates to an address on the APIC-access page causes an APIC-access VM exit. Such accesses are never virtualized regardless of the page offset.

The following items specify the priority relative to other events of APIC-access VM exits caused by guest-physical accesses to the APIC-access page.

- The priority of an APIC-access VM exit caused by a guest-physical access to memory is below that of any EPT violation that that access may incur. That is, a guest-physical access does not cause an APIC-access VM exit if it would cause an EPT violation.
- With respect to all other events, any APIC-access VM exit caused by a guest-physical access has the same priority as any EPT violation that the guest-physical access could cause.

32.4.6.2 Physical Accesses to the APIC-Access Page

Physical accesses include the following:

- If the “enable EPT” VM-execution control is 0:
 - Reads from the paging structures when translating a linear address.
 - Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging (see Section 5.4).
 - Updates to the accessed and dirty flags in the paging structures.
- If the “enable EPT” VM-execution control is 1, accesses to the EPT paging structures (including updates to the accessed and dirty flags for EPT).
- Any of the following accesses made by the processor to support VMX non-root operation:
 - Accesses to the VMCS region.

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

2. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “enable EPT” VM-execution control were 0. See Section 27.6.2.

- Accesses to data structures referenced (directly or indirectly) by physical addresses in VM-execution control fields in the VMCS. These include the I/O bitmaps, the MSR bitmaps, and the virtual-APIC page.
- Accesses that effect transitions into and out of SMM.¹ These include the following:
 - Accesses to SMRAM during SMI delivery and during execution of RSM.
 - Accesses during SMM VM exits (including accesses to MSEG) and during VM entries that return from SMM.

A physical access to the APIC-access page may or may not cause an APIC-access VM exit. If it does not cause an APIC-access VM exit, it may access the APIC-access page or the virtual-APIC page. Physical write accesses to the APIC-access page may or may not cause APIC-write emulation or APIC-write VM exits.

The priority of an APIC-access VM exit caused by physical access is not defined relative to other events that the access may cause.

It is recommended that software not set the APIC-access address to any of the addresses used by physical memory accesses (identified above). For example, it should not set the APIC-access address to the physical address of any of the active paging structures if the “enable EPT” VM-execution control is 0.

32.5 VIRTUALIZING MSR-BASED APIC ACCESSES

When the local APIC is in x2APIC mode, software accesses the local APIC’s control registers using the MSR interface. Specifically, software uses the RDMSR and WRMSR instructions, setting ECX (identifying the MSR being accessed) to values in the range 800H–8FFH (see Section 13.12, “Extended XAPIC (x2APIC)”). This section describes how these accesses can be virtualized.

A virtual-machine monitor can virtualize these MSR-based APIC accesses by configuring the MSR bitmaps (see Section 27.6.9) to ensure that the accesses cause VM exits (see Section 28.1.3). Alternatively, there are methods for virtualizing some MSR-based APIC accesses without VM exits.

Normally, an execution of RDMSR or WRMSR that does not fault or cause a VM exit accesses the MSR indicated in ECX. However, such an execution treats some values of ECX in the range 800H–8FFH specially if the “virtualize x2APIC mode” VM-execution control is 1. The following items provide details:

- **RDMSR.** The instruction’s behavior depends on the setting of the “APIC-register virtualization” VM-execution control.
 - If the “APIC-register virtualization” VM-execution control is 0, behavior depends upon the value of ECX.
 - If ECX contains 808H (indicating the TPR MSR), the instruction reads the 8 bytes from offset 080H on the virtual-APIC page (VTPR and the 4 bytes above it) into EDX:EAX. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not x2APIC mode).
 - If ECX contains any other value in the range 800H–8FFH, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a readable APIC register, EDX and EAX are loaded with the value of that register. If the local APIC is not in x2APIC mode or ECX does not indicate a readable APIC register, a general-protection fault occurs.
 - If “APIC-register virtualization” is 1 and ECX contains a value in the range 800H–8FFH, the instruction reads the 8 bytes from offset X on the virtual-APIC page into EDX:EAX, where $X = (ECX \& FFH) \ll 4$. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not in x2APIC mode).
- **WRMSR.** The instruction’s behavior depends on the value of ECX and the setting of the “virtual-interrupt delivery” and “IPI virtualization” VM-execution controls.

Special processing applies in the following cases: (1) ECX contains 808H (indicating the TPR MSR); (2) ECX contains 80BH (indicating the EOI MSR) and the “virtual-interrupt delivery” VM-execution control is 1; (3) ECX contains 83FH (indicating the self-IPI MSR) and the “virtual-interrupt delivery” VM-execution control is 1; and (4) ECX contains 830H (indicating the ICR MSR) and the “IPI virtualization” VM-execution control is 1.

1. Technically, these accesses do not occur in VMX non-root operation. They are included here for clarity.

If special processing applies, no general-protection exception is produced due to the fact that the local APIC is in xAPIC mode. However, WRMSR does perform the normal reserved-bit checking:

- If ECX contains 808H or 83FH, a general-protection fault occurs if either EDX or EAX[31:8] is non-zero.
- If ECX contains 80BH, a general-protection fault occurs if either EDX or EAX is non-zero.
- If ECX contains 830H, a general-protection fault occurs if any of bits 31:20, 17:16, or 13 of EAX is non-zero.

If there is no fault, WRMSR stores EDX:EAX at offset X on the virtual-APIC page, where $X = (ECX \& FFH) \ll 4$. Following this, the processor performs an operation depending on the value of ECX:

- If ECX contains 808H, the processor performs TPR virtualization (see Section 32.1.2).
- If ECX contains 80BH, the processor performs EOI virtualization (see Section 32.1.4).
- If ECX contains 83FH, the processor then checks the value of EAX[7:4] and proceeds as follows:
 - If the value is non-zero, the logical processor performs self-IPI virtualization with the 8-bit vector in EAX[7:0] (see Section 32.1.5).
 - If the value is zero, the logical processor causes an APIC-write VM exit as if there had been a write access to page offset 3F0H on the APIC-access page (see Section 32.4.3.3).
- If ECX contains 830H, the processor then checks the value of VICR to determine whether the following are all true:
 - Bits 19:18 (destination shorthand) are 00B (no shorthand).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bit 12 (unused) is 0.
 - Bit 11 (destination mode) is 0 (physical).
 - Bits 10:8 (delivery mode) are 000B (fixed).

If all of the items above are true, the processor performs IPI virtualization using the 8-bit vector in byte 0 of VICR and the 32-bit APIC ID in VICR[63:32] (see Section 32.1.6). Otherwise, the logical processor causes an APIC-write VM exit (see Section 32.4.3.3).

If special processing does not apply, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a writable APIC register, the value in EDX:EAX is written to that register. If the local APIC is not in x2APIC mode or ECX does not indicate a writable APIC register, a general-protection fault occurs.

32.6 POSTED-INTERRUPT PROCESSING

Posted-interrupt processing is a feature by which a processor processes the virtual interrupts by recording them as pending on the virtual-APIC page.

Posted-interrupt processing is enabled by setting the “process posted interrupts” VM-execution control. The processing is performed in response to the arrival of an interrupt with the **posted-interrupt notification vector**. In response to such an interrupt, the processor processes virtual interrupts recorded in a data structure called a **posted-interrupt descriptor (PID)**. The posted-interrupt notification vector and the address of the PID are fields in the VMCS; see Section 27.6.8.

If the “process posted interrupts” VM-execution control is 1, a logical processor uses a 64-byte posted-interrupt descriptor located at the posted-interrupt descriptor address. Table 32-2 gives the format of a PID:¹

Use of the PID differs from that of other data structures that are referenced by pointers in a VMCS. There is a general requirement that software ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. That requirement does not apply to the posted-interrupt descriptor. There is a requirement, however, that such modifications be done using locked read-modify-write instructions or other atomic operations.

1. Table 32-1 gives the same format; it is repeated here for the reader, omitting fields that are not used by posted-interrupt processing.

Table 32-2. Format of Posted-Interrupt Descriptor (PID)

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests (PIR)	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1.
256	Outstanding notification (ON)	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0.
511:257	Reserved or used for virtual-interrupt posting	Some of these bits are used by virtual-interrupt posting (Section 32.1.6.1). Posted-interrupt processing does not use or modify these bits.

If the “external-interrupt exiting” VM-execution control is 1, any unmasked external interrupt causes a VM exit (see Section 28.2). If the “process posted interrupts” VM-execution control is also 1, this behavior is changed and the processor handles an external interrupt as follows:¹

1. The local APIC is acknowledged; this provides the processor core with an interrupt vector, called here the **physical vector**.
2. If the physical vector equals the posted-interrupt notification vector, the logical processor continues to the next step. Otherwise, a VM exit occurs as it would normally due to an external interrupt; the vector is saved in the exiting-event identification field.
3. The processor clears the outstanding-notification bit in the posted-interrupt descriptor. This is done atomically so as to leave the remainder of the descriptor unmodified (e.g., with a locked AND operation).
4. The processor writes zero to the EOI register in the local APIC; this dismisses the interrupt with the posted-interrupt notification vector from the local APIC.
5. The logical processor performs a logical-OR of PIR into VIRR and clears PIR. No other agent can read or write a PIR bit (or group of bits) between the time it is read (to determine what to OR into VIRR) and when it is cleared.
6. The logical processor sets RVI to be the maximum of the old value of RVI and the highest index of all bits that were set in PIR; if no bit was set in PIR, RVI is left unmodified.
7. The logical processor evaluates pending virtual interrupts as described in Section 32.2.1.

The logical processor performs the steps above in an uninterruptible manner. If step #7 leads to recognition of a virtual interrupt, the processor may deliver that interrupt immediately.

Steps #1 to #7 above occur when the interrupt controller delivers an unmasked external interrupt to the CPU core. The following items consider certain cases of interrupt delivery:

- Interrupt delivery can occur between iterations of a REP-prefixed instruction (after at least one iteration has completed but before all iterations have completed). If this occurs, the following items characterize processor state after posted-interrupt processing completes and before guest execution resumes:
 - RIP references the REP-prefixed instruction;
 - RCX, RSI, and RDI are updated to reflect the iterations completed; and
 - RFLAGS.RF = 1.
- Interrupt delivery can occur when the logical processor is in the active, HLT, or MWAIT states. If the logical processor had been in the active or MWAIT state before the arrival of the interrupt, it is in the active state following completion of step #7; if it had been in the HLT state, it returns to the HLT state after step #7 (if a pending virtual interrupt was recognized, the logical processor may immediately wake from the HLT state).
- Interrupt delivery can occur while the logical processor is in enclave mode. If the logical processor had been in enclave mode before the arrival of the interrupt, an Asynchronous Enclave Exit (AEX) may occur before the steps #1 to #7 (see Chapter 40, “Enclave Exiting Events”). If no AEX occurs before step #1 and a VM exit occurs at step #2, an AEX occurs before the VM exit is delivered.

1. VM entry ensures that the “process posted interrupts” VM-execution control is 1 only if the “external-interrupt exiting” VM-execution control is also 1. See Section 29.2.1.1.

32.7 VIRTUALIZING SENDUIPI

The user-interrupt feature includes the SENDUIPI instruction that software operating with CPL = 3 can use to send user interrupts to another software thread (“user IPIs”). The SENDUIPI instruction has the following high-level operation:

- read selected entry from user-interrupt target table;
- use address in entry to read the referenced user posted-interrupt descriptor (UPID);
- update certain fields in UPID;
- if necessary, send ordinary IPI indicated in UPID’s notification information;

The last step uses two fields in the UPID: an 8-bit notification vector (UPID.NV) and a 32-bit notification destination (an APIC ID, UPID.NDST). Outside of VMX non-root operation, the processor implements the last step as follows:

- If the local APIC is in xAPIC mode, it writes UPID.NDST[15:8] to ICR_HI[31:24] (offset 310H from IA32_APIC_BASE) and then writes UPID.NV to ICR_LO (offset 300H).
- If the local APIC is in x2APIC mode, it performs the control-register write that would be done by an execution of WRMSR with ECX = 310H (ICR), EAX = UPID.NV, and EDX = UPID.NDST.

In VMX non-root operation, implementation of the step depends on the settings of the “use TPR shadow,” “virtualize APIC accesses,” and “IPI virtualization” VM-execution controls:¹

1. If the “use TPR shadow” VM-execution control is 0, the behavior is not modified: the logical processor sends the specified IPI by writing to the local APIC’s ICR as specified above (based on the current mode of the local APIC).
2. If the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” VM-execution control is 0, the logical processor virtualizes the sending of an x2APIC-mode IPI with the following steps:
 - a. The 64-bit value Z is written to offset 300H on the virtual-APIC page (VICR), where Z[7:0] = UPID.NV (the 8-bit virtual vector), Z[63:32] = UPID.NDST (the 32-bit virtual APIC ID) and Z[31:8] = 000000H (indicating a physically addressed fixed-mode IPI).
 - b. If the “IPI virtualization” VM-execution control is 1, IPI virtualization (Section 32.1.6) is performed using the vector UPID.NV and the 32-bit virtual APIC ID UPID.NDST.
3. If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1, the logical processor virtualizes the sending of an xAPIC-mode IPI by performing the following steps:
 - a. The 32-bit value X is written to offset 310H on the virtual-APIC page (VICR_HI), where X[31:24] = UPID.NDST[15:8] (the 8-bit virtual APIC ID) and X[23:0] = 000000H.²
 - b. The 32-bit value Y is written to offset 300H on the virtual-APIC page (VICR_LO), where Y[7:0] = UPID.NV (the 8-bit virtual vector) and Y[31:8] = 000000H (indicating a physically addressed fixed-mode IPI).
 - c. If the “IPI virtualization” VM-execution control is 1, IPI virtualization is performed using the vector UPID.NV and the APIC ID UPID.NDST[15:8]. IPI virtualization will use only the 8-bit APIC ID from bits 15:8 of the UPID’s destination field (the 8-bit value written earlier to bits 31:24 of VICR_HI).
4. If the “use TPR shadow” VM-execution control is 1 and the “IPI virtualization” VM-execution control is 0, an APIC-write VM exit occurs as if there had been a write access to page offset 300H on the APIC-access page (see Section 32.4.3.3).

1. The setting of the “virtualize x2APIC mode” VM-execution control does not affect this operation.

2. For xAPIC mode (which is virtualized if the “virtualize APIC accesses” VM-execution control is 1), the destination APIC ID is in byte 1 (**not** byte 0) of the UPID’s 4-byte NDST field.

17. Updates to Chapter 41, Volume 3D

Change bars and violet text show changes to Chapter 41 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- In the EUPDATESVN leaf function (Section 41.3), updated the Operation section. The temp variable TMP_KEY size was corrected to 16 bytes and pseudocode was updated accordingly.

CHAPTER 41

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) and ENCLU (user) instruction mnemonics. Different leaf functions are encoded by specifying an input value in the EAX register of the respective instruction mnemonic.

41.1 INTEL® SGX INSTRUCTION SYNTAX AND OPERATION

ENCLS and ENCLU 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.

41.1.1 ENCLS Register Usage Summary

Table 41-1 summarizes the implicit register usage of supervisor mode enclave instructions.

Table 41-1. 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)	
EDBGDR	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)	
EUPDATESVN	018H			
EA: Effective Address				

41.1.2 ENCLU Register Usage Summary

Table 41-2 summarizes the implicit register usage of user mode enclave instructions.

Table 41-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDY
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)
EDECSSA	09H (In)			
EA: Effective Address				

41.1.3 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 41-3 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 41-3. 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
SGX_EPC_PAGE_CONFLICT	7	EBLOCK, EMODPR, EMODT, EUPDATESVN
SGX_INVALID_SIGNATURE	8	EINIT
SGX_MAC_COMPARE_FAIL	9	ELDB, ELDU
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_ENTRYPOCH_LOCKED	15	EBLOCK
SGX_INVALID_EINITTOKEN	16	EINIT
SGX_PREV_TRK_INCMPL	17	ETRACK
SGX_PG_IS_SECS	18	EBLOCK
SGX_PAGE_ATTRIBUTES_MISMATCH	19	EACCEPT, EACCEPTCOPY
SGX_PAGE_NOT_MODIFIABLE	20	EMODPR, EMODT

Table 41-3. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
SGX_PAGE_NOT_DEBUGGABLE	21	EDBGDR, EDBGWR
SGX_INSUFFICIENT_ENTROPY	29	EUPDATESVN
SGX_EPC_NOT_READY	30	EUPDATESVN
SGX_NO_UPDATE	31	EUPDATESVN
SGX_INVALID_CPUSVN	32	EINIT, EGETKEY
SGX_INVALID_ISVSVN	64	EGETKEY
SGX_UNMASKED_EVENT	128	EINIT
SGX_INVALID_KEYNAME	256	EGETKEY

41.1.4 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 41-4 appear at various places in the pseudo-code of this document. They are used to enhance understanding of the operations.

Table 41-4. 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
CR_SAVE_DEBUGCTL	64	LP
CR_SAVE_PEBS_ENABLE	64	LP
CR_CPUSVN	128	PACKAGE
CR_SGXOWNERPOCH	128	PACKAGE
CR_SAVE_XCRO	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

Table 41-4. List of Internal CREG

Name	Size (Bits)	Scope
CR_SAVE_IA32_U_CET	64	LP
CR_SAVE_SSP	64	LP

41.1.5 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 leaves 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.

41.1.5.1 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.

Table 41-5. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions	
			Access	On Conflict
EACCEPT	Target	[DS:RCX]	Shared	#GP
	SECINFO	[DS:RBX]	Concurrent	

Table 41-5. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions	
			Access	On Conflict
EACCEPTCOPY	Target	[DS:RCX]	Concurrent	
	Source	[DS:RDX]	Concurrent	
	SECINFO	[DS:RBX]	Concurrent	
EADD	Target	[DS:RCX]	Exclusive	#GP
	SECS	[DS:RBX]PAGEINFO. SECS	Shared	#GP
EAUG	Target	[DS:RCX]	Exclusive	#GP
	SECS	[DS:RBX]PAGEINFO. SECS	Shared	#GP
EBLOCK	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT
ECREATE	SECS	[DS:RCX]	Exclusive	#GP
EDBGD	Target	[DS:RCX]	Shared	#GP
EDBGWR	Target	[DS:RCX]	Shared	#GP
EENTER	TCS	[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	
EINIT	SECS	[DS:RCX]	Shared	#GP
ELDB/ELDU	Target	[DS:RCX]	Exclusive	#GP
	VA	[DS:RDX]	Shared	#GP
	SECS	[DS:RBX]PAGEINFO. SECS	Shared	#GP
EMODPE	Target	[DS:RCX]	Concurrent	
	SECINFO	[DS:RBX]	Concurrent	
EMODPR	Target	[DS:RCX]	Shared	#GP
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT
EPA	VA	[DS:RCX]	Exclusive	#GP
EREMOVE	Target	[DS:RCX]	Exclusive	#GP
EREPORT	TARGETINFO	[DS:RBX]	Concurrent	
	REPORTDATA	[DS:RCX]	Concurrent	
	OUTPUTDATA	[DS:RDX]	Concurrent	
ERESUME	TCS	[DS:RBX]	Shared	#GP
ETRACK	SECS	[DS:RCX]	Shared	#GP
EWB	Source	[DS:RCX]	Exclusive	#GP
	VA	[DS:RDX]	Shared	#GP

Table 41-6. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
			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	
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	
EENTER	TCS	[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	
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	
EMODPE	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EMODPR	Target	[DS:RCX]	Exclusive	SGX_EPC_ PAGE_CON FLICT	Concurrent		Concurrent	
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_ PAGE_CON FLICT	Concurrent		Concurrent	
EPA	VA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREMOVE	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	

Table 41-6. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
			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	
EResume	TCS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
ETrack	SECS	[DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT
EUpdateSVN	EPCM		Exclusive	SGX_EPC_PAGE_CONFLICT	Exclusive	SGX_EPC_PAGE_CONFLICT	Exclusive	SGX_EPC_PAGE_CONFLICT
EWB	Source	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	

41.2 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 OF 01 CF ENCLS	Z0	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
Z0	NA	NA	NA	See Section 41.3

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, 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.12H.00H:EAX.SGX1 = 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.12H.00H:EAX.SGX1[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.12H.00H:EAX.SGX1[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 OF 01 D7 ENCLU	Z0	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
Z0	NA	NA	NA	See Section 41.4

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.12H.00H:EAX.SGX1 = 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.12H.00H:EAX.SGX1 = 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 or EAX = 9)
(* EREPORT, EGETKEY, EEXIT, EACCEPT, EMODPE, EACCEPTCOPY, or EDECCSSA *)
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.12H.00H:EAX.SGX1[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.12H.00H:EAX.SGX1[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.

41.3 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 41-7. Base Concurrency Restrictions of EADD

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EADD	Target [DS:RCX]	Exclusive	#GP
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP

Table 41-8. Additional Concurrency Restrictions of EADD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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_SRCPE	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_SRCPE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECS;
TMP_SECINFO := DS:RBX.SECINFO;
TMP_LINADDR := DS:RBX.LINADDR;

IF (DS:TMP_SRCPE 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.12H.01H:EAX[6] = 1) or
    (SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST and CPUID.12H.01H: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
        #GP(0); 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_SRCPE[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.07H.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 be 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;
            ELSE
                (* Check the 8 bytes are zero *)
                IF ( DS:RCX[4095:4088] != 0 ) #GP(0); FI;
            FI;
        FI;

```

```

    IF (SCRATCH_SECSINFO.FLAGS.W = 0 OR SCRATCH_SECSINFO.FLAGS.R = 0 OR
        SCRATCH_SECSINFO.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_SECSINFO.FLAGS.PT = PT_TCS)
    THEN
        SCRATCH_SECSINFO.FLAGS.R := 0;
        SCRATCH_SECSINFO.FLAGS.W := 0;
        SCRATCH_SECSINFO.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_SECSINFO[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_SECSINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECSINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECSINFO.FLAGS.X;
EPCM(DS:RCX).PT := SCRATCH_SECSINFO.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 PAGEINFO (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 41-9. Base Concurrency Restrictions of EAUG

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EAUG	Target [DS:RCX]	Exclusive	#GP
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP

Table 41-10. Additional Concurrency Restrictions of EAUG

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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 is not available for EAUG)
    THEN
        #GP(0); 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.12H.01H: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 be 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.07H.00H: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 41-11. EBLOCK Return Value in RAX

Error Code (see Table 41-3)	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_ENTRYPOCH_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 41-12. Base Concurrency Restrictions of EBLOCK**

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EBLOCK	Target [DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT

Table 41-13. Additional Concurrency Restrictions of EBLOCK

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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 42.7.

Concurrency Restrictions**Table 41-14. Base Concurrency Restrictions of ECREATE**

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
ECREATE	SECS [DS:RCX]	Exclusive	#GP

Table 41-15. Additional Concurrency Restrictions of ECREATE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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 #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_SECS.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.07H.00H:EDX.CET_IBT = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||

(CPUID.07H.00H:EDX.CET_IBT = 0 and DS:TMP_SECS.CET_ATTRIBUTES[5:2] ≠ 0) ||

(CPUID.07H.00H: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.12H.00H: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 42.7.2.2*)

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.12H.00H:EDX[7:0])))

THEN #GP(0); FI;

IF ((DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.12H.00H: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 (CUID.07H.00H: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);
```

```
(* 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 41-16. EDBGRD Return Value in RAX

Error Code (see Table 41-3)	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 41-17. Base Concurrency Restrictions of EDBGD

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EDBGD	Target [DS:RCX]	Shared	#GP

Table 41-18. Additional Concurrency Restrictions of EDBGD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDBGD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDBGD 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] := 0FFFFFFFFFFFFFFFFFH;
                    ELSE RBX[63:0] := 0H;
                FI;
            ELSE
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN EBX[31:0] := 0FFFFFFFFFH;
                    ELSE EBX[31:0] := 0H;
                FI;
            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 41-19. EDBGWR Return Value in RAX

Error Code (see Table 41-3)	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 EDBGWR does not result in a #GP.

Concurrency Restrictions

Table 41-20. Base Concurrency Restrictions of EDBGWR

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EDBGWR	Target [DS:RCX]	Shared	#GP

Table 41-21. Additional Concurrency Restrictions of EDBGWR

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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:RCX).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 comprising 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 41-22. Base Concurrency Restrictions of EEXTEND

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EEXTEND	Target [DS:RCX]	Shared	#GP
	SECS [DS:RBX]	Concurrent	

Table 41-23. Additional Concurrency Restrictions of EEXTEND

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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_ENCLAVEOFFS ET	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:

$$q1 = \text{floor}(\text{Signature}^2 / \text{Modulus});$$

$$q2 = \text{floor}((\text{Signature}^3 - q1 * \text{Signature} * \text{Modulus}) / \text{Modulus});$$

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.

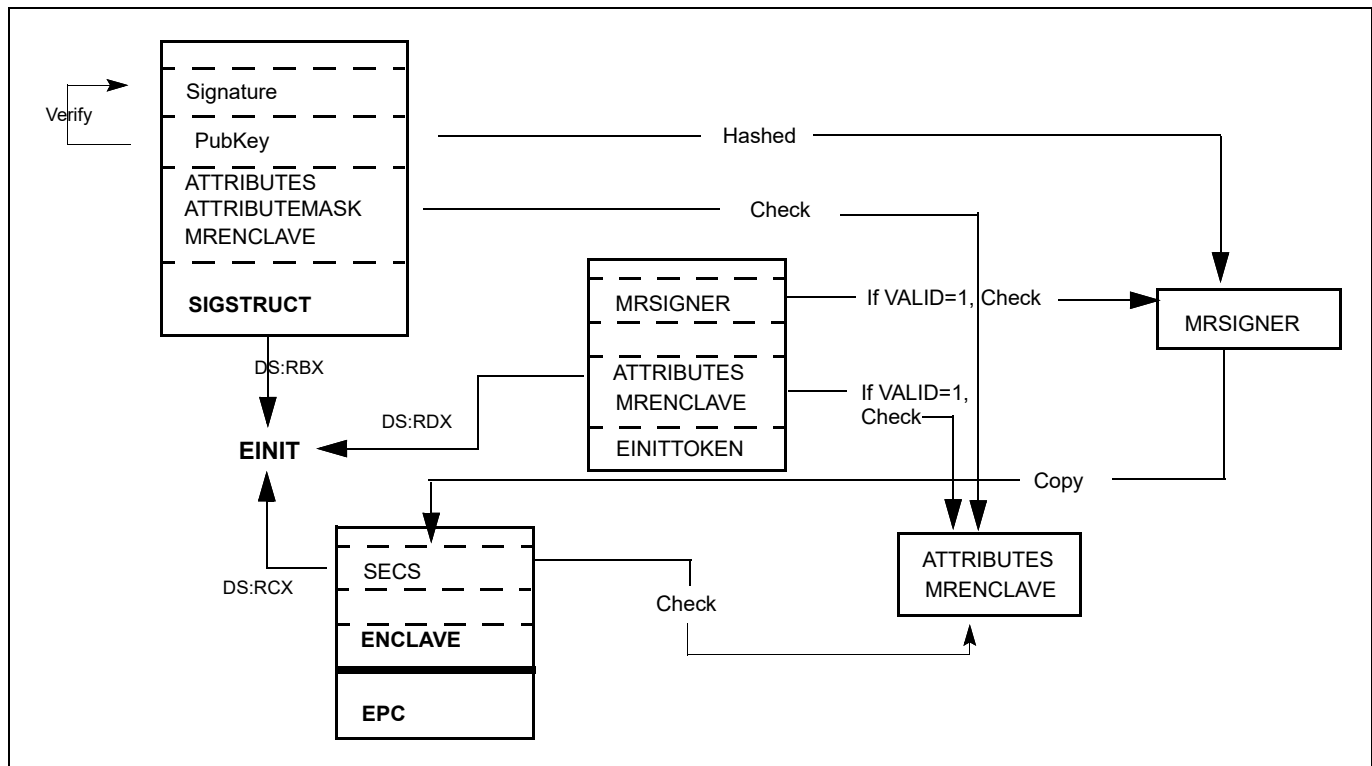


Figure 41-1. Relationships Between SECS, SIGSTRUCT, and EINITTOKEN

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 41-1:

1. Validates that SIGSTRUCT is signed using the enclosed public key.
2. Checks that the completed computation of SECS.MRENCLAVE equals SIGSTRUCT.HASHENCLAVE.
3. Checks that no controlled ATTRIBUTES bits are set in SIGSTRUCT.ATTRIBUTES unless the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
4. Checks that the result of bitwise and-ing SIGSTRUCT.ATTRIBUTEMASK with SIGSTRUCT.ATTRIBUTES equals the result of bitwise and-ing SIGSTRUCT.ATTRIBUTEMASK with SECS.ATTRIBUTES.
5. If EINITTOKEN.VALID is 0, checks that the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
6. If EINITTOKEN.VALID is 1, checks the validity of EINITTOKEN.
7. If EINITTOKEN.VALID is 1, checks that EINITTOKEN.MRENCLAVE equals SECS.MRENCLAVE.
8. If EINITTOKEN.VALID is 1 and EINITTOKEN.ATTRIBUTES.DEBUG is 1, SECS.ATTRIBUTES.DEBUG must be 1.
9. Commits SECS.MRENCLAVE, and sets SECS.MRSIGNER, SECS.ISVSVN, and SECS.ISVPRODID based on SIGSTRUCT.
10. 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 include 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 41-24. EINIT Return Value in RAX

Error Code (see Table 41-3)	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 41-25. Base Concurrency Restrictions of EINIT

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EINIT	SECS [DS:RCX]	Shared	#GP

Table 41-26. Additional Concurrency Restrictions of ENIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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.12H.01H:EAX[6] = 1)
```

```
    IF ( DS:RCX.CET_ATTRIBUTES & TMP_SIG.CET_ATTRIBUTES_MASK ≠ TMP_SIG.CET_ATTRIBUTES &  
        TMP_SIG.CET_ATTRIBUTES_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.SGXOWNERPOUCH := CR_SGXOWNERPOUCH;
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.12H.01H: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—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.

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 table below provides additional information on the memory parameter of ELDB/ELDU leaf functions.

ELDB/ELDU 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 per- mitted by Enclave

The error codes are:

Table 41-27. ELDB/ELDU Return Value in RAX

Error Code (see Table 41-3)	Description
No Error	ELDB/ELDU successful.
SGX_MAC_COMPARE_FAIL	If the MAC check fails.

Concurrency Restrictions

Table 41-28. Base Concurrency Restrictions of ELDB/ELDU

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
ELDB/ELDU	Target [DS:RCX]	Exclusive	#GP
	VA [DS:RDX]	Shared	#GP
	SECS [DS:RBX]PAGEINFO.SECs	Shared	#GP

Table 41-29. Additional Concurrency Restrictions of ELDB/ELDU

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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	

Operation**Temp Variables in ELDB/ELDU Operational Flow**

Name	Type	Size (Bits)	Description
TMP_SRCPE	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_SRCPE := 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_SRCPE is not 4KByte aligned))

THEN #GP(0); FI;

(* Check concurrency of EPC by other Intel SGX instructions *)

IF (other instructions accessing EPC)

```

THEN
    #GP(0); FI;

(* Check concurrency of EPC and VASLOT by other Intel SGX instructions *)
IF (Other instructions modifying VA slot)
THEN
    #GP(0); 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 & ~OFFFH).VALID = 0) or (EPCM(DS:RDX & ~OFFFH).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.12H.01H:EAX[6] = 1) or
    (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_REST and CPUID.12H.01H: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 (Another instruction is currently modifying the SECS)
    THEN
        #GP(0); FI;
    TMP_HEADER.EID := DS:TMP_SECS.EID;
ELSE
    (* TMP_HEADER.SECINFO.FLAGS.PT is PT_SECS or PT_VA which do not have a parent SECS, and hence no EID binding *)
    TMP_HEADER.EID := 0;
    IF (DS:TMP_SECS ≠ 0)
    THEN #GP(0) FI;
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;

(* Clear VA Slot *)
DS:RDX := 0

(* 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) 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;

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.

#PF(error code)	If the instruction fails to verify MAC.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
	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 41-30. EMODPR Return Value in RAX

Error Code (see Table 41-3)	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 41-31. Base Concurrency Restrictions of EMODPR

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EMODPR	Target [DS:RCX]	Shared	#GP

Table 41-32. Additional Concurrency Restrictions of EMODPR

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODPR	Target [DS:RCX]	Exclusive	SGX_EP-C_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 41-33. EMODT Return Value in RAX

Error Code (see Table 41-3)	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 41-34. Base Concurrency Restrictions of EMODT

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EMODT	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT

Table 41-35. Additional Concurrency Restrictions of EMODT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODT	Target [DS:RCX]	Exclusive	SGX_EP-C_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 41-36. Base Concurrency Restrictions of EPA

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EPA	VA [DS:RCX]	Exclusive	#GP

Table 41-37. Additional Concurrency Restrictions of EPA

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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
#GP(0); 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.

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 41-38. EREMOVE Return Value in RAX

Error Code (see Table 41-3)	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 41-39. Base Concurrency Restrictions of EREMOVE

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EREMOVE	Target [DS:RCX]	Exclusive	#GP

Table 41-40. Additional Concurrency Restrictions of EREMOVE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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
#GP(0); 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;
    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.

ETRAK—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0CH ENCLS[ETRAK]	IR	V/V	SGX1	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	ETRAK (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 ETRAK leaf function.

ETRAK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 41-41. ETRAK Return Value in RAX

Error Code (see Table 41-3)	Description
No Error	ETRAK successful.
SGX_PREV_TRK_INCMPL	All processors did not complete the previous shoot-down sequence.

Concurrency Restrictions

Table 41-42. Base Concurrency Restrictions of ETRAK

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
ETRAK	SECS [DS:RCX]	Shared	#GP

Table 41-43. Additional Concurrency Restrictions of ETRAK

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRAK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ETRAK	SECS [DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EP-C_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
#GP(0); 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
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.

EUPDATESVN—Update CR_CPUSVN

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 18H ENCLS[EUPDATESVN]	None	V/V	Bit 10	This leaf function updates the CR_CPUSVN if microcode has been updated and EPC is ready.

Description

If EPC is ready for SVN update, this leaf function updates CR_CPUSVN to the currently loaded microcode update SVN and generates new cryptographic assets. The EPC is ready when no page in the EPC is valid. EREMOVE should be used to mark all pages as unused.

It is the responsibility of system software to ensure that no other thread is executing or attempts to execute any ENCLS leaf while executing EUPDATESVN. Concurrency violations between EUPDATESVN and some ENCLS leaves may cause the ENCLS leaf to generate #GP(0) in ways unexpected to legacy software. System software should also prevent unnecessary software from having access to EUPDATESVN. For example, enable ENCLS exiting should be used to prevent VMs that are not part of the management system software from using EUPDATESVN.

The EUPDATESVN leaf function fails if an ENCLS instruction is in progress on any thread, the EPC is not ready for an update, or there is insufficient entropy in the random number generator. The ZF flag will be set to indicate an error and a code returned in RAX. If EUPDATESVN was successful but CR_CPUSVN was already up to date, the CF flag will be set and RAX will indicate that no update occurred.

If insufficient entropy causes a failure, software should repeat the instruction.

The error codes are:

Table 41-44. EUPDATESVN Return Value in RAX

Error Code (see Table 41-3)	Value	Description
No Error	0	EUPDATESVN successful.
SGX_EPC_PAGE_CONFLICT	7	An instruction concurrency rule was violated.
SGX_INSUFFICIENT_ENTROPY	29	RNG contains insufficient entropy.
SGX_EPC_NOT_READY	30	EPC is not ready for SVN update.
SGX_NO_UPDATE	31	EUPDATESVN was successful, but CR_CPUSVN was not updated because the current SVN is older than CR_CPUSVN.

Concurrency Restrictions

Table 41-45. Base Concurrency Restrictions of EUPDATESVN

Leaf	Base Concurrency Restrictions	
	Access	On Conflict
EUPDATESVN	Exclusive	SGX_EPC_PAGE_CONFLICT

Table 41-46. Additional Concurrency Restrictions of EUPDATESVN

Leaf	Additional Concurrency Restrictions	
	vs. EADD, EAUG, ECREATE, ELDB, EPA, EREMOVE, EWB	
	Access	On Conflict
EUPDATESVN	Exclusive	SGX_EPC_PAGE_CONFLICT

Operation

Temp Variables in EUPDATESVN Operational Flow

Name	Type	Size (Bytes)	Description
TMP_CPUSVN	CPUSVN	16	Temporary copy of CR_CPUSVN prior to update.
TMP_KEY	Key	16	Temporary copy of new paging key.

```
RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
RAX := 0;

IF (Other logical processor is executing an SGX instruction that accesses EPC) THEN
    RFLAGS.ZF := 1
    RAX := SGX_EPC_PAGE_CONFLICT;
    GOTO ERROR_EXIT;
FI

(* Verify EPC is ready *)
IF (the EPC contains any valid pages) THEN
    RFLAGS.ZF := 1;
    RAX := SGX_EPC_NOT_READY;
    GOTO ERROR_EXIT;
FI

(* Refresh paging key *)
TMP_KEY = (* Generate a 128-bit cryptographically random number *)
IF (insufficient entropy available) THEN
    RFLAGS.ZF := 1;
    RAX := SGX_INSUFFICIENT_ENTROPY;
    GOTO ERROR_EXIT;
FI

(* Commit *)
TMP_CPUSVN := CR_CPUSVN;

(* Update CR_CPUSVN to reflect current microcode update SVN *)

(* Determine if info status is needed *)
IF (TMP_CPUSVN = CR_CPUSVN) THEN
    RFLAGS.CF := 1;
    RAX := SGX_NO_UPDATE;
ELSE
    CR_BASE_PK := TMP_KEY[127:0];
FI
ERROR_EXIT:
```

Flags Affected

ZF is set if an error occurs; otherwise, cleared.
CF is set when the instruction is completed successfully and no SVN update was needed.
PF, AF, OF, and SF are cleared.

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 41-47. EWB Return Value in RAX

Error Code (see Table 41-3)	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 41-48. Base Concurrency Restrictions of EWB

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EWB	Source [DS:RCX]	Exclusive	#GP
	VA [DS:RDX]	Shared	#GP

Table 41-49. Additional Concurrency Restrictions of EWB

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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
 #GP(0); 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 & ~OFFFH).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;
    FI;

```

```

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 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.

41.4 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 41-50. EACCEPT Return Value in RAX

Error Code (see Table 41-3)	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 41-51. Base Concurrency Restrictions of EACCEPT

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EACCEPT	Target [DS:RCX]	Shared	#GP
	SECINFO [DS:RBX]	Concurrent	

Table 41-52. Additional Concurrency Restrictions of EACCEPT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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.12H.01H: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.07H.00H: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 41-53. EACCEPTCOPY Return Value in RAX

Error Code (see Table 41-3)	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 41-54. Base Concurrency Restrictions of EACCEPTCOPY

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EACCEPTCOPY	Target [DS:RCX]	Concurrent	
	Source [DS:RDX]	Concurrent	
	SECINFO [DS:RBX]	Concurrent	

Table 41-55. Additional Concurrency Restrictions of EACCEPTCOPY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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 4KBytes 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.

EDECCSSA—Decrements TCS.CSSA

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 09H ENCLU[EDECCSSA]	IR	V/V	EDECCSSA	This leaf function decrements TCS.CSSA.

Instruction Operand Encoding

Op/En	EAX
IR	EDECCSSA (In)

Description

This leaf function changes the current SSA frame by decrementing TCS.CSSA for the current enclave thread. If the enclave has enabled CET shadow stacks or indirect branch tracking, then EDECCSSA also changes the current CET state save frame. This instruction leaf can only be executed inside an enclave.

EDECCSSA Memory Parameter Semantics

TCS
Read/Write access by Enclave

The instruction faults if any of the following:

EDECCSSA Faulting Conditions

TCS.CSSA is 0.	TCS is not valid or available or locked.
The SSA frame is not valid or in use.	

Concurrency Restrictions

Table 41-56. Base Concurrency Restrictions of EDECCSSA

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EDECCSSA	TCS [CR_TCS_PA]	Shared	#GP

Table 41-57. Additional Concurrency Restrictions of EDECCSSA

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDECCSSA	TCS [CR_TCS_PA]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDECCSSA Operational Flow

Name	Type	Size (bits)	Description
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 target frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the target SSA frame.
TMP_XSAVE_PAGE_PA_n	Physical Address	32/64	Physical address of the nth page within the target SSA frame.
TMP_CET_SAVE_AREA	Effective Address	32/64	Address of the current CET save area.
TMP_CET_SAVE_PAGE	Effective Address	32/64	Address of the current CET save area page.

(* Check concurrency of TCS operation *)

IF (Other Intel SGX instructions are operating on TCS)
 THEN #GP(0); FI;

IF (CR_TCS_PA.CSSA = 0)
 THEN #GP(0); FI;

(* Compute linear address of SSA frame *)

TMP_SSA := CR_TCS_PA.OSSA + CR_ACTIVE_SECS.BASEADDR + 4096 * CR_ACTIVE_SECS.SSAFRAMESIZE * (CR_TCS_PA.CSSA - 1);

TMP_XSIZE := compute_XSAVE_frame_size(CR_ACTIVE_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(CR_TCS_PA).ENCLAVESECS) or

 (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0))

 THEN #PF(DS:TMP_SSA_PAGE); FI;

 TMP_XSAVE_PAGE_PA_n := Physical_Address(DS:TMP_SSA_PAGE);

ENDFOR

(* Compute address of GPR area*)

TMP_GPR := TMP_SSA + 4096 * CR_ACTIVE_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(CR_TCS_PA).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 + (sizeof(GPRSGX_AREA) - 1) is not in DS segment)
            THEN #GP(0); FI;
FI;

IF (CPUID.12H.01H:EAX[6] = 1)
    THEN
        IF ((CR_ACTIVE_SECS.CET_ATTRIBUTES.SH_STK_EN == 1) OR (CR_ACTIVE_SECS.CET_ATTRIBUTES.ENDBR_EN == 1))
            THEN
                (* Compute linear address of what will become new CET state save area and cache its PA *)
                TMP_CET_SAVE_AREA := CR_TCS_PA.OCETSSA + CR_ACTIVE_SECS.BASEADDR + (CR_TCS_PA.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(CR_TCS_PA).ENCLAVESECS))
                    THEN #PF(DS:TMP_CET_SAVE_PAGE); FI;
            FI;
        FI;

    (* At this point, the instruction is guaranteed to complete *)
    CR_TCS_PA.CSSA := CR_TCS_PA.CSSA - 1;

    CR_GPR_PA := Physical_Address(DS:TMP_GPR);

    FOR EACH TMP_XSAVE_PAGE_n
        CR_XSAVE_PAGE_n := TMP_XSAVE_PAGE_PA_n;

```

ENDFOR

IF (CPUID.12H.01H:EAX[6] = 1)

THEN

IF ((TMP_SECS.CET_ATTRIBUTES.SH_STK_EN == 1) OR

(TMP_SECS.CET_ATTRIBUTES.ENDBR_EN == 1))

THEN

CR_CET_SAVE_AREA_PA := Physical_Address(DS:TMP_CET_SAVE_AREA);

FI;

FI;

Flags Affected

None

Protected Mode Exceptions

#GP(0) If executed outside an enclave.

If CR_TCS_PA.CSSA = 0.

#PF(error code) If a page fault occurs in accessing memory.

If one or more pages of the target SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

If CET is enabled for the enclave and the target CET SSA frame is not readable/writable, or does not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0) If executed outside an enclave.

If CR_TCS_PA.CSSA = 0.

#PF(error code) If a page fault occurs in accessing memory.

If one or more pages of the target SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

If CET is enabled for the enclave and the target CET SSA frame is not readable/writable, or does not resolve to a valid PT_REG EPC page.

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.
If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.	

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 43.1.2):
 - 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 43.2.5).
 - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 43.2.2).

- 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 43.2.3):
 - 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 41-58. Base Concurrency Restrictions of EENTER

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EENTER	TCS [DS:RBX]	Shared	#GP

Table 41-59. Additional Concurrency Restrictions of EENTER

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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 GS 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[bits 11:9] != 001B))
 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 are 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 & FFFFFFFFFFFFFFFCH) ≠ 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 & XCRO) ≠ TMP_SECS.ATTRIBUTES.XFRM) THEN #GP(0); FI;
FI;

IF ((DS:RBX).CSSA.FLAGS.DBGOPTIN = 0) and (DS:RBX).CSSA.FLAGS.AEXNOTIFY ≠ TMP_SECS.ATTRIBUTES.AEXNOTIFY))
    THEN #GP(0); 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.12H.01H: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 XCRO and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCRO := XCRO;
    XCRO := 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[bit 9];
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS[bit 21];
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS[bit 9];
GS.S := 1;

```

```

GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS[bit 21];
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.07H.00H:EDX.CET_IBT = 1) OR (CPUID.07H.00H: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.07H.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 = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0. If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#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 = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0. If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#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 41-60. Base Concurrency Restrictions of EEXIT

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EEXIT		Concurrent	

Table 41-61. Additional Concurrency Restrictions of EEXIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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_monitor_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.12H.01H:EAX[6] = 1)
    THEN
        (* Record PREVSSP *)
        IF (IA32_U_CET.SH_STK_EN == 1)
            THEN CR_TCS_PA.PREVSSP = SSP; FI;
FI;

IF ((CPUID.07H.00H:EDX.CET_IBT = 1) OR (CPUID.07H.00H: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.07H.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.

EGETKEY 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 41-62.
- 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 CR_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 38.18.1) 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 41-62) 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 41-62 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 41-62. Key Derivation

	Key Name	Attributes	Owner Epoch	CPU SVN	ISV SVN	ISV PRODID	ISVEXT PRODID	ISVFAMILYID	MRENCLAVE	MRSIGNER	CONFIG ID	CONFIGSVN	RAND
Source	Key Dependent Constant	Y := SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES;	CR_SGX_OWNER EPOCH	Y := CPUSVN Register;	R := Req.ISV SVN;	SECS.ISVID	SECS.ISVEXTPR ODID	SECS.ISVFAMILYID	SECS.MRENCLAVE	SECS.MRSIGNER	SECS.CONFIGID	SECS.CONFIGSVN	Req. KEYID
		R := AttribMask & SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES;		R := Req.CPU SVN;									
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 hardcoded 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 41-63. EGETKEY Return Value in RAX

Error Code (see Table 41-3)	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 41-64. Base Concurrency Restrictions of EGETKEY

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent	

Table 41-65. Additional Concurrency Restrictions of EGETKEY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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.12H.01H: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.ATTRIBUTESMASK;
    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.12H.01H: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.12H.01H: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;
    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.12H.01H: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.12H.01H: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)
        THEN TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
    FI;

    (* Include enclave Config ID? *)
    TMP_CONFIGID := 0;
    TMP_CONFIGSVN := 0;
    IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
        THEN 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.ATTRIBUTESMASK;
    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.12H.01H: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 41-66. Base Concurrency Restrictions of EMODPE

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EMODPE	Target [DS:RCX]	Concurrent	
	SECINFO [DS:RBX]	Concurrent	

Table 41-67. Additional Concurrency Restrictions of EMODPE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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 41-68. Base Concurrency Restrictions of EREPORT

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EREPORT	TARGETINFO [DS:RBX]	Concurrent	
	REPORTDATA [DS:RCX]	Concurrent	
	OUTPUTDATA [DS:RDX]	Concurrent	

Table 41-69. Additional Concurrency Restrictions of EREPORT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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.12H.01H: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.12H.01H: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 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 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 XCRO.
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.	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.

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, XCRO 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 43.1.2):
 - 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 43.2.5).
 - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 43.2.3).
- 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 43.2.3):
 - 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 41-70. Base Concurrency Restrictions of ERESUME

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
ERESUME	TCS [DS:RBX]	Shared	#GP

Table 41-71. Additional Concurrency Restrictions of ERESUME

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		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 GS 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_NOTIFY	Boolean	1	When set to 1, deliver an AEX notification.

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[bits 11:9] != 001B))

THEN #GP(0); FI;

(* Check that CS, SS, DS, ES.base is 0 *)

IF (TMP_MODE64 = 0)

THEN

IF(CS.base $\neq$ 0 or DS.base $\neq$ 0) #GP(0); FI;

IF(ES usable and ES.base $\neq$ 0) #GP(0); FI;

IF(SS usable and SS.base $\neq$ 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 are 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 $\neq$ DS:RBX) or (EPCM(DS:RBX).PT $\neq$ 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 & FFFFFFFFFFFFFFFCH) $\neq$ 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 & XCRO) ≠ TMP_SECS.ATTRIBUTES.XFRM) THEN #GP(0); FI;
FI;

IF ( (DS:RBX).CSSA.FLAGS.DBGOPTIN = 0) and (DS:RBX).CSSA.FLAGS.AEXNOTIFY ≠ TMP_SECS.ATTRIBUTES.AEXNOTIFY))
    THEN #GP(0); 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);

IF ((DS:RBX).FLAGS.AEXNOTIFY = 1) and (DS:TMP_GPR.AEXNOTIFY[0] = 1))
    THEN
        TMP_NOTIFY := 1;
    ELSE
        TMP_NOTIFY := 0;
FI;

IF (TMP_NOTIFY = 1)
    THEN
        (* Make sure the SSA contains at least one more frame *)
        IF ((DS:RBX).CSSA ≥ (DS:RBX).NSSA)
            THEN #GP(0); FI;

        TMP_SSA := TMP_SSA + 4096 * TMP_SECS.SSAFRAMESIZE;
        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);

TMP_TARGET := (DS:RBX).OENTRY + TMP_SECS.BASEADDR;

ELSE

TMP_TARGET := (DS:TMP_GPR).RIP;

FI;

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
    IF (TMP_NOTIFY = 1)
        THEN
            TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
            TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        ELSE
            TMP_FSBASE := DS:TMP_GPR.FSBASE;
            TMP_GSBASE := DS:TMP_GPR.GSBASE;
        FI;
    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.12H.01H: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;
        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 *)
        IF (TMP_NOTIFY = 1)
            THEN
                TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA) * 16;
            ELSE
                TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA - 1) * 16;
            FI;
        FI;

```

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_NOTIFY = 1)

THEN

IF TMP_IA32_U_CET.SH_STK_EN = 1

THEN TMP_SSP = TCS.PREVSSP; FI;

ELSE

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 & 0xFFFFFFFFF0000000) ≠ 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;

FI;

IF (TMP_NOTIFY = 0)

THEN

(* 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;

FI;

CR_ENCLAVE_MODE := 1;

CR_ACTIVE_SECS := TMP_SECS;

CR_ELRange := (TMP_SECS.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 (TMP_NOTIFY = 1)
  THEN
    (* If XSAVE is enabled, save XCRO and replace it with SECS.ATTRIBUTES.XFRM*)
    IF (CR4.OSXSAVE = 1)
      THEN
        CR_SAVE_XCRO := XCRO;
        XCRO := TMP_SECS.ATTRIBUTES.XFRM;
      FI;
    FI;

RIP := TMP_TARGET;

IF (TMP_NOTIFY = 1)
  THEN
    RCX := RIP;
    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;
  ELSE
    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 XCRO and replace it with SECS.ATTRIBUTES.XFRM*)
    IF (CR4.OSXSAVE = 1)
    THEN
        CR_SAVE_XCRO := XCRO;
        XCRO := TMP_SECS.ATTRIBUTES.XFRM;
    FI;

    (* Pop the SSA stack*)
    (DS:RBX).CSSA := (DS:RBX).CSSA - 1;
FI;

(* Do the FS/GS swap *)
FS.base := TMP_FSBASE;
FS.limit := DS:RBX.FSLIMIT;
FS.type := 0001b;
FS.W := DS[bit 9];
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS[bit 21];
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS[bit 9];
GS.S := 1;
GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS[bit 21];
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

```



```

    IF (TMP_NOTIFY = 1)
        THEN
            IF RFLAGS.TF = 1
                THEN pend a single-step #DB at the end of ERESUME; FI;
            IF the "monitor trap flag" VM-execution control is set
                THEN pend an MTF VM exit at the end of ERESUME; FI;
        ELSE
            Clear all pending debug exceptions;
            Clear pending MTF VM exits;
        FI;
    FI;

    IF ((CPUID.07H.00H:EDX.CET_IBT = 1) OR (CPUID.07H.00H: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.07H.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 = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0. If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#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 = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0. If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#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.

18. Updates to Chapter 2, Volume 4

Change bars and violet text show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers*.

Changes to this chapter:

- In IA32_PQR_ASSOC MSR (C8FH) of Table 2-2, the bits containing the CLOS field was updated to M:32 and formulas for CLOS_MAX, CLOS_WIDTH, and M were added.
- Table 2-56, "Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors," and Table 2-57, "Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors," were updated to include MSR_FEATURE_CONFIG register at address 13CH, which includes the AESNI_LOCK bit. Also in Table 2-56 and Table 2-57, bit 0 of the MSR_FEATURE_ENABLES register (address 140H) was corrected to the CPUID_GP_ON_CPL_GT_0 bit.

CHAPTER 2

MODEL-SPECIFIC REGISTERS (MSRS)

This chapter lists MSRs across Intel processor families. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions. The scope of an MSR defines the set of processors that access the same MSR with RDMSR and WRMSR. Thread-scope MSRs are unique to every logical processor. Core-scope MSRs are shared by the threads in the same core; similarly for module-scope, die-scope, and package-scope.

When a processor package contains a single die, die-scope and package-scope are synonymous. When a package contains multiple die, they are distinct.

NOTE

For information on hierarchical level types supported, refer to the CPUID.1FH definition for the actual level type numbers: "V2 Extended Topology Enumeration Leaf" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. Also see Section 11.9.1, "Hierarchical Mapping of Shared Resources," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

Register addresses are given in both hexadecimal and decimal. The register name is the mnemonic register name and the bit description describes individual bits in registers.

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H to query the combination of DisplayFamily and DisplayModel to determine model-specific availability of MSRs (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). Table 2-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_BDH	Intel® Series 2 Core™ Ultra processors supporting Lunar Lake performance hybrid architecture
06_ADH, 06_AEH	Intel® Xeon® 6 P-core processors based on Granite Rapids microarchitecture
06_AFH	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture
06_AAH	Intel® Core™ Ultra 7 processors supporting Meteor Lake performance hybrid architecture
06_CFH	5th generation Intel® Xeon® Scalable Processor Family based on Emerald Rapids microarchitecture
06_8FH	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture
06_BAH, 06_B7H, 06_BFH	13th generation Intel® Core™ processors supporting Raptor Lake performance hybrid architecture
06_97H, 06_9AH	12th generation Intel® Core™ processors supporting Alder Lake performance hybrid architecture
06_8CH, 06_8DH	11th generation Intel® Core™ processors based on Tiger Lake microarchitecture
06_A7H	11th generation Intel® Core™ processors based on Rocket Lake microarchitecture
06_7EH	10th generation Intel® Core™ processors based on Ice Lake microarchitecture
06_A5H, 06_A6H	10th generation Intel® Core™ processors based on Comet Lake microarchitecture
06_66H	Intel® Core™ processors based on Cannon Lake microarchitecture
06_8EH, 06_9EH	7th generation Intel® Core™ processors based on Kaby Lake microarchitecture, 8th and 9th generation Intel® Core™ processors based on Coffee Lake microarchitecture, Intel® Xeon® E processors based on Coffee Lake microarchitecture
06_6AH, 06_6CH	3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_55H	Intel® Xeon® Scalable Processor Family based on Skylake microarchitecture, 2nd generation Intel® Xeon® Scalable Processor Family based on Cascade Lake product, and 3rd generation Intel® Xeon® Scalable Processor Family based on Cooper Lake product
06_4EH, 06_5EH	6th generation Intel Core processors and Intel Xeon processor E3-1500m v5 product family and E3-1200 v5 product family based on Skylake microarchitecture
06_85H	Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture
06_57H	Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series based on Knights Landing microarchitecture
06_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture
06_4FH	Intel Xeon processor E5 v4 Family based on Broadwell microarchitecture, Intel Xeon processor E7 v4 Family, Intel Core i7-69xx Processor Extreme Edition
06_47H	5th generation Intel Core processors, Intel Xeon processor E3-1200 v4 product family based on Broadwell microarchitecture
06_3DH	Intel Core M-5xxx Processor, 5th generation Intel Core processors based on Broadwell microarchitecture
06_3FH	Intel Xeon processor E5-4600/2600/1600 v3 product families, Intel Xeon processor E7 v3 product families based on Haswell-E microarchitecture, Intel Core i7-59xx Processor Extreme Edition
06_3CH, 06_45H, 06_46H	4th Generation Intel Core processor and Intel Xeon processor E3-1200 v3 product family based on Haswell microarchitecture
06_3EH	Intel Xeon processor E7-8800/4800/2800 v2 product families based on Ivy Bridge-E microarchitecture
06_3EH	Intel Xeon processor E5-2600/1600 v2 product families and Intel Xeon processor E5-2400 v2 product family based on Ivy Bridge-E microarchitecture, Intel Core i7-49xx Processor Extreme Edition
06_3AH	3rd Generation Intel Core Processor and Intel Xeon processor E3-1200 v2 product family based on Ivy Bridge microarchitecture
06_2DH	Intel Xeon processor E5 Family based on Sandy Bridge microarchitecture, Intel Core i7-39xx Processor Extreme Edition
06_2FH	Intel Xeon Processor E7 Family
06_2AH	Intel Xeon processor E3-1200 product family; 2nd Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series
06_25H, 06_2CH	Intel Xeon processors 3600, 5600 series, Intel Core i7, i5, and i3 Processors
06_1EH, 06_1FH	Intel Core i7 and i5 Processors
06_1AH	Intel Core i7 Processor, Intel Xeon processor 3400, 3500, 5500 series
06_1DH	Intel Xeon processor MP 7400 series
06_17H	Intel Xeon processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series
06_0FH	Intel Xeon processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_86H, 06_96H, 06_9CH	Intel Atom® processors, Intel® Celeron® processors, Intel® Pentium® processors, and Intel® Pentium® Silver processors based on Tremont Microarchitecture
06_7AH	Intel Atom processors based on Goldmont Plus microarchitecture
06_5FH	Intel Atom processors based on Goldmont microarchitecture (Denverton)
06_5CH	Intel Atom processors based on Goldmont microarchitecture

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_4CH	Intel Atom processor X7-Z8000 and X5-Z8000 series based on Airmont microarchitecture
06_5DH	Intel Atom processor X3-C3000 based on Silvermont microarchitecture
06_5AH	Intel Atom processor Z3500 series
06_4AH	Intel Atom processor Z3400 series
06_37H	Intel Atom processor E3000 series, Z3600 series, Z3700 series
06_4DH	Intel Atom processor C2000 series
06_36H	Intel Atom processor S1000 Series
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	Intel Atom processor family, Intel Atom processor D2000, N2000, E2000, Z2000, C1000 series
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon processor, Intel Xeon processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon processor, Intel Pentium III processor
06_03H, 06_05H	Intel Pentium II Xeon processor, Intel Pentium II processor
06_01H	Intel Pentium Pro processor
05_01H, 05_02H, 05_04H	Intel Pentium processor, Intel Pentium processor with MMX Technology
06_BEH	Intel® Processor and Intel® Core™ i3 and Intel® Core™ 3 N-Series Processors and Intel Atom® x7000 Processor Series based on Gracemont microarchitecture

The Intel® Quark™ SoC X1000 processor can be identified by the signature of DisplayFamily_DisplayModel = 05_09H and SteppingID = 0

2.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32_”. Table 2-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table 2-2 and certain bit fields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a model-specific MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table 2-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of “DF_DM” (see Table 2-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as “MAXPHYADDR” in Table 2-2. MAXPHYADDR is derived from the value enumerated in CPUID.80000008H:EAX[7:0] (this width is at most 52). However, if IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3); the value is not reduced in SEAM.¹

MSR address range between 40000000H - 4000FFFFH is marked as a specially reserved range. All existing and future processors will not implement any features using any MSR in this range.

Table 2-2. IA-32 Architectural MSRs

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 0H, 0		IA32_P5_MC_ADDR (P5_MC_ADDR)	
See Section 2.23, "MSRs in Pentium Processors."			Pentium Processor (05_01H)
Register Address: 1H, 1		IA32_P5_MC_TYPE (P5_MC_TYPE)	
See Section 2.23, "MSRs in Pentium Processors."			DF_DM = 05_01H
Register Address: 6H, 6		IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination."			0F_03H
Register Address: 10H, 16		IA32_TIME_STAMP_COUNTER (TSC)	
See Section 20.17, "Time-Stamp Counter."			05_01H
Register Address: 17H, 23		IA32_PLATFORM_ID (MSR_PLATFORM_ID)	
Platform ID (R/O) The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.			06_01H
49:0	Reserved.		
52:50	Platform ID (R/O) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7		
63:53	Reserved.		
Register Address: 1BH, 27		IA32_APIC_BASE (APIC_BASE)	
This register holds the APIC base address, permitting the relocation of the APIC memory map. See Section 13.4.4, "Local APIC Status and Location," and Section 13.4.5, "Relocating the Local APIC Registers."			06_01H
7:0	Reserved.		
8	BSP Flag (R/W)		
9	Reserved.		
10	Enable x2APIC mode.		06_1AH
11	APIC Global Enable (R/W)		

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MAXAPICADDR -1:12	APIC Base (R/W)	MAXAPICADDR is normally CPUID.80000008H:EAX[7:0] for processors that support CPUID.80000008H and 36 otherwise. If IA32_TME_ACTIVATE[39:36] > 0, MAXAPICADDR is reduced by IA32_TME_ACTIVATE[35:32].
63: MAXAPICADDR	Reserved.	
Register Address: 2FH, 47		IA32_BARRIER
IA32_BARRIER (R/O) The IA32_BARRIER MSR ensures ordered execution by acting like LFENCE, controlling the sequencing of subsequent MSR reads after prior MSR reads and instructions.		CPUID.07H.01H:EAX[27] = 1
31:0	DATA Reserved. Always 0.	
63:32	Reserved.	
Register Address: 3AH, 58		IA32_FEATURE_CONTROL
Control Features in Intel 64 Processor (R/W)		If any one enumeration condition for defined bit field holds.
0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written; writes to this bit will result in GP(0). Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support for Intel Virtualization Technology and prior to transferring control to an option ROM or the OS. Hence, once the Lock bit is set, the entire IA32_FEATURE_CONTROL contents are preserved across RESET when PWRGOOD is not deasserted.	If any one enumeration condition for defined bit field position greater than bit 0 holds.
1	Enable VMX inside SMX operation (R/WL) This bit enables a system executive to use VMX in conjunction with SMX to support Intel® Trusted Execution Technology. BIOS must set this bit only when CPUID.01H:ECX[6:5] returns 11b (SMX and VMX respectively).	If CPUID.01H:ECX[5] = 1 && CPUID.01H:ECX[6] = 1
2	Enable VMX outside SMX operation (R/WL) This bit enables VMX for a system executive that does not require SMX. BIOS must set this bit only when CPUID.01H:ECX.VMX[5] is set.	If CPUID.01H:ECX[5] = 1
7:3	Reserved.	
14:8	SENTER Local Function Enables (R/WL) When set, each bit in the field represents an enable control for a corresponding SENTER function. This field is supported only if CPUID.01H:ECX[6] is set.	If CPUID.01H:ECX[6] = 1
15	SENTER Global Enable (R/WL) This bit must be set to enable SENTER leaf functions. This bit is supported only if CPUID.01H:ECX[6] is set.	If CPUID.01H:ECX[6] = 1
16	Reserved.	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime re-configuration of SGX Launch Control via the IA32_SGXLEPUBKEYHASHn MSR.	If CPUID.07H.00H:ECX[30] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
18	SGX Global Enable (R/WL) This bit must be set to enable SGX leaf functions.	If CPUID.07H.00H:EBX[2] = 1
19	Reserved.	
20	LMCE On (R/WL) When set, system software can program the MSRs associated with LMCE to configure delivery of some machine check exceptions to a single logical processor.	If IA32_MCG_CAP[27] = 1
63:21	Reserved.	
Register Address: 3BH, 59		IA32_TSC_ADJUST
Per Logical Processor TSC Adjust (R/Write to clear)		If CPUID.07H.00H:EBX[1] = 1
63:0	THREAD_ADJUST Local offset value of the IA32_TSC for a logical processor. Reset value is zero. A write to IA32_TSC will modify the local offset in IA32_TSC_ADJUST and the content of IA32_TSC, but does not affect the internal invariant TSC hardware.	
Register Address: 48H, 72		IA32_SPEC_CTRL
Speculation Control (R/W) The MSR bits are defined as logical processor scope. On some core implementations, the bits may impact sibling logical processors on the same core. This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.		If any one of the enumeration conditions for defined bit field positions holds.
0	Indirect Branch Restricted Speculation (IBRS). Restricts speculation of indirect branch.	If CPUID.07H.00H:EDX[26] = 1
1	Single Thread Indirect Branch Predictors (STIBP). Prevents indirect branch predictions on all logical processors on the core from being controlled by any sibling logical processor in the same core.	If CPUID.07H.00H:EDX[27] = 1
2	Speculative Store Bypass Disable (SSBD) delays speculative execution of a load until the addresses for all older stores are known.	If CPUID.07H.00H:EDX[31] = 1
3	IPRED_DIS_U If 1, enables IPRED_DIS control for CPL3.	If CPUID.07H.02H:EDX[1] = 1
4	IPRED_DIS_S If 1, enables IPRED_DIS control for CPL0/1/2.	If CPUID.07H.02H:EDX[1] = 1
5	RRSBA_DIS_U If 1, disables RRSBA behavior for CPL3.	If CPUID.07H.02H:EDX[2] = 1
6	RRSBA_DIS_S If 1, disables RRSBA behavior for CPL0/1/2.	If CPUID.07H.02H:EDX[2] = 1
7	PSFD If 1, disables Fast Store Forwarding Predictor. Note that setting bit 2 (SSBD) also disables this.	If CPUID.07H.02H:EDX[0] = 1
8	DDPD_U If 1, disables the Data Dependent Prefetcher that examines data values in memory while CPL = 3. Note that setting bit 2 (SSBD) also disables this.	If CPUID.07H.02H:EDX[3] = 1
9	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
10	BHI_DIS_S When '1, enables BHI_DIS_S behavior.	If CPUID.07H.02H:EDX[4] = 1
63:11	Reserved.	
Register Address: 49H, 73		IA32_PRED_CMD
Prediction Command (W0) Gives software a way to issue commands that affect the state of predictors.		If any one of the enumeration conditions for defined bit field positions holds.
0	Indirect Branch Prediction Barrier (IBPB)	If CPUID.07H.00H:EDX[26] = 1
63:1	Reserved.	
Register Address: 4EH, 78		IA32_PPIN_CTL
Protected Processor Inventory Number Enable Control (R/W)		If CPUID.07H.01H:EBX[0] = 1 ¹
0	LockOut (R/W0) If 0, indicates that further writes to IA32_PPIN_CTL is allowed. If 1, indicates that further writes to IA32_PPIN_CTL is disallowed. Writing 1 to this bit is only permitted if the Enable_PPIN bit is clear. The Privileged System Software Inventory Agent should read IA32_PPIN_CTL[bit 1] to determine if IA32_PPIN is accessible. The Privileged System Software Inventory Agent is not expected to write to this MSR.	
1	Enable_PPIN (R/W) If 1, indicates that IA32_PPIN is accessible using RDMSR. If 0, indicates that IA32_PPIN is inaccessible using RDMSR. Any attempt to read IA32_PPIN will cause #GP.	
63:2	Reserved.	
Register Address: 4FH, 79		IA32_PPIN
Protected Processor Inventory Number (R/O)		If CPUID.07H.01H:EBX[0] = 1 ¹
63:0	Protected Processor Inventory Number (R/O) A unique value within a given CPUID family/model/stepping signature that a privileged inventory initialization agent can access to identify each physical processor, when access to IA32_PPIN is enabled. Access to IA32_PPIN is permitted only if IA32_PPIN_CTL[bits 1:0] = '10b'.	
Register Address: 79H, 121		IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)
BIOS Update Trigger (W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 12.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.		06_01H
Register Address: 7AH, 122		IA32_FEATURE_ACTIVATION
Feature Activation (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread.		
0	SE Secure Enclaves feature activation.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
1	KL Keylocker feature activation.	
63:2	Reserved.	
Register Address: 7BH, 123		IA32_MCU_ENUMERATION
IA32_MCU_ENUMERATION (R/O) Enumeration of architectural features.		
0	UNIFORM_MCU_AVAIL When set to 1, uniform microcode update is available, and UNIFORM_MCU_SCOPE (bits [10:8]) indicates the scope of writes to IA32_BIOS_UPDT_TRIG. When set to 0, uniform microcode update is not available, and writes to IA32_BIOS_UPDT_TRIG are core scoped.	
1	UNIFORM_MCU_CONFIG_REQD When set to 1, indicates that configuration is required to ensure that all MCU components are updated on WRMSR 79H, and UNIFORM_MCU_CONFIG_COMPLETE (bit 2) should be checked to determine whether the necessary configuration has been completed. When set to 0, indicates that no configuration is required, and UNIFORM_MCU_CONFIG_COMPLETE should be ignored.	
2	UNIFORM_MCU_CONFIG_COMPLETE If UNIFORM_MCU_CONFIG_REQD (bit 1) is 0, then this bit should be ignored. If UNIFORM_MCU_CONFIG_REQD is 1, then this bit indicates whether all necessary configurations have been completed to ensure that all MCU components will be updated on WRMSR 79H.	
3	ARCH_ROLLBACK_SVN_COMMIT When set to 1, indicates support for the MCU deferred SVN architecture, SVN reporting architecture, and MCU rollback architecture.	
4	MCU_STAGING When set to 1, indicates that the microcode update staging capability is supported by the processor. When supported, the use of the MCU staging capability is recommended to reduce the latency of the IA32_BIOS_UPDT_TRIG operation.	
7:5	Reserved for future use.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
15:8	UNIFORM_MCU_SCOPE Indicates the current* uniform microcode update scope: 0x02: Core Scoped 0x03: Module Scoped** 0x04: Tile Scoped** 0x05: Die Scoped** 0x80: Package Scoped 0xC0: Platform Scoped All others: Reserved for future use * The value of this field reflects the state of platform configuration and may change as the configuration changes during the boot process. Once configuration is complete, it is not expected to change during runtime. ** If these domains are enumerated by CPUID.1FH, then this field may also report them as appropriate.	
63:16	Reserved for future use.	
Register Address: 7CH, 124		IA32_MCU_STATUS
MCU Status (R/O) Communicates results from the previous patch loads.		
0	MCU_PARTIAL_UPDATE When set to 1, indicates that the most recent write to IA32_BIOS_UPDT_TRIG resulted in a partial update. This means that microcode update components were only partially updated after some portion of the MCU had already been committed and the Revision ID had been updated.	
1	AUTH_FAIL_ON_MCU_COMPONENT When set to 1, indicates that an authentication failure occurred on some portion of the MCU after another portion of the MCU had already been committed and the Revision ID had already been updated on the most recent write to IA32_BIOS_UPDT_TRIG.	
2	Reserved for future use.	
3	POST_BIOS_MCU When set to 1, indicates that an update was successfully loaded via IA32_BIOS_UPDT_TRIG after bit 0 of MSR_BIOS_DONE (address 151H) was set to 1.	
63:4	Reserved for future use.	
Register Address: 82H, 130		IA32_FZM_RANGE_INDEX
IA32_FZM_RANGE_INDEX (R/W) Index and Domain handle for a valid FZM region. Programmed by software and used by other FRM MSRs FZM Range Index register to R/W Domain Index.		
3:0	REGION_INDEX Holds the Index of domain.	
7:4	Reserved.	
12:8	DOMAIN_HANDLE Holds the Domain Handle.	
63:13	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 83H, 131		IA32_FZM_DOMAIN_CONFIG
IA32_FZM_DOMAIN_CONFIG (R/O) Bit mask of valid regions within the domain identified by FZM_RANGE_INDEX.		
63:0	REGION_BITMAP Bitmap of valid regions for a given domain.	
Register Address: 84H, 132		IA32_FZM_RANGE_STARTADDR
IA32_FZM_RANGE_STARTADDR (R/O) Start address of the FZM range pointed to by FZM_RANGE_INDEX.		
51:0	START_ADDR Start address of the specified domain in FZM_RANGE_INDEX.	
63:52	Reserved.	
Register Address: 85H, 133		IA32_FZM_RANGE_ENDADDR
IA32_FZM_RANGE_ENDADDR (R/O) End address of the specified domain in FZM_RANGE_INDEX.		
51:0	END_ADDR End address of the specified domain in FZM_RANGE_INDEX.	
63:52	Reserved.	
Register Address: 86H, 134		IA32_FZM_RANGE_WRITESTATUS
IA32_FZM_RANGE_WRITESTATUS (R/O) Write status of the FZM range pointed to by FZM_RANGE_INDEX.		
0	WRITE_STATUS Write status of the specified domain in FZM_RANGE_INDEX.	
1	READ_STATUS Read status of the specified domain in FZM_RANGE_INDEX.	
63:2	Reserved.	
Register Address: 87H, 135		IA32_MKTME_KEYID_PARTITIONING
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX.		
31:0	NUM_MKTME_KEYIDS Number of activated Intel TME-MK KeyIDs. This field is supported on all parts that enumerate support for Intel Total Memory Encryption - Multi-Key (Intel TME-MK). If IA32_TME_ACTIVATE.LOCK is 1, this field reports the number of shared KeyIDs supported by the processor (all KeyIDs other than SEAM-private KeyIDs and KeyID 0); otherwise, it reports 0. Intel TME-MK KeyIDs will always span the KeyID range [1 ... NUM_MKTME_KEYIDS]. Note: KeyID 0 is reserved for TME and will not be included.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:32	NUM_TDX_PRIV_KEYIDS Number of activated SEAM-private KeyIDs. This field is supported on all parts that enumerate support for SEAM mode. If IA32_TME_ACTIVATE.LOCK is 1, this field reports the number of SEAM-private KeyIDs supported by the processor (all KeyIDs, except shared KeyIDs and KeyID 0); otherwise, it reports 0. SEAM-private KeyIDs will always span the range [NUM_MKTME_KEYIDS+1... (NUM_MKTME_KEYIDS + NUM_TDX_PRIV_KEYIDS)]. Note: KeyID 0 is reserved for TME and will not be included.	
Register Address: 8BH, 139		IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)
BIOS Update Signature (R/W) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.		06_01H
31:0	Reserved.	
63:32	PATCH_SIGN_ID It is recommended that this field be preloaded with zero prior to executing CPUID. If the field remains zero following the execution of CPUID, this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature patch signature ID.	
Register Address: 8CH, 140		IA32_SGXLEPUBKEYHASH0
IA32_SGXLEPUBKEYHASH[63:0] (R/W) Bits 63:0 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Read permitted if CPUID.12H.00H:EAX[0] = 1 && CPUID.07H.00H:ECX[30] = 1. Write permitted if CPUID.12H.00H:EAX[0] = 1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1.
Register Address: 8DH, 141		IA32_SGXLEPUBKEYHASH1
IA32_SGXLEPUBKEYHASH[127:64] (R/W) Bits 127:64 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 8EH, 142		IA32_SGXLEPUBKEYHASH2
IA32_SGXLEPUBKEYHASH[191:128] (R/W) Bits 191:128 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 8FH, 143		IA32_SGXLEPUBKEYHASH3
IA32_SGXLEPUBKEYHASH[255:192] (R/W) Bits 255:192 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 90H, 144		IA32_SGXLEPUBKEYHASH4
IA32_SGXLEPUBKEYHASH[319:256] (R/W) Bits 319:256 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 91H, 145		IA32_SGXLEPUBKEYHASH5
IA32_SGXLEPUBKEYHASH[383:320] (R/W) Bits 383:320 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 9BH, 155		IA32_SMM_MONITOR_CTL
SMM Monitor Configuration (R/W)		If CPUID.01H:ECX[5] = 1 CPUID.01H:ECX[6] = 1
0	Valid (R/W)	
1	Reserved.	
2	Controls SMI unblocking by VMXOFF (see Section 34.14.4).	If IA32_VMX_MISC[28]
11:3	Reserved.	
31:12	MSEG Base (R/W)	
63:32	Reserved.	
Register Address: 9EH, 158		IA32_SMBASE
Base address of the logical processor's SMRAM image (R/O, SMM only).		If IA32_VMX_MISC[15]
Register Address: BCH, 188		IA32_MISC_PACKAGE_CTL5
Power Filtering Control (R/W) This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.		If IA32_ARCH_CAPABILITIES [10] = 1
0	ENERGY_FILTERING_ENABLE (R/W) If set, RAPL MSRs report filtered processor power consumption data. This bit can be changed from 0 to 1, but cannot be changed from 1 to 0. After setting, all attempts to clear it are ignored until the next processor reset.	If IA32_ARCH_CAPABILITIES [11] = 1
63:1	Reserved.	
Register Address: BDH, 189		IA32_XAPIC_DISABLE_STATUS
xAPIC Disable Status (R/O)		If CPUID.07H:00H:EDX[29] = 1 and IA32_ARCH_CAPABILITIES [21] = 1
0	LEGACY_XAPIC_DISABLED When set, indicates that the local APIC is in x2APIC mode (IA32_APIC_BASE.EXTD = 1) and that attempts to clear IA32_APIC_BASE.EXTD will fail (e.g., WRMSR will #GP).	
63:1	Reserved.	
Register Address: C1H, 193		IA32_PMC0 (PERFCTR0)
General Performance Counter 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0
Register Address: C2H, 194		IA32_PMC1 (PERFCTR1)
General Performance Counter 1 (R/W)		If CPUID.0AH:EAX[15:8] > 1
Register Address: C3H, 195		IA32_PMC2
General Performance Counter 2 (R/W)		If CPUID.0AH:EAX[15:8] > 2
Register Address: C4H, 196		IA32_PMC3
General Performance Counter 3 (R/W)		If CPUID.0AH:EAX[15:8] > 3

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: C5H, 197		IA32_PMC4
General Performance Counter 4 (R/W)		If CPUID.0AH:EAX[15:8] > 4
Register Address: C6H, 198		IA32_PMC5
General Performance Counter 5 (R/W)		If CPUID.0AH:EAX[15:8] > 5
Register Address: C7H, 199		IA32_PMC6
General Performance Counter 6 (R/W)		If CPUID.0AH:EAX[15:8] > 6
Register Address: C8H, 200		IA32_PMC7
General Performance Counter 7 (R/W)		If CPUID.0AH:EAX[15:8] > 7
Register Address: C9H, 201		IA32_PMC8
General Performance Counter 8 (R/W)		If CPUID.0AH:EAX[15:8] > 8
Register Address: CAH, 202		IA32_PMC9
General Performance Counter 9 (R/W)		If CPUID.0AH:EAX[15:8] > 9
Register Address: CFH, 207		IA32_CORE_CAPABILITIES
IA32 Core Capabilities Register		If CPUID.07H.00H:EDX[30] = 1
63:0	Reserved.	No architecturally defined bits.
Register Address: E1H, 225		IA32_UMWAIT_CONTROL
UMWAIT Control (R/W)		
0	C0.2 is not allowed by the OS. Value of "1" means all C0.2 requests revert to C0.1.	
1	Reserved.	
31:2	Determines the maximum time in TSC-quanta that the processor can reside in either C0.1 or C0.2. A zero value indicates no maximum time. The maximum time value is a 32-bit value where the upper 30 bits come from this field and the lower two bits are zero.	
Register Address: E7H, 231		IA32_MPERF
TSC Frequency Clock Counter (R/Write to clear)		If CPUID.06H:ECX[0] = 1
63:0	CO_MCNT: C0 TSC Frequency Clock Count Increments at fixed interval (relative to TSC freq.) when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_APERF.	
Register Address: E8H, 232		IA32_APERF
Actual Performance Clock Counter (R/Write to clear)		If CPUID.06H:ECX[0] = 1
63:0	CO_ACNT: C0 Actual Frequency Clock Count Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_MPERF.	
Register Address: FEH, 254		IA32_MTRRCAP (MTRRcap)
MTRR Capability (R/O) See Section 14.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		06_01H
7:0	VCNT: The number of variable memory type ranges in the processor.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
8	Fixed range MTRRs are supported when set.	
9	Reserved.	
10	WC supported when set.	
11	SMRR supported when set.	
12	PRMRR supported when set.	
14:13	Reserved.	
15	SEAMRR supported when set.	
63:16	Reserved.	
Register Address: 10AH, 266		IA32_ARCH_CAPABILITIES
Enumeration of Architectural Features (R/O)		If CPUID.07H.00H:EDX[29] = 1
0	RDCL_NO: The processor is not susceptible to Rogue Data Cache Load (RDCL).	
1	IBRS_ALL: The processor supports enhanced IBRS.	
2	RSBA: The processor supports RSB Alternate. Alternative branch predictors may be used by RET instructions when the RSB is empty. SW using retpoline may be affected by this behavior.	
3	SKIP_L1DFL_VMENTRY: A value of 1 indicates the hypervisor need not flush the L1D on VM entry.	
4	SSB_NO: Processor is not susceptible to Speculative Store Bypass.	
5	MDS_NO: Processor is not susceptible to Microarchitectural Data Sampling (MDS).	
6	IF_PSCCHANGE_MC_NO: The processor is not susceptible to a machine check error due to modifying the size of a code page without TLB invalidation.	
7	TSX_CTRL: If 1, indicates presence of IA32_TSX_CTRL MSR.	
8	TAA_NO: If 1, processor is not affected by TAA.	
9	MCU_CONTROL: If 1, the processor supports the IA32_MCU_CONTROL MSR.	
10	MISC_PACKAGE_CTL: The processor supports IA32_MISC_PACKAGE_CTL MSR.	
11	ENERGY_FILTERING_CTL: The processor supports setting and reading the IA32_MISC_PACKAGE_CTL[0] (ENERGY_FILTERING_ENABLE) bit.	
12	DOITM: If 1, the processor supports Data Operand Independent Timing Mode.	
13	SBDR_SSDP_NO: The processor is not affected by either the Shared Buffers Data Read (SBDR) vulnerability or the Sideband Stale Data Propagator (SSDP).	
14	FBSDP_NO: The processor is not affected by the Fill Buffer Stale Data Propagator (FBSDP).	
15	PSDP_NO: The processor is not affected by vulnerabilities involving the Primary Stale Data Propagator (PSDP).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
16	MCU_ENUMERATION: If 1, the processor supports the IA32_MCU_ENUMERATION and IA32_MCU_STATUS MSRs.	
17	FB_CLEAR: If 1, the processor supports overwrite of fill buffer values as part of MD_CLEAR operations with the VERW instruction. On these processors, L1D_FLUSH does not overwrite fill buffer values.	
18	FB_CLEAR_CTRL: If 1, the processor supports the IA32_MCU_OPT_CTRL MSR and allows software to set bit 3 of that MSR (FB_CLEAR_DIS).	
19	RRSBA: A value of 1 indicates the processor may have the RRSBA alternate prediction behavior, if not disabled by RRSBA_DIS_U or RRSBA_DIS_S.	
20	BHI_NO: A value of 1 indicates BHI_NO branch prediction behavior, regardless of the value of IA32_SPEC_CTRL[BHI_DIS_S] MSR bit.	
21	XAPIC_DISABLE_STATUS: Enumerates that the IA32_XAPIC_DISABLE_STATUS MSR exists, and that bit 0 specifies whether the legacy xAPIC is disabled and APIC state is locked to x2APIC.	
22	MCU_EXTENDED_SERVICE: If 1, the processor supports MCU Extended servicing - IA32_MCU_EXT_SERVICE MSR.	
23	OVERCLOCKING_STATUS: If set, the IA32_OVERCLOCKING_STATUS MSR exists.	
24	PBSRB_NO: If 1, the processor is not affected by issues related to Post-Barrier Return Stack Buffer Predictions.	
25	GDS_CTRL: If 1, the processor supports the GDS_MITG_DIS and GDS_MITG_LOCK bits of the IA32_MCU_OPT_CTRL MSR.	
26	GDS_NO: If 1, the processor is not affected by Gather Data Sampling.	
27	RFDS_NO: If 1, the processor is not affected by Register File Data Sampling.	
28	RFDS_CLEAR: If 1, when VERW is executed the processor will clear stale data from register files affected by Register File Data Sampling.	
29	IGN_UMONITOR_SUPPORT If 0, IA32_MCU_OPT_CTRL bit 6 (IGN_UMONITOR) is not supported. If 1, it indicates support of IA32_MCU_OPT_CTRL bit 6 (IGN_UMONITOR).	
30	MON_UMON_MITG_SUPPORT If 0, IA32_MCU_OPT_CTRL bit 7 (MON_UMON_MITG) is not supported. If 1, it indicates support of IA32_MCU_OPT_CTRL bit 7 (MON_UMON_MITG).	
31	Reserved.	
32	PBOPT_SUPPORT If 0, IA32_PBOPT_CTRL bit 0 (Prediction Barrier Option [PBOPT]) is not supported. If 1, IA32_PBOPT_CTRL bit 0 (Prediction Barrier Option [PBOPT]) is supported.	
61:33	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
62	ITS_NO If 0, the hypervisor indicates that the system is not affected by Indirect Target Selection. If 1, the hypervisor indicates that the system may be affected by Indirect Target Selection.		
63	Reserved.		
Register Address: 10BH, 267		IA32_FLUSH_CMD	
Flush Command (W0) Gives software a way to invalidate structures with finer granularity than other architectural methods.			If any one of the enumeration conditions for defined bit field positions holds.
0	L1D_FLUSH Writeback and invalidate the L1 data cache.		If CPUID.07H.00H:EDX[28] = 1
63:1	Reserved.		
Register Address: 10FH, 271		IA32_TSX_FORCE_ABORT	
TSX Force Abort			If CPUID.07H.00H:EDX[13] = 1
0	RTM_FORCE_ABORT If 1, all RTM transactions abort with EAX code 0.		R/W, Default: 0 If CPUID.07H.00H:EDX[11] = 1, bit 0 is always 1 and writes to change it are ignored. If SDV_ENABLE_RTM is 1, bit 0 is always 0 and writes to change it are ignored.
1	TSX_CPUID_CLEAR When set, CPUID.07H.00H:EBX[11] = 0 and CPUID.07H.00H:EBX[4] = 0.		R/W, Default: 0 Can be set only if CPUID.07H.00H:EDX[11] = 1 or if SDV_ENABLE_RTM is 1.
2	SDV_ENABLE_RTM When set, CPUID.07H.00H:EDX[11] = 0 and the processor may not force abort RTM. This unsupported mode should only be used for software development and not for production usage.		R/W, Default: 0 If 0, can be set only if CPUID.07H.00H:EDX[11] = 1.
63:3	Reserved.		
Register Address: 122H, 290		IA32_TSX_CTRL	
IA32_TSX_CTRL (R/W)			Thread scope. Not architecturally serializing. Available when CPUID.07H.00H:EDX.ARCH_CAPABILITIES[29] = 1 and IA32_ARCH_CAPABILITIES.bit 7 = 1.
0	RTM_DISABLE When set to 1, XBEGIN will always abort with EAX code 0.		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
1	TSX_CPUID_CLEAR When set to 1, CPUID.07H.00H:EBX.RTM[11] and CPUID.07H.00H:EBX.HLE[4] report 0. When set to 0 and the SKU supports TSX, these bits will return 1.	
63:2	Reserved.	
Register Address: 123H, 291		IA32_MCU_OPT_CTRL
Microcode Update Option Control (R/W)		If CPUID.07H.00H:EDX[9] = 1 or CPUID.07H.00H:EDX[11] = 1 IA32_ARCH_CAPABILITIES [18] = 1 or IA32_ARCH_CAPABILITIES [25] = 1 or IA32_ARCH_CAPABILITIES [29] = 1 or IA32_ARCH_CAPABILITIES [30] = 1
0	RNGDS_MITG_DIS (R/W) If 0 (default), SRBDS mitigation is enabled for RDRAND and RDSEED. If 1, SRBDS mitigation is disabled for RDRAND and RDSEED executed outside of Intel SGX enclaves.	If CPUID.07H.00H:EDX[9] = 1
1	RTM_ALLOW If 0, XBEGIN will always abort with EAX code 0. If 1, XBEGIN behavior depends on the value of IA32_TSX_CTRL[RTM_DISABLE].	If CPUID.07H.00H:EDX[11] = 1 Read/Write Setting RTM_LOCKED prevents writes to this bit.
2	RTM_LOCKED When 1, RTM_ALLOW is locked at zero, writes to RTM_ALLOW will be ignored.	If CPUID.07H.00H:EDX[11] = 1 Read-Only status bit.
3	FB_CLEAR_DIS If 1, prevents the VERW instruction from performing an FB_CLEAR action.	If IA32_ARCH_CAPABILITIES [18] = 1
4	GDS_MITG_DIS If 0, the Gather Data Sampling mitigation is enabled (patch load time default). If 1 on all threads for a given core, the Gather Data Sampling mitigation is disabled.	If IA32_ARCH_CAPABILITIES [25] = 1
5	GDS_MITG_LOCK If 0, not locked, and GDS_MITG_DIS is under OS control. If 1, locked and GDS_MITG_DIS is forced to 0 (writes are ignored).	If IA32_ARCH_CAPABILITIES [25] = 1
6	IGN_UMONITOR If 0, enable CPL0-3 software to use the UMONITOR/UMWAIT instructions. If 1 (default), disable UMONITOR functionality. CPL0-3 software will be able to call the UMONITOR instruction without causing a fault, however the address monitoring hardware will not be armed. When UMWAIT is called, it will not enter an implementation-dependent optimized state.	If IA32_ARCH_CAPABILITIES [29] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
7	MON_UMON_MITG If 0 (default), disabled. If 1, enable: Flush the thread's previously monitored address from the CPU caches as part of the (U)MONITOR instruction. Additionally, for every 4th (U)MONITOR instruction within a core, flush the peer hyperthread's monitored address from the CPU caches as well. This will increase the latency of the instruction. This may have a minor impact on workloads using the (U)MONITOR instruction.	If IA32_ARCH_CAPABILITIES [30] = 1
63:8	Reserved.	
Register Address: 174H, 372		IA32_SYSENTER_CS
SYSENTER_CS_MSR (R/W)		06_01H
15:0	CS Selector.	
31:16	Not used.	Can be read and written.
63:32	Not used.	Writes ignored; reads return zero.
Register Address: 175H, 373		IA32_SYSENTER_ESP
SYSENTER_ESP_MSR (R/W)		06_01H
Register Address: 176H, 374		IA32_SYSENTER_EIP
SYSENTER_EIP_MSR (R/W)		06_01H
Register Address: 179H, 377		IA32_MCG_CAP (MCG_CAP)
Global Machine Check Capability (R/O)		06_01H
7:0	Count: Number of reporting banks.	
8	MCG_CTL_P: IA32_MCG_CTL is present if this bit is set.	
9	MCG_EXT_P: Extended machine check state registers are present if this bit is set.	
10	MCP_CMCI_P: Support for corrected MC error event is present.	06_01H
11	MCG_TES_P: Threshold-based error status register are present if this bit is set.	
12	MCG_SEAM_NP: Indicates that the processor supports SEAM non-root operation indication in IA32_MCG_STATUS.	
15:13	Reserved.	
23:16	MCG_EXT_CNT: Number of extended machine check state registers present.	
24	MCG_SER_P: The processor supports software error recovery if this bit is set.	
25	Reserved.	
26	MCG_ELOG_P: Indicates that the processor allows platform firmware to be invoked when an error is detected so that it may provide additional platform specific information in an ACPI format "Generic Error Data Entry" that augments the data included in machine check bank registers.	06_3EH
27	MCG_LMCE_P: Indicates that the processor supports extended state in IA32_MCG_STATUS and associated MSR necessary to configure Local Machine Check Exception (LMCE).	06_3EH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:28	Reserved.	
Register Address: 17AH, 378		IA32_MCG_STATUS (MCG_STATUS)
Global Machine Check Status (R/W)		06_01H
0	RIPV. Restart IP valid.	06_01H
1	EIPV. Error IP valid.	06_01H
2	MCIP. Machine check in progress.	06_01H
3	LMCE_S. Local machine check.	If IA32_MCG_CAP.LMCE_P[27] = 1
11:4	Reserved.	
12	SEAM_NR. SEAM non-root operation.	If IA32_MCG_CAP.SEAM_NR_P[12] = 1
63:13	Reserved.	
Register Address: 17BH, 379		IA32_MCG_CTL (MCG_CTL)
Global Machine Check Control (R/W)		If IA32_MCG_CAP.CTL_P[8] = 1
Register Address: 180H–185H, 384–389		N/A
Reserved		06_0EH ²
Register Address: 186H, 390		IA32_PERFVTSEL0 (PERFVTSEL0)
Performance Event Select Register 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0
7:0	Event Select: Selects a performance event logic unit.	
15:8	UMask: Qualifies the microarchitectural condition to detect on the selected event logic.	
16	USR: Counts while in privilege level is not ring 0.	
17	OS: Counts while in privilege level is ring 0.	
18	Edge: Enables edge detection if set.	
19	PC: Enables pin control.	
20	INT: Enables interrupt on counter overflow.	
21	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
22	EN: Enables the corresponding performance counter to commence counting when this bit is set.	
23	INV: Invert the CMASK.	
31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.	
63:32	Reserved.	
Register Address: 187H, 391		IA32_PERFVTSEL1 (PERFVTSEL1)
Performance Event Select Register 1 (R/W)		If CPUID.0AH:EAX[15:8] > 1
Register Address: 188H, 392		IA32_PERFVTSEL2
Performance Event Select Register 2 (R/W)		If CPUID.0AH:EAX[15:8] > 2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 189H, 393		IA32_PERFEVTSEL3
Performance Event Select Register 3 (R/W)		If CPUID.0AH:EAX[15:8] > 3
Register Address: 18AH, 394		IA32_PERFEVTSEL4
Performance Event Select Register 4 (R/W)		If CPUID.0AH:EAX[15:8] > 4
Register Address: 18BH, 395		IA32_PERFEVTSEL5
Performance Event Select Register 5 (R/W)		If CPUID.0AH:EAX[15:8] > 5
Register Address: 18CH, 396		IA32_PERFEVTSEL6
Performance Event Select Register 6 (R/W)		If CPUID.0AH:EAX[15:8] > 6
Register Address: 18DH, 397		IA32_PERFEVTSEL7
Performance Event Select Register 7 (R/W)		If CPUID.0AH:EAX[15:8] > 7
Register Address: 18EH, 398		IA32_PERFEVTSEL8
Performance Event Select Register 8 (R/W)		If CPUID.0AH:EAX[15:8] > 8
Register Address: 18FH, 399		IA32_PERFEVTSEL9
Performance Event Select Register 9 (R/W)		If CPUID.0AH:EAX[15:8] > 9
Register Address: 18AH–194H, 394–404		N/A
Reserved.		06_0EH ³
Register Address: 195H, 405		IA32_OVERCLOCKING_STATUS
Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR.		
0	Overclocking Utilized Indicates if specific forms of overclocking have been enabled on this boot or reset cycle: 0 indicates no, 1 indicates yes.	
1	Undervolt Protection Indicates if the “Dynamic OC Undervolt Protection” security feature is active: 0 indicates disabled, 1 indicates enabled.	
2	Overclocking Secure Status Indicates that overclocking capabilities have been unlocked by BIOS, with or without overclocking: 0 indicates Not Secured, 1 indicates Secure.	
63:3	Reserved.	
Register Address: 196H–197H, 406–407		N/A
Reserved.		06_0EH ³
Register Address: 198H, 408		IA32_PERF_STATUS
Current Performance Status (R/O) See Section 17.1.1, “Software Interface For Initiating Performance State Transitions.”		0F_03H
15:0	Current Performance State Value.	
63:16	Reserved.	
Register Address: 199H, 409		IA32_PERF_CTL

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Performance Control MSR (R/W) Software makes a request for a new Performance state (P-State) by writing this MSR. See Section 17.1.1, "Software Interface For Initiating Performance State Transitions."		0F_03H
15:0	Target performance State Value.	
31:16	Reserved.	
32	Intel® Dynamic Acceleration Technology Engage (R/W) When set to 1: Disengages Intel Dynamic Acceleration Technology.	06_0FH (Mobile only)
63:33	Reserved.	
Register Address: 19AH, 410		IA32_CLOCK_MODULATION
Clock Modulation Control (R/W) See Section 17.8.3, "Software Controlled Clock Modulation."		If CPUID.01H:EDX[22] = 1
0	Extended On-Demand Clock Modulation Duty Cycle.	If CPUID.06H:EAX[5] = 1
3:1	On-Demand Clock Modulation Duty Cycle: Specific encoded values for target duty cycle modulation.	If CPUID.01H:EDX[22] = 1
4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	If CPUID.01H:EDX[22] = 1
63:5	Reserved.	
Register Address: 19BH, 411		IA32_THERM_INTERRUPT
Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the processor's thermal sensors and thermal monitor. See Section 17.8.2, "Thermal Monitor."		If CPUID.01H:EDX[22] = 1
0	High-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
1	Low-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
2	PROCHOT# Interrupt Enable	If CPUID.01H:EDX[22] = 1
3	FORCEPR# Interrupt Enable	If CPUID.01H:EDX[22] = 1
4	Critical Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
7:5	Reserved.	
14:8	Threshold #1 Value	If CPUID.01H:EDX[22] = 1
15	Threshold #1 Interrupt Enable	If CPUID.01H:EDX[22] = 1
22:16	Threshold #2 Value	If CPUID.01H:EDX[22] = 1
23	Threshold #2 Interrupt Enable	If CPUID.01H:EDX[22] = 1
24	Power Limit Notification Enable	If CPUID.06H:EAX[4] = 1
63:25	Reserved.	
Register Address: 19CH, 412		IA32_THERM_STATUS
Thermal Status Information (R/O) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 17.8.2, "Thermal Monitor."		If CPUID.01H:EDX[22] = 1
0	Thermal Status (R/O)	If CPUID.01H:EDX[22] = 1
1	Thermal Status Log (R/W)	If CPUID.01H:EDX[22] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
2	PROCHOT # or FORCEPR# event (R/O)	If CPUID.01H:EDX[22] = 1
3	PROCHOT # or FORCEPR# log (R/WCO)	If CPUID.01H:EDX[22] = 1
4	Critical Temperature Status (R/O)	If CPUID.01H:EDX[22] = 1
5	Critical Temperature Status log (R/WCO)	If CPUID.01H:EDX[22] = 1
6	Thermal Threshold #1 Status (R/O)	If CPUID.01H:ECX[8] = 1
7	Thermal Threshold #1 log (R/WCO)	If CPUID.01H:ECX[8] = 1
8	Thermal Threshold #2 Status (R/O)	If CPUID.01H:ECX[8] = 1
9	Thermal Threshold #2 log (R/WCO)	If CPUID.01H:ECX[8] = 1
10	Power Limitation Status (R/O)	If CPUID.06H:EAX[4] = 1
11	Power Limitation log (R/WCO)	If CPUID.06H:EAX[4] = 1
12	Current Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
13	Current Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
14	Cross Domain Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
15	Cross Domain Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
22:16	Digital Readout (R/O)	If CPUID.06H:EAX[0] = 1
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O)	If CPUID.06H:EAX[0] = 1
31	Reading Valid (R/O)	If CPUID.06H:EAX[0] = 1
63:32	Reserved.	
Register Address: 1A0H, 416		IA32_MISC_ENABLE
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable When set, the fast-strings feature (for REP MOVSB and REP STOSB) is enabled (default). When clear, fast-strings are disabled.	OF_OH
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled. Note: In some products clearing this bit might be ignored in critical thermal conditions, and TM1, TM2, and adaptive thermal throttling will still be activated. The default value of this field varies with product. See respective tables where default value is listed.	OF_OH
6:4	Reserved.	
7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.	OF_OH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS). 0 = BTS is supported.	0F_0H
12	Processor Event Based Sampling (PEBS) Unavailable (R/O) 1 = PEBS is not supported. 0 = PEBS is supported.	06_0FH
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) 0 = Enhanced Intel SpeedStep Technology disabled. 1 = Enhanced Intel SpeedStep Technology enabled.	If CPUID.01H:ECX[7] = 1
17	Reserved.	
18	ENABLE MONITOR FSM (R/W) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 2.	CPUID.00H:EAX > 2 and CPUID.07H:01H:EBX. CPUIDMAXVAL_LIM_RMV[3] = 0
23	xTPR Message Disable (R/W) When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.	If CPUID.01H:ECX[14] = 1
63:24	Reserved. Note: Some older processors defined one of these bits as a disable for the execute-disable feature of paging. If a processor supports this bit, this information is provided in the model-specific tables. See Table 2-3 for the definition of this bit.	
Register Address: 1B0H, 432		IA32_ENERGY_PERF_BIAS
Performance Energy Bias Hint (R/W)		If CPUID.06H:ECX[3] = 1
3:0	Power Policy Preference: 0 indicates preference to highest performance. 15 indicates preference to maximize energy saving.	
63:4	Reserved.	
Register Address: 1B1H, 433		IA32_PACKAGE_THERM_STATUS

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Package Thermal Status Information (R/O) Contains status information about the package's thermal sensor. See Section 17.9, "Package Level Thermal Management."		If CPUID.06H:EAX[6] = 1
0	Pkg Thermal Status (R/O)	
1	Pkg Thermal Status Log (R/W)	
2	Pkg PROCHOT # event. (R/O)	
3	Pkg PROCHOT # log. (R/WC0)	
4	Pkg Critical Temperature Status. (R/O)	
5	Pkg Critical Temperature Status Log. (R/WC0)	
6	Pkg Thermal Threshold #1 Status. (R/O)	
7	Pkg Thermal Threshold #1 Log. (R/WC0)	
8	Pkg Thermal Threshold #2 Status. (R/O)	
9	Pkg Thermal Threshold #1 Log. (R/WC0)	
10	Pkg Power Limitation Status. (R/O)	
11	Pkg Power Limitation Log. (R/WC0)	
15:12	Reserved.	
22:16	Pkg Digital Readout. (R/O)	
25:23	Reserved.	
26	Hardware Feedback Interface Structure Change Status.	If CPUID.06H:EAX[19] = 1
63:27	Reserved.	
Register Address: 1B2H, 434		IA32_PACKAGE_THERM_INTERRUPT
Pkg Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the package's thermal sensor. See Section 17.9, "Package Level Thermal Management."		If CPUID.06H:EAX[6] = 1
0	Pkg High-Temperature Interrupt Enable.	
1	Pkg Low-Temperature Interrupt Enable.	
2	Pkg PROCHOT# Interrupt Enable.	
3	Reserved.	
4	Pkg Overheat Interrupt Enable.	
7:5	Reserved.	
14:8	Pkg Threshold #1 Value.	
15	Pkg Threshold #1 Interrupt Enable.	
22:16	Pkg Threshold #2 Value.	
23	Pkg Threshold #2 Interrupt Enable.	
24	Pkg Power Limit Notification Enable.	
25	Hardware Feedback Interrupt Enable.	If CPUID.06H:EAX[19] = 1
63:26	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 1C4H, 452		IA32_XFD	
Extended Feature Disable Control (R/W) Controls which XSAVE-enabled features are temporarily disabled. See Section 13.14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.		If CPUID.0DH.01H:EAX[4] = 1	
Register Address: 1C5H, 453		IA32_XFD_ERR	
Extended Feature Disable Error Code (R/W) Reports which XSAVE-enabled features caused a fault due to being disabled. See Section 13.14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.		If CPUID.0DH.01H:EAX[4] = 1	
Register Address: 1CCH, 460		IA32_FRED_RSP0	
FRED Regular Stack Pointer 0 If FRED event delivery causes a transition from CPL 3, RSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1	
Register Address: 1CDH, 461		IA32_FRED_RSP1	
FRED Regular Stack Pointer 1 If FRED event delivery causes a transition to stack level 1 (from CPL 0), RSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1	
Register Address: 1CEH, 462		IA32_FRED_RSP2	
FRED Regular Stack Pointer 2 If FRED event delivery causes a transition to stack level 2 (from CPL 0), RSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1	
Register Address: 1CFH, 463		IA32_FRED_RSP3	
FRED Regular Stack Pointer 3 If FRED event delivery causes a transition to stack level 3 (from CPL 0), RSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1	
Register Address: 1D0H, 464		IA32_FRED_STKLVLS	
FRED Stack Levels This MSR is interpreted as an array of 32 2-bit values, one for each vector in the range 0-31. For an exception with vector v (or for a non-maskable interrupt, which always has vector v = 2) that occurs in ring 0, FRED event delivery ensures that the new stack level is at least the value of IA32_FRED_STKLVLS[2v+1:2v].		If CPUID.07H.01H:EAX.FRED[17] = 1	
Register Address: 1D1H, 465		IA32_FRED_SSP1	
FRED Shadow-Stack Pointer 1 If shadow stacks are enabled and FRED event delivery causes a transition to stack level 1, SSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1 AND CPUID.07H.00H:ECX.CET_SS[7] = 1	
2:0	Reserved.		
63:3	Address to the new stack level.		
Register Address: 1D2H, 466		IA32_FRED_SSP2	
FRED Shadow-Stack Pointer 2 If shadow stacks are enabled and FRED event delivery causes a transition to stack level 2, SSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1 AND CPUID.07H.00H:ECX.CET_SS[7] = 1	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
2:0	Reserved.	
63:3	Address to the new stack level.	
Register Address: 1D3H, 467		IA32_FRED_SSP3
FRED Shadow-Stack Pointer 3 If shadow stacks are enabled and FRED event delivery causes a transition to stack level 3, SSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1 AND CPUID.07H.00H:ECX.CET_SS[7] = 1
2:0	Reserved.	
63:3	Address to the new stack level.	
Register Address: 1D4H, 468		IA32_FRED_CONFIG
FRED Configuration Register		If CPUID.07H.01H:EAX.FRED[17] = 1
1:0	CSL:	
2	Reserved.	
3	SSP: If FRED event delivery is not changing stacks, setting this bit to 1 decrements the shadow-stack pointer (SSP) by 8.	
5:4	Reserved.	
8:6	The amount, measured in 64-byte cache lines, that FRED event delivery decrements the regular stack pointer when not changing stacks.	
10:9	The stack level used for maskable interrupts that are delivered in ring 0.	
11	Reserved.	
63:12	The upper bits of the linear address of a page in memory containing event handlers. FRED event delivery loads RIP to refer to an entry point on this page.	
Register Address: 1D9H, 473		IA32_DEBUGCTL (MSR_DEBUGCTLA, MSR_DEBUGCTLB)
Trace/Profile Resource Control (R/W)		06_0EH
0	LBR: Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	06_01H
1	BTF: Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	06_01H
2	BLD: Enable OS bus-lock detection. See Section 20.3.1.6 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.	If (CPUID.07H.00H:ECX[24] = 1)
5:3	Reserved.	
6	TR: Setting this bit to 1 enables branch trace messages to be sent.	06_0EH
7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	06_0EH
8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	06_0EH
9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH
10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H:ECX[15] = 1 && CPUID.0AH:EAX[7:0] > 1
12	FREEZE_PERFMON_ON_PMI: When set, each ENABLE bit of the global counter control MSR are frozen (address 38FH) on a PMI request.	If CPUID.01H:ECX[15] = 1 && CPUID.0AH:EAX[7:0] > 1
13	ENABLE_UNCORE_PMI: When set, enables the logical processor to receive and generate PMI on behalf of the uncore.	06_1AH
14	FREEZE_WHILE_SMM: When set, freezes PerfMon and trace messages while in SMM.	If IA32_PERF_CAPABILITIES[12] = 1
15	RTM_DEBUG: When set, enables DR7 debug bit on XBEGIN.	If (CPUID.07H.00H:EBX[11] = 1)
63:16	Reserved.	
Register Address: 1DDH, 477		IA32_LER_FROM_IP
Last Event Record Source IP Register (R/W)		
63:0	FROM_IP The source IP of the recorded branch or event, in canonical form.	Reset Value: 0
Register Address: 1DEH, 478		IA32_LER_TO_IP
Last Event Record Destination IP Register (R/W)		
63:0	TO_IP The destination IP of the recorded branch or event, in canonical form.	Reset Value: 0
Register Address: 1E0H, 480		IA32_LER_INFO
Last Event Record Info Register (R/W)		
55:0	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
59:56	BR_TYPE The branch type recorded by this LBR. Encodings match those of IA32_LBR_X_INFO.	Reset Value: 0
60	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel® TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel® TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
63	MISPRED The recorded branch taken/not-taken resolution (for conditional branches) or target (for any indirect branch, including RETs) was mispredicted.	Reset Value: 0
Register Address: 1F2H, 498		IA32_SMRR_PHYSBASE
SMRR Base Address (Writeable only in SMM) Base address of SMM memory range.		If IA32_MTRRCAP.SMRR[11] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
7:0	Type. Specifies memory type of the range.	
11:8	Reserved.	
31:12	PhysBase SMRR physical Base Address.	
63:32	Reserved.	
Register Address: 1F3H, 499		IA32_SMRR_PHYSMASK
SMRR Range Mask (Writeable only in SMM) Range Mask of SMM memory range.		If IA32_MTRRCAP[SMRR] = 1
10:0	Reserved.	
11	Valid Enable range mask.	
31:12	PhysMask SMRR address range mask.	
63:32	Reserved.	
Register Address: 1F8H, 504		IA32_PLATFORM_DCA_CAP
DCA Capability (R)		If CPUID.01H:ECX[18] = 1
Register Address: 1F9H, 505		IA32_CPU_DCA_CAP
If set, CPU supports Prefetch-Hint type.		If CPUID.01H:ECX[18] = 1
Register Address: 1FAH, 506		IA32_DCA_0_CAP
DCA type 0 Status and Control register.		If CPUID.01H:ECX[18] = 1
0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	
2:1	TRANSACTION	
6:3	DCA_TYPE	
10:7	DCA_QUEUE_SIZE	
12:11	Reserved.	
16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	
23:17	Reserved.	
24	SW_BLOCK: SW can request DCA block by setting this bit.	
25	Reserved.	
26	HW_BLOCK: Set when DCA is blocked by HW (e.g., CRO.CD = 1).	
31:27	Reserved.	
Register Address: 200H, 512		IA32_MTRR_PHYSBASE0 (MTRRphysBase0)
See Section 14.11.2.3, "Variable Range MTRRs."		If IA32_MTRRCAP[7:0] > 0
Register Address: 201H, 513		IA32_MTRR_PHYSMASK0
MTRRphysMask0		If IA32_MTRRCAP[7:0] > 0
Register Address: 202H, 514		IA32_MTRR_PHYSBASE1
MTRRphysBase1		If IA32_MTRRCAP[7:0] > 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
MTRRphysMask1		If IA32_MTRRCAP[7:0] > 1
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
MTRRphysBase2		If IA32_MTRRCAP[7:0] > 2
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
MTRRphysMask2		If IA32_MTRRCAP[7:0] > 2
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
MTRRphysBase3		If IA32_MTRRCAP[7:0] > 3
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
MTRRphysMask3		If IA32_MTRRCAP[7:0] > 3
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
MTRRphysBase4		If IA32_MTRRCAP[7:0] > 4
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
MTRRphysMask4		If IA32_MTRRCAP[7:0] > 4
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
MTRRphysBase5		If IA32_MTRRCAP[7:0] > 5
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
MTRRphysMask5		If IA32_MTRRCAP[7:0] > 5
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
MTRRphysBase6		If IA32_MTRRCAP[7:0] > 6
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
MTRRphysMask6		If IA32_MTRRCAP[7:0] > 6
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
MTRRphysBase7		If IA32_MTRRCAP[7:0] > 7
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
MTRRphysMask7		If IA32_MTRRCAP[7:0] > 7
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
MTRRphysBase8		If IA32_MTRRCAP[7:0] > 8
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
MTRRphysMask8		If IA32_MTRRCAP[7:0] > 8
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
MTRRphysBase9		If IA32_MTRRCAP[7:0] > 9
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
MTRRphysMask9		If IA32_MTRRCAP[7:0] > 9
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
MTRRfix64K_00000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MTRRfix16K_80000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 259H, 601		IA32_MTRR_FIX16K_A0000
MTRRfix16K_A0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 268H, 616		IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)
See Section 14.11.2.2, "Fixed Range MTRRs."		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 269H, 617		IA32_MTRR_FIX4K_C8000
MTRRfix4K_C8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26AH, 618		IA32_MTRR_FIX4K_D0000
MTRRfix4K_D0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26BH, 619		IA32_MTRR_FIX4K_D8000
MTRRfix4K_D8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26CH, 620		IA32_MTRR_FIX4K_E0000
MTRRfix4K_E0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26DH, 621		IA32_MTRR_FIX4K_E8000
MTRRfix4K_E8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26EH, 622		IA32_MTRR_FIX4K_F0000
MTRRfix4K_F0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26FH, 623		IA32_MTRR_FIX4K_F8000
MTRRfix4K_F8000.		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 277H, 631		IA32_PAT
IA32_PAT (R/W)		If CPUID.01H:EDX.PAT[16] = 1
2:0	PA0	
7:3	Reserved.	
10:8	PA1	
15:11	Reserved.	
18:16	PA2	
23:19	Reserved.	
26:24	PA3	
31:27	Reserved.	
34:32	PA4	
39:35	Reserved.	
42:40	PA5	
47:43	Reserved.	
50:48	PA6	
55:51	Reserved.	
58:56	PA7	
63:59	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 280H, 640		IA32_MCO_CTL2
MSR to enable/disable CMCI capability for bank 0. (R/W) See Section 18.3.2.5, "IA32_MCi_CTL2 MSRs."		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 0
14:0	Corrected error count threshold.	
29:15	Reserved.	
30	CMCI_EN	
63:31	Reserved.	
Register Address: 281H, 641		IA32_MC1_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1
Register Address: 282H, 642		IA32_MC2_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
Register Address: 283H, 643		IA32_MC3_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3
Register Address: 284H, 644		IA32_MC4_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4
Register Address: 285H, 645		IA32_MC5_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
Register Address: 286H, 646		IA32_MC6_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
Register Address: 287H, 647		IA32_MC7_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
Register Address: 288H, 648		IA32_MC8_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
Register Address: 289H, 649		IA32_MC9_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9
Register Address: 28AH, 650		IA32_MC10_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
Register Address: 28BH, 651		IA32_MC11_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11
Register Address: 28CH, 652		IA32_MC12_CTL2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
Register Address: 28DH, 653	IA32_MC13_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
Register Address: 28EH, 654	IA32_MC14_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
Register Address: 28FH, 655	IA32_MC15_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15
Register Address: 290H, 656	IA32_MC16_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16
Register Address: 291H, 657	IA32_MC17_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17
Register Address: 292H, 658	IA32_MC18_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18
Register Address: 293H, 659	IA32_MC19_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
Register Address: 294H, 660	IA32_MC20_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
Register Address: 295H, 661	IA32_MC21_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
Register Address: 296H, 662	IA32_MC22_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
Register Address: 297H, 663	IA32_MC23_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
Register Address: 298H, 664	IA32_MC24_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24
Register Address: 299H, 665	IA32_MC25_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
Register Address: 29AH, 666	IA32_MC26_CTL2	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26
Register Address: 29BH, 667		IA32_MC27_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
Register Address: 29CH, 668		IA32_MC28_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
Register Address: 29DH, 669		IA32_MC29_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
Register Address: 29EH, 670		IA32_MC30_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30
Register Address: 29FH, 671		IA32_MC31_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 31
Register Address: 2DCH, 732		IA32_INTEGRITY_STATUS
IA32_INTEGRITY_STATUS (R/O) Provides status information for integrity features.		If any one enumeration condition for defined bit field holds.
0	STATIC_LOCKSTEP 0: Static Lockstep Mode is not active on this logical processor. 1: Static Lockstep Mode is active on this logical processor.	If CPUID.07H.01H:EDX[24] = 1
63:1	Reserved.	
Register Address: 2FFH, 767		IA32_MTRR_DEF_TYPE
MTRRdefType (R/W)		If CPUID.01H:EDX.MTRR[12] = 1
2:0	Default Memory Type	
9:3	Reserved.	
10	Fixed Range MTRR Enable	
11	MTRR Enable	
63:12	Reserved.	
Register Address: 309H, 777		IA32_FIXED_CTR0
Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any.		If CPUID.0AH:EDX[4:0] > 0 CPUID.0AH:ECX[0] = 1 CPUID.23H.01H:EBX[0] = 1
Register Address: 30AH, 778		IA32_FIXED_CTR1
Fixed-Function Performance Counter 1 (R/W): Counts CPU_CLK_Unhalted.Core.		If CPUID.0AH:EDX[4:0] > 1 CPUID.0AH:ECX[1] = 1 CPUID.23H.01H:EBX[1] = 1
Register Address: 30BH, 779		IA32_FIXED_CTR2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Fixed-Function Performance Counter 2 (R/W): Counts CPU_CLK_Unhalted.Ref.		If CPUID.0AH:EDX[4:0] > 2 CPUID.0AH:ECX[2] = 1 CPUID.23H.01H:EBX[2] = 1
Register Address: 30CH, 780		IA32_FIXED_CTR3
Fixed-Function Performance Counter 3 (R/W): Top-down Microarchitecture Analysis unhalted number of available slots.		If CPUID.0AH:EDX[4:0] > 3 CPUID.0AH:ECX[3] = 1 CPUID.23H.01H:EBX[3] = 1
Register Address: 30DH, 781		IA32_FIXED_CTR4
Fixed-Function Performance Counter 4 (R/W): Top-down bad speculation.		If CPUID.0AH:EDX[4:0] > 4 CPUID.0AH:ECX[4] = 1 CPUID.23H.01H:EBX[4] = 1
47:0	FIXED_COUNTER Top-down bad speculation counter.	
63:46	Reserved.	
Register Address: 30EH, 782		IA32_FIXED_CTR5
Fixed-Function Performance Counter 5 (R/W): Top-down Frontend Bound.		If CPUID.0AH:EDX[4:0] > 5 CPUID.0AH:ECX[5] = 1 CPUID.23H.01H:EBX[5] = 1
47:0	FIXED_COUNTER Top-down Frontend Bound counter.	
63:46	Reserved.	
Register Address: 30FH, 783		IA32_FIXED_CTR6
Fixed-Function Performance Counter 6 (R/W): Top-down retiring.		If CPUID.0AH:EDX[4:0] > 6 CPUID.0AH:ECX[6] = 1 CPUID.23H.01H:EBX[6] = 1
47:0	FIXED_COUNTER Top-down Retiring counter.	
63:46	Reserved.	
Register Address: 345H, 837		IA32_PERF_CAPABILITIES
Read Only MSR that enumerates the existence of performance monitoring features. (R/O)		If CPUID.01H:ECX[15] = 1
5:0	LBR format	
6	PEBS Trap	
7	PEBSSaveArchRegs	
11:8	PEBS Record Format	
12	1: Freeze while SMM is supported.	
13	1: Full width of counter writable via IA32_A_PMCx.	
14	PEBS_BASELINE	
15	1: Performance metrics available.	
16	1: PEBS output will be written into the Intel PT trace stream.	If CPUID.07H.00H:EBX[25] = 1
17	1: Indicates support for PEBS Retire Latency output.	
18	TSX_ADDRESS	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
19	RDPMC_METRICS_CLEAR	
63:20	Reserved.	
Register Address: 38DH, 909		IA32_FIXED_CTR_CTRL
Fixed-Function Performance Counter Control (R/W) Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.		If CPUID.0AH:EAX[7:0] > 1
0	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0.	
1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0.	
2	AnyThr0: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
3	EN0_PMI: Enable PMI when fixed counter 0 overflows.	
4	EN1_OS: Enable Fixed Counter 1 to count while CPL = 0.	
5	EN1_Usr: Enable Fixed Counter 1 to count while CPL > 0.	
6	AnyThr1: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
7	EN1_PMI: Enable PMI when fixed counter 1 overflows.	
8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0.	
9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0.	
10	AnyThr2: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
11	EN2_PMI: Enable PMI when fixed counter 2 overflows.	
12	EN3_OS: Enable Fixed Counter 3 to count while CPL = 0.	
13	EN3_Usr: Enable Fixed Counter 3 to count while CPL > 0.	
14	Reserved.	
15	EN3_PMI: Enable PMI when fixed counter 3 overflows.	
63:16	Reserved.	
Register Address: 38EH, 910		IA32_PERF_GLOBAL_STATUS
Global Performance Counter Status (R/O)		If CPUID.0AH:EAX[7:0] > 0 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Ovf_PMC0: Overflow status of IA32_PMC0.	If CPUID.0AH:EAX[15:8] > 0
1	Ovf_PMC1: Overflow status of IA32_PMC1.	If CPUID.0AH:EAX[15:8] > 1
2	Ovf_PMC2: Overflow status of IA32_PMC2.	If CPUID.0AH:EAX[15:8] > 2
3	Ovf_PMC3: Overflow status of IA32_PMC3.	If CPUID.0AH:EAX[15:8] > 3
n	Ovf_PMCn: Overflow status of IA32_PMCn.	If CPUID.0AH:EAX[15:8] > n

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31:n+1	Reserved.	
32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0.	If CPUID.0AH:EAX[7:0] > 1
33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1.	If CPUID.0AH:EAX[7:0] > 1
34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2.	If CPUID.0AH:EAX[7:0] > 1
32+m	Ovf_FixedCtrm: Overflow status of IA32_FIXED_CTRm.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	OVF_PERF_METRICS: If this bit is set, it indicates that PERF_METRIC counter has overflowed and a PMI is triggered; however, an overflow of fixed counter 3 should normally happen first. If this bit is clear no overflow occurred.	
54:49	Reserved.	
55	Trace_ToPA_PMI: A PMI occurred due to a ToPA entry memory buffer that was completely filled.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	
58	LBR_Frz. LBRs are frozen due to: - IA32_DEBUGCTL.FREEZE_LBR_ON_PMI=1. - The LBR stack overflowed.	If CPUID.0AH:EAX[7:0] > 3
59	CTR_Frz. Performance counters in the core PMU are frozen due to: - IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1. - One or more core PMU counters overflowed.	If CPUID.0AH:EAX[7:0] > 3
60	ASCI: Data in the performance counters in the core PMU may include contributions from the direct or indirect operation Intel SGX to protect an enclave.	If the processor supports Intel® SGX.
61	Ovf_Uncore: Uncore counter overflow status.	If CPUID.0AH:EAX[7:0] > 2
62	OvfBuf: DS SAVE area Buffer overflow status.	If CPUID.0AH:EAX[7:0] > 0
63	CondChgd: Status bits of this register have changed.	If CPUID.0AH:EAX[7:0] > 0
Register Address: 38FH, 911		IA32_PERF_GLOBAL_CTRL
Global Performance Counter Control (R/W) Counter increments while the result of ANDing the respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.		If CPUID.0AH:EAX[7:0] > 0
0	EN_PMC0	If CPUID.0AH:EAX[15:8] > 0
1	EN_PMC1	If CPUID.0AH:EAX[15:8] > 1
2	EN_PMC2	If CPUID.0AH:EAX[15:8] > 2
n	EN_PMCn	If CPUID.0AH:EAX[15:8] > n
31:n+1	Reserved.	
32	EN_FIXED_CTR0	If CPUID.0AH:EDX[4:0] > 0
33	EN_FIXED_CTR1	If CPUID.0AH:EDX[4:0] > 1
34	EN_FIXED_CTR2	If CPUID.0AH:EDX[4:0] > 2
32+m	EN_FIXED_CTRm	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:33+m	Reserved.	
48	EN_PERF_METRICS: If this bit is set and fixed counter 3 is effectively enabled, built-in performance metrics are enabled.	
63:49	Reserved.	
Register Address: 390H, 912		IA32_PERF_GLOBAL_STATUS_RESET
Global Performance Counter Overflow Reset Control (R/W)		If CPUID.0AH:EAX[7:0] > 3 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH:EAX[15:8] > 0
1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH:EAX[15:8] > 1
2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH:EAX[15:8] > 2
n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH:EAX[15:8] > n
31:n	Reserved.	
32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH:EDX[4:0] > 0
33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH:EDX[4:0] > 1
34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH:EDX[4:0] > 2
32+m	Set 1 to Clear Ovf_FIXED_CTRm bit.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	RESET_OVF_PERF_METRICS: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
54:49	Reserved.	
55	Set 1 to Clear Trace_ToPA_PMI bit.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	
58	Set 1 to Clear LBR_Frz bit.	If CPUID.0AH:EAX[7:0] > 3
59	Set 1 to Clear CTR_Frz bit.	If CPUID.0AH:EAX[7:0] > 3
60	Set 1 to Clear ASCII bit.	If the processor supports Intel® SGX.
61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
62	Set 1 to Clear OvfBuf bit.	If CPUID.0AH:EAX[7:0] > 0
63	Set 1 to clear CondChgd bit.	If CPUID.0AH:EAX[7:0] > 0
Register Address: 391H, 913		IA32_PERF_GLOBAL_STATUS_SET
Global Performance Counter Overflow Set Control (R/W)		If CPUID.0AH:EAX[7:0] > 3 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Set 1 to cause Ovf_PMC0 = 1.	If CPUID.0AH:EAX[7:0] > 3
1	Set 1 to cause Ovf_PMC1 = 1.	If CPUID.0AH:EAX[15:8] > 1
2	Set 1 to cause Ovf_PMC2 = 1.	If CPUID.0AH:EAX[15:8] > 2
n	Set 1 to cause Ovf_PMCn = 1.	If CPUID.0AH:EAX[15:8] > n

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31:n	Reserved.	
32	Set 1 to cause Ovf_FIXED_CTR0 = 1.	If CPUID.0AH:EAX[7:0] > 3
33	Set 1 to cause Ovf_FIXED_CTR1 = 1.	If CPUID.0AH:EAX[7:0] > 3
34	Set 1 to cause Ovf_FIXED_CTR2 = 1.	If CPUID.0AH:EAX[7:0] > 3
32+m	Set 1 to cause Ovf_FIXED_CTRm = 1.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	SET_OVF_PERF_METRICS: If this bit is set, it will set the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
54:49	Reserved.	
55	Set 1 to cause Trace_ToPA_PMI = 1.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	
58	Set 1 to cause LBR_Frз = 1.	If CPUID.0AH:EAX[7:0] > 3
59	Set 1 to cause CTR_Frз = 1.	If CPUID.0AH:EAX[7:0] > 3
60	Set 1 to cause ASCI = 1.	If the processor supports Intel® SGX.
61	Set 1 to cause Ovf_Uncore = 1.	If CPUID.0AH:EAX[7:0] > 3
62	Set 1 to cause OvfBuf = 1.	If CPUID.0AH:EAX[7:0] > 3
63	Reserved.	
Register Address: 392H, 914		IA32_PERF_GLOBAL_INUSE
Indicator that core PerfMon interface is in use. (R/O)		If CPUID.0AH:EAX[7:0] > 3
0	IA32_PERFEVTSEL0 in use.	
1	IA32_PERFEVTSEL1 in use.	If CPUID.0AH:EAX[15:8] > 1
2	IA32_PERFEVTSEL2 in use.	If CPUID.0AH:EAX[15:8] > 2
n	IA32_PERFEVTSELn in use.	If CPUID.0AH:EAX[15:8] > n
31:n+1	Reserved.	
32	IA32_FIXED_CTR0 in use.	
33	IA32_FIXED_CTR1 in use.	
34	IA32_FIXED_CTR2 in use.	
32+m	IA32_FIXED_CTRm in use.	
62:33+m	Reserved or model specific.	
63	PMI in use.	
Register Address: 3F1H, 1009		IA32_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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:36	Reserved.	
Register Address: 3F4H, 1012		IA32_PEBB_BASE
Architectural PEBB Output Buffer Base Register (R/W) Holds basic structural controls for Architectural PEBB. This MSR is initialized to all zeros on reset.		CPUID.23H.00H:EAX[5]=1
3:0	SIZE (R/W) Size options for the PEBB output buffer (actual size is power-of-2 multiples of 4KB, that is, Buffer Size = 4KB * 2^SIZE).	
11:4	Reserved.	
MAXPHYADDR - 1:12	PHYS_ADDR (R/W) Start of the PEBB output buffer (4KB-aligned physical address, excluding the 12b page offset).	
63:MAXPHYADDR	Reserved.	
Register Address: 3F5H, 1013		IA32_PEBB_INDEX
Arch PEBB Output Buffer Index Register (R/W) Holds status and additional controls for Architectural PEBB. This MSR is initialized to all zeros on reset.		CPUID.23H.00H:EAX[5]=1
3:0	Reserved.	
26:4	WR_OFFSET (R/W) Next-byte-to-write offset from IA32_PEBB_BASE.PHYS_ADDR.	
30:27	Reserved.	
31	FULL (R/W) The buffer is full, and no further records will be generated.	
32	THRESH_EN (R/W) Enable PMI on PEBB buffer Threshold checks.	
35:33	Reserved.	
58:36	THRESH_OFFSET (R/W) The threshold for the PEBB buffer (specified as offset in bytes from IA32_PEBB_BASE.PHYS_ADDR).	
63:59	Reserved.	
Register Address: 400H, 1024		IA32_MCO_CTL
MCO_CTL		If IA32_MCG_CAP.CNT > 0
Register Address: 401H, 1025		IA32_MCO_STATUS
MCO_STATUS		If IA32_MCG_CAP.CNT > 0
Register Address: 402H, 1026		IA32_MCO_ADDR ¹
MCO_ADDR		If IA32_MCG_CAP.CNT > 0
Register Address: 403H, 1027		IA32_MCO_MISC
MCO_MISC		If IA32_MCG_CAP.CNT > 0
Register Address: 404H, 1028		IA32_MC1_CTL
MC1_CTL		If IA32_MCG_CAP.CNT > 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 405H, 1029		IA32_MC1_STATUS	
MC1_STATUS		If IA32_MCG_CAP.CNT > 1	
Register Address: 406H, 1030		IA32_MC1_ADDR ²	
MC1_ADDR		If IA32_MCG_CAP.CNT > 1	
Register Address: 407H, 1031		IA32_MC1_MISC	
MC1_MISC		If IA32_MCG_CAP.CNT > 1	
Register Address: 408H, 1032		IA32_MC2_CTL	
MC2_CTL		If IA32_MCG_CAP.CNT > 2	
Register Address: 409H, 1033		IA32_MC2_STATUS	
MC2_STATUS		If IA32_MCG_CAP.CNT > 2	
Register Address: 40AH, 1034		IA32_MC2_ADDR ¹	
MC2_ADDR		If IA32_MCG_CAP.CNT > 2	
Register Address: 40BH, 1035		IA32_MC2_MISC	
MC2_MISC		If IA32_MCG_CAP.CNT > 2	
Register Address: 40CH, 1036		IA32_MC3_CTL	
MC3_CTL		If IA32_MCG_CAP.CNT > 3	
Register Address: 40DH, 1037		IA32_MC3_STATUS	
MC3_STATUS		If IA32_MCG_CAP.CNT > 3	
Register Address: 40EH, 1038		IA32_MC3_ADDR ¹	
MC3_ADDR		If IA32_MCG_CAP.CNT > 3	
Register Address: 40FH, 1039		IA32_MC3_MISC	
MC3_MISC		If IA32_MCG_CAP.CNT > 3	
Register Address: 410H, 1040		IA32_MC4_CTL	
MC4_CTL		If IA32_MCG_CAP.CNT > 4	
Register Address: 411H, 1041		IA32_MC4_STATUS	
MC4_STATUS		If IA32_MCG_CAP.CNT > 4	
Register Address: 412H, 1042		IA32_MC4_ADDR ¹	
MC4_ADDR		If IA32_MCG_CAP.CNT > 4	
Register Address: 413H, 1043		IA32_MC4_MISC	
MC4_MISC		If IA32_MCG_CAP.CNT > 4	
Register Address: 414H, 1044		IA32_MC5_CTL	
MC5_CTL		If IA32_MCG_CAP.CNT > 5	
Register Address: 415H, 1045		IA32_MC5_STATUS	
MC5_STATUS		If IA32_MCG_CAP.CNT > 5	
Register Address: 416H, 1046		IA32_MC5_ADDR ¹	
MC5_ADDR		If IA32_MCG_CAP.CNT > 5	
Register Address: 417H, 1047		IA32_MC5_MISC	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC5_MISC		If IA32_MCG_CAP.CNT > 5
Register Address: 418H, 1048	IA32_MC6_CTL	
MC6_CTL		If IA32_MCG_CAP.CNT > 6
Register Address: 419H, 1049	IA32_MC6_STATUS	
MC6_STATUS		If IA32_MCG_CAP.CNT > 6
Register Address: 41AH, 1050	IA32_MC6_ADDR ¹	
MC6_ADDR		If IA32_MCG_> 6CAP.CNT > 6
Register Address: 41BH, 1051	IA32_MC6_MISC	
MC6_MISC		If IA32_MCG_CAP.CNT > 6
Register Address: 41CH, 1052	IA32_MC7_CTL	
MC7_CTL		If IA32_MCG_CAP.CNT > 7
Register Address: 41DH, 1053	IA32_MC7_STATUS	
MC7_STATUS		If IA32_MCG_CAP.CNT > 7
Register Address: 41EH, 1054	IA32_MC7_ADDR ¹	
MC7_ADDR		If IA32_MCG_CAP.CNT > 7
Register Address: 41FH, 1055	IA32_MC7_MISC	
MC7_MISC		If IA32_MCG_CAP.CNT > 7
Register Address: 420H, 1056	IA32_MC8_CTL	
MC8_CTL		If IA32_MCG_CAP.CNT > 8
Register Address: 421H, 1057	IA32_MC8_STATUS	
MC8_STATUS		If IA32_MCG_CAP.CNT > 8
Register Address: 422H, 1058	IA32_MC8_ADDR ¹	
MC8_ADDR		If IA32_MCG_CAP.CNT > 8
Register Address: 423H, 1059	IA32_MC8_MISC	
MC8_MISC		If IA32_MCG_CAP.CNT > 8
Register Address: 424H, 1060	IA32_MC9_CTL	
MC9_CTL		If IA32_MCG_CAP.CNT > 9
Register Address: 425H, 1061	IA32_MC9_STATUS	
MC9_STATUS		If IA32_MCG_CAP.CNT > 9
Register Address: 426H, 1062	IA32_MC9_ADDR ¹	
MC9_ADDR		If IA32_MCG_CAP.CNT > 9
Register Address: 427H, 1063	IA32_MC9_MISC	
MC9_MISC		If IA32_MCG_CAP.CNT > 9
Register Address: 428H, 1064	IA32_MC10_CTL	
MC10_CTL		If IA32_MCG_CAP.CNT > 10
Register Address: 429H, 1065	IA32_MC10_STATUS	
MC10_STATUS		If IA32_MCG_CAP.CNT > 10

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 42AH, 1066		IA32_MC10_ADDR ¹	
MC10_ADDR		If IA32_MCG_CAP.CNT > 10	
Register Address: 42BH, 1067		IA32_MC10_MISC	
MC10_MISC		If IA32_MCG_CAP.CNT > 10	
Register Address: 42CH, 1068		IA32_MC11_CTL	
MC11_CTL		If IA32_MCG_CAP.CNT > 11	
Register Address: 42DH, 1069		IA32_MC11_STATUS	
MC11_STATUS		If IA32_MCG_CAP.CNT > 11	
Register Address: 42EH, 1070		IA32_MC11_ADDR ¹	
MC11_ADDR		If IA32_MCG_CAP.CNT > 11	
Register Address: 42FH, 1071		IA32_MC11_MISC	
MC11_MISC		If IA32_MCG_CAP.CNT > 11	
Register Address: 430H, 1072		IA32_MC12_CTL	
MC12_CTL		If IA32_MCG_CAP.CNT > 12	
Register Address: 431H, 1073		IA32_MC12_STATUS	
MC12_STATUS		If IA32_MCG_CAP.CNT > 12	
Register Address: 432H, 1074		IA32_MC12_ADDR ¹	
MC12_ADDR		If IA32_MCG_CAP.CNT > 12	
Register Address: 433H, 1075		IA32_MC12_MISC	
MC12_MISC		If IA32_MCG_CAP.CNT > 12	
Register Address: 434H, 1076		IA32_MC13_CTL	
MC13_CTL		If IA32_MCG_CAP.CNT > 13	
Register Address: 435H, 1077		IA32_MC13_STATUS	
MC13_STATUS		If IA32_MCG_CAP.CNT > 13	
Register Address: 436H, 1078		IA32_MC13_ADDR ¹	
MC13_ADDR		If IA32_MCG_CAP.CNT > 13	
Register Address: 437H, 1079		IA32_MC13_MISC	
MC13_MISC		If IA32_MCG_CAP.CNT > 13	
Register Address: 438H, 1080		IA32_MC14_CTL	
MC14_CTL		If IA32_MCG_CAP.CNT > 14	
Register Address: 439H, 1081		IA32_MC14_STATUS	
MC14_STATUS		If IA32_MCG_CAP.CNT > 14	
Register Address: 43AH, 1082		IA32_MC14_ADDR ¹	
MC14_ADDR		If IA32_MCG_CAP.CNT > 14	
Register Address: 43BH, 1083		IA32_MC14_MISC	
MC14_MISC		If IA32_MCG_CAP.CNT > 14	
Register Address: 43CH, 1084		IA32_MC15_CTL	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC15_CTL		If IA32_MCG_CAP.CNT > 15
Register Address: 43DH, 1085	IA32_MC15_STATUS	
MC15_STATUS		If IA32_MCG_CAP.CNT > 15
Register Address: 43EH, 1086	IA32_MC15_ADDR ¹	
MC15_ADDR		If IA32_MCG_CAP.CNT > 15
Register Address: 43FH, 1087	IA32_MC15_MISC	
MC15_MISC		If IA32_MCG_CAP.CNT > 15
Register Address: 440H, 1088	IA32_MC16_CTL	
MC16_CTL		If IA32_MCG_CAP.CNT > 16
Register Address: 441H, 1089	IA32_MC16_STATUS	
MC16_STATUS		If IA32_MCG_CAP.CNT > 16
Register Address: 442H, 1090	IA32_MC16_ADDR ¹	
MC16_ADDR		If IA32_MCG_CAP.CNT > 16
Register Address: 443H, 1091	IA32_MC16_MISC	
MC16_MISC		If IA32_MCG_CAP.CNT > 16
Register Address: 444H, 1092	IA32_MC17_CTL	
MC17_CTL		If IA32_MCG_CAP.CNT > 17
Register Address: 445H, 1093	IA32_MC17_STATUS	
MC17_STATUS		If IA32_MCG_CAP.CNT > 17
Register Address: 446H, 1094	IA32_MC17_ADDR ¹	
MC17_ADDR		If IA32_MCG_CAP.CNT > 17
Register Address: 447H, 1095	IA32_MC17_MISC	
MC17_MISC		If IA32_MCG_CAP.CNT > 17
Register Address: 448H, 1096	IA32_MC18_CTL	
MC18_CTL		If IA32_MCG_CAP.CNT > 18
Register Address: 449H, 1097	IA32_MC18_STATUS	
MC18_STATUS		If IA32_MCG_CAP.CNT > 18
Register Address: 44AH, 1098	IA32_MC18_ADDR ¹	
MC18_ADDR		If IA32_MCG_CAP.CNT > 18
Register Address: 44BH, 1099	IA32_MC18_MISC	
MC18_MISC		If IA32_MCG_CAP.CNT > 18
Register Address: 44CH, 1100	IA32_MC19_CTL	
MC19_CTL		If IA32_MCG_CAP.CNT > 19
Register Address: 44DH, 1101	IA32_MC19_STATUS	
MC19_STATUS		If IA32_MCG_CAP.CNT > 19
Register Address: 44EH, 1102	IA32_MC19_ADDR ¹	
MC19_ADDR		If IA32_MCG_CAP.CNT > 19

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 44FH, 1103		IA32_MC19_MISC	
MC19_MISC		If IA32_MCG_CAP.CNT > 19	
Register Address: 450H, 1104		IA32_MC20_CTL	
MC20_CTL		If IA32_MCG_CAP.CNT > 20	
Register Address: 451H, 1105		IA32_MC20_STATUS	
MC20_STATUS		If IA32_MCG_CAP.CNT > 20	
Register Address: 452H, 1106		IA32_MC20_ADDR ¹	
MC20_ADDR		If IA32_MCG_CAP.CNT > 20	
Register Address: 453H, 1107		IA32_MC20_MISC	
MC20_MISC		If IA32_MCG_CAP.CNT > 20	
Register Address: 454H, 1108		IA32_MC21_CTL	
MC21_CTL		If IA32_MCG_CAP.CNT > 21	
Register Address: 455H, 1109		IA32_MC21_STATUS	
MC21_STATUS		If IA32_MCG_CAP.CNT > 21	
Register Address: 456H, 1110		IA32_MC21_ADDR ¹	
MC21_ADDR		If IA32_MCG_CAP.CNT > 21	
Register Address: 457H, 1111		IA32_MC21_MISC	
MC21_MISC		If IA32_MCG_CAP.CNT > 21	
Register Address: 458H, 1112		IA32_MC22_CTL	
MC22_CTL		If IA32_MCG_CAP.CNT > 22	
Register Address: 459H, 1113		IA32_MC22_STATUS	
MC22_STATUS		If IA32_MCG_CAP.CNT > 22	
Register Address: 45AH, 1114		IA32_MC22_ADDR ¹	
MC22_ADDR		If IA32_MCG_CAP.CNT > 22	
Register Address: 45BH, 1115		IA32_MC22_MISC	
MC22_MISC		If IA32_MCG_CAP.CNT > 22	
Register Address: 45CH, 1116		IA32_MC23_CTL	
MC23_CTL		If IA32_MCG_CAP.CNT > 23	
Register Address: 45DH, 1117		IA32_MC23_STATUS	
MC23_STATUS		If IA32_MCG_CAP.CNT > 23	
Register Address: 45EH, 1118		IA32_MC23_ADDR ¹	
MC23_ADDR		If IA32_MCG_CAP.CNT > 23	
Register Address: 45FH, 1119		IA32_MC23_MISC	
MC23_MISC		If IA32_MCG_CAP.CNT > 23	
Register Address: 460H, 1120		IA32_MC24_CTL	
MC24_CTL		If IA32_MCG_CAP.CNT > 24	
Register Address: 461H, 1121		IA32_MC24_STATUS	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC24_STATUS		If IA32_MCG_CAP.CNT > 24
Register Address: 462H, 1122		IA32_MC24_ADDR ¹
MC24_ADDR		If IA32_MCG_CAP.CNT > 24
Register Address: 463H, 1123		IA32_MC24_MISC
MC24_MISC		If IA32_MCG_CAP.CNT > 24
Register Address: 464H, 1124		IA32_MC25_CTL
MC25_CTL		If IA32_MCG_CAP.CNT > 25
Register Address: 465H, 1125		IA32_MC25_STATUS
MC25_STATUS		If IA32_MCG_CAP.CNT > 25
Register Address: 466H, 1126		IA32_MC25_ADDR ¹
MC25_ADDR		If IA32_MCG_CAP.CNT > 25
Register Address: 467H, 1127		IA32_MC25_MISC
MC25_MISC		If IA32_MCG_CAP.CNT > 25
Register Address: 468H, 1128		IA32_MC26_CTL
MC26_CTL		If IA32_MCG_CAP.CNT > 26
Register Address: 469H, 1129		IA32_MC26_STATUS
MC26_STATUS		If IA32_MCG_CAP.CNT > 26
Register Address: 46AH, 1130		IA32_MC26_ADDR ¹
MC26_ADDR		If IA32_MCG_CAP.CNT > 26
Register Address: 46BH, 1131		IA32_MC26_MISC
MC26_MISC		If IA32_MCG_CAP.CNT > 26
Register Address: 46CH, 1132		IA32_MC27_CTL
MC27_CTL		If IA32_MCG_CAP.CNT > 27
Register Address: 46DH, 1133		IA32_MC27_STATUS
MC27_STATUS		If IA32_MCG_CAP.CNT > 27
Register Address: 46EH, 1134		IA32_MC27_ADDR ¹
MC27_ADDR		If IA32_MCG_CAP.CNT > 27
Register Address: 46FH, 1135		IA32_MC27_MISC
MC27_MISC		If IA32_MCG_CAP.CNT > 27
Register Address: 470H, 1136		IA32_MC28_CTL
MC28_CTL		If IA32_MCG_CAP.CNT > 28
Register Address: 471H, 1137		IA32_MC28_STATUS
MC28_STATUS		If IA32_MCG_CAP.CNT > 28
Register Address: 472H, 1138		IA32_MC28_ADDR ¹
MC28_ADDR		If IA32_MCG_CAP.CNT > 28
Register Address: 473H, 1139		IA32_MC28_MISC
MC28_MISC		If IA32_MCG_CAP.CNT > 28

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 474H, 1140		IA32_MC29_CTL
MC29_CTL		If IA32_MCG_CAP.CNT > 29
Register Address: 475H, 1141		IA32_MC29_STATUS
MC29_STATUS		If IA32_MCG_CAP.CNT > 29
Register Address: 476H, 1142		IA32_MC29_ADDR
MC29_ADDR		If IA32_MCG_CAP.CNT > 29
Register Address: 477H, 1143		IA32_MC29_MISC
MC29_MISC		If IA32_MCG_CAP.CNT > 29
Register Address: 478H, 1144		IA32_MC30_CTL
MC30_CTL		If IA32_MCG_CAP.CNT > 30
Register Address: 479H, 1145		IA32_MC30_STATUS
MC30_STATUS		If IA32_MCG_CAP.CNT > 30
Register Address: 47AH, 1146		IA32_MC30_ADDR
MC30_ADDR		If IA32_MCG_CAP.CNT > 30
Register Address: 47BH, 1147		IA32_MC30_MISC
MC30_MISC		If IA32_MCG_CAP.CNT > 30
Register Address: 47CH, 1148		IA32_MC31_CTL
MC31_CTL		If IA32_MCG_CAP.CNT > 31
Register Address: 47DH, 1149		IA32_MC31_STATUS
MC31_STATUS		If IA32_MCG_CAP.CNT > 31
Register Address: 47EH, 1150		IA32_MC31_ADDR
MC31_ADDR		If IA32_MCG_CAP.CNT > 31
Register Address: 47FH, 1151		IA32_MC31_MISC
MC31_MISC		If IA32_MCG_CAP.CNT > 31
Register Address: 480H, 1152		IA32_VMX_BASIC
Reporting Register of Basic VMX Capabilities (R/O) See Appendix A.1, "Basic VMX Information."		If CPUID.01H:ECX[5] = 1
Register Address: 481H, 1153		IA32_VMX_PINBASED_CTL
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 482H, 1154		IA32_VMX_PROCBASED_CTL
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 483H, 1155		IA32_VMX_EXIT_CTL
Capability Reporting Register of Primary VM-Exit Controls (R/O) See Appendix A.4.1, "Primary VM-Exit Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 484H, 1156		IA32_VMX_ENTRY_CTL

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 485H, 1157		IA32_VMX_MISC
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data."		If CPUID.01H:ECX[5] = 1
Register Address: 486H, 1158		IA32_VMX_CR0_FIXED0
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0."		If CPUID.01H:ECX[5] = 1
Register Address: 487H, 1159		IA32_VMX_CR0_FIXED1
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0."		If CPUID.01H:ECX[5] = 1
Register Address: 488H, 1160		IA32_VMX_CR4_FIXED0
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."		If CPUID.01H:ECX[5] = 1
Register Address: 489H, 1161		IA32_VMX_CR4_FIXED1
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."		If CPUID.01H:ECX[5] = 1
Register Address: 48AH, 1162		IA32_VMX_VMCS_ENUM
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration."		If CPUID.01H:ECX[5] = 1
Register Address: 48BH, 1163		IA32_VMX_PROCBASED_CTL2
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.3, "Secondary Processor-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL2[63])
Register Address: 48CH, 1164		IA32_VMX_EPT_VPID_CAP
Capability Reporting Register of EPT and VPID (R/O) See Appendix A.10, "VPID and EPT Capabilities."		If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL2[63 3] IA32_VMX_PROCBASED_CTL2[3 7]))
Register Address: 48DH, 1165		IA32_VMX_TRUE_PINBASED_CTL2
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])
Register Address: 48EH, 1166		IA32_VMX_TRUE_PROCBASED_CTL2
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])
Register Address: 48FH, 1167		IA32_VMX_TRUE_EXIT_CTL2
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Appendix A.4, "VM-Exit Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTL5	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Appendix A.5, "VM-Entry Controls."	If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])	
Register Address: 491H, 1169	IA32_VMX_VMFUNC	
Capability Reporting Register of VM-Function Controls (R/O)	If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL5[63] && IA32_VMX_PROCBASED_CTL5[45])	
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTL53	
Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.4, "Tertiary Processor-Based VM-Execution Controls."	If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL5[49])	
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL52	
Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Appendix A.4.2, "Secondary VM-Exit Controls."	If (CPUID.01H:ECX[5] && IA32_VMX_EXIT_CTL5[63])	
Register Address: 4C1H, 1217	IA32_A_PMC0	
Full Width Writable IA32_PMC0 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 0) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[0] = 1	
Register Address: 4C2H, 1218	IA32_A_PMC1	
Full Width Writable IA32_PMC1 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 1) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[1] = 1	
Register Address: 4C3H, 1219	IA32_A_PMC2	
Full Width Writable IA32_PMC2 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 2) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[2] = 1	
Register Address: 4C4H, 1220	IA32_A_PMC3	
Full Width Writable IA32_PMC3 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 3) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[3] = 1	
Register Address: 4C5H, 1221	IA32_A_PMC4	
Full Width Writable IA32_PMC4 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 4) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[4] = 1	
Register Address: 4C6H, 1222	IA32_A_PMC5	
Full Width Writable IA32_PMC5 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 5) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[5] = 1	
Register Address: 4C7H, 1223	IA32_A_PMC6	
Full Width Writable IA32_PMC6 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 6) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[6] = 1	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 4C8H, 1224		IA32_A_PMC7
Full Width Writable IA32_PMC7 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 7) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[7] = 1
Register Address: 4C9H, 1225		IA32_A_PMC8
Full Width Writable IA32_PMC8 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 8) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[8] = 1
Register Address: 4CAH, 1226		IA32_A_PMC9
Full Width Writable IA32_PMC9 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 9) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[9] = 1
Register Address: 4DOH, 1232		IA32_MCG_EXT_CTL
Allows software to signal some MCEs to only a single logical processor in the system. (R/W) See Section 18.3.1.4, "IA32_MCG_EXT_CTL MSR."		If IA32_MCG_CAP.LMCE_P = 1
0	LMCE_EN Enable / Disable local machine check exception.	
63:1	Reserved.	
Register Address: 500H, 1280		IA32_SGX_SVN_STATUS
Status and SVN Threshold of SGX Support for ACM (R/O)		If CPUID.07H.00H:EBX[2] = 1
0	Lock.	See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."
15:1	Reserved.	
23:16	SGX_SVN_SINIT	See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."
63:24	Reserved.	
Register Address: 560H, 1376		IA32_RTIT_OUTPUT_BASE
Trace Output Base Register (R/W)		If ((CPUID.07H.00H:EBX[25] = 1) && ((CPUID.14H.00H:ECX[0] = 1) (CPUID.14H.00H:ECX[2] = 1)))
6:0	Reserved.	
M-1:7	Base physical address.	M is the value enumerated by CPUID.80000008H:EAX[7:0].
63:M	Reserved.	
Register Address: 561H, 1377		IA32_RTIT_OUTPUT_MASK_PTRS
Trace Output Mask Pointers Register (R/W)		If ((CPUID.07H.00H:EBX[25] = 1) && ((CPUID.14H.00H:ECX[0] = 1) (CPUID.14H.00H:ECX[2] = 1)))
6:0	Reserved.	
31:7	MaskOrTableOffset.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:32	Output Offset.	
Register Address: 570H, 1392		IA32_RTIT_CTL
Trace Control Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
0	TraceEn	
1	CYCEn	If (CPUID.07H.00H:EBX[1] = 1)
2	OS	
3	User	
4	PwrEvtEn	If (CPUID.07H.01H:EBX[5] = 1)
5	FUPonPTW	If (CPUID.07H.01H:EBX[4] = 1)
6	FabricEn	If (CPUID.07H.00H:ECX[3] = 1)
7	CR3Filter	If (CPUID.14H.00H:EBX[0] = 1)
8	ToPA	
9	MTCEn	If (CPUID.07H.00H:EBX[3] = 1)
10	TSCEn	
11	DisRETC	
12	PTWEn	If (CPUID.07H.01H:EBX[4] = 1)
13	BranchEn	
17:14	MTCFreq.	If (CPUID.07H.00H:EBX[3] = 1)
18	Reserved, must be zero.	
22:19	CycThresh	If (CPUID.07H.00H:EBX[1] = 1)
23	Reserved, must be zero.	
27:24	PSBFreq	If (CPUID.07H.00H:EBX[1] = 1)
30:28	Reserved, must be zero.	
31	EventEn	If (CPUID.14H.00H:EBX[7] = 1)
35:32	ADDR0_CFG	If (CPUID.07H.01H:EAX[2:0] > 0)
39:36	ADDR1_CFG	If (CPUID.07H.01H:EAX[2:0] > 1)
43:40	ADDR2_CFG	If (CPUID.07H.01H:EAX[2:0] > 2)
47:44	ADDR3_CFG	If (CPUID.07H.01H:EAX[2:0] > 3)
54:48	Reserved, must be zero.	
55	DisTNT	If (CPUID.14H.00H:EBX[8] = 1)
56	InjectPsbPmiOnEnable	If (CPUID.07H.01H:EBX[6] = 1)
63:57	Reserved, must be zero.	
Register Address: 571H, 1393		IA32_RTIT_STATUS
Tracing Status Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
0	FilterEn (writes ignored).	If (CPUID.07H.00H:EBX[2] = 1)
1	ContexEn (writes ignored).	
2	TriggerEn (writes ignored).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
3	Reserved.	
4	Error	
5	Stopped	
6	PendPSB	If (CPUID.07H.00H:EBX[6] = 1)
7	PendToPAPMI	If (CPUID.07H.00H:EBX[6] = 1)
31:8	Reserved, must be zero.	
48:32	PacketByteCnt	If (CPUID.07H.00H:EBX[1] > 3)
63:49	Reserved.	
Register Address: 572H, 1394		IA32_RTIT_CR3_MATCH
Trace Filter CR3 Match Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
4:0	Reserved.	
63:5	CR3[63:5] value to match.	
Register Address: 580H, 1408		IA32_RTIT_ADDR0_A
Region 0 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 0)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 581H, 1409		IA32_RTIT_ADDR0_B
Region 0 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 0)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 582H, 1410		IA32_RTIT_ADDR1_A
Region 1 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 1)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 583H, 1411		IA32_RTIT_ADDR1_B
Region 1 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 1)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 584H, 1412		IA32_RTIT_ADDR2_A
Region 2 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 2)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 585H, 1413		IA32_RTIT_ADDR2_B
Region 2 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 2)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 586H, 1414		IA32_RTIT_ADDR3_A

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Region 3 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 3)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 587H, 1415		IA32_RTIT_ADDR3_B
Region 3 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 3)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 600H, 1536		IA32_DS_AREA
DS Save Area (R/W) Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 22.6.3.4, “Debug Store (DS) Mechanism.”		If (CPUID.01H:EDX.DS[21] = 1)
63:0	The linear address of the first byte of the DS buffer management area, if IA-32e mode is active.	
31:0	The linear address of the first byte of the DS buffer management area, if not in IA-32e mode.	
63:32	Reserved if not in IA-32e mode.	
Register Address: 6A0H, 1696		IA32_U_CET
Configure User Mode CET (R/W)		Bits 1:0 are defined if CPUID.07H.00H:ECX.CET_SS[7] = 1. Bits 5:2 and bits 63:10 are defined if CPUID.07H.00H:EDX.CET_IBT[20] = 1.
0	SH_STK_EN: When set to 1, enable shadow stacks at CPL3.	
1	WR_SHSTK_EN: When set to 1, enables the WRSSD/WRSSQ instructions.	
2	ENDBR_EN: When set to 1, enables indirect branch tracking.	
3	LEG_IW_EN: Enable legacy compatibility treatment for indirect branch tracking.	
4	NO_TRACK_EN: When set to 1, enables use of no-track prefix for indirect branch tracking.	
5	SUPPRESS_DIS: When set to 1, disables suppression of CET indirect branch tracking on legacy compatibility.	
9:6	Reserved; must be zero.	
10	SUPPRESS: When set to 1, indirect branch tracking is suppressed. This bit can be written to 1 only if TRACKER is written as IDLE.	
11	TRACKER: Value of the indirect branch tracking state machine. Values: IDLE (0), WAIT_FOR_ENDBRANCH(1).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:12	EB_LEG_BITMAP_BASE: Linear address bits 63:12 of a legacy code page bitmap used for legacy compatibility when indirect branch tracking is enabled. If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are used.	
Register Address: 6A2H, 1698		IA32_S_CET
Configure Supervisor Mode CET (R/W)		See IA32_U_CET (6A0H) for reference; similar format.
Register Address: 6A4H, 1700		IA32_PL0_SSP (also called IA32_FRED_SSP0)
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 0 will check that bit 2 is also 0. FRED event delivery uses this MSR as IA32_FRED_SSP0: if FRED event delivery causes a transition from CPL 3, SSP is loaded from this MSR.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6A5H, 1701		IA32_PL1_SSP
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 1 from a higher privilege level will check that bit 2 is also 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6A6H, 1702		IA32_PL2_SSP
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 2 from a higher privilege level will check that bit 2 is also 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6A7H, 1703		IA32_PL3_SSP
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6A8H, 1704		IA32_INTERRUPT_SSP_TABLE_ADDR
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) This MSR is not present on processors that do not support Intel 64 architecture. This field cannot represent a non-canonical address.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6E0H, 1760		IA32_TSC_DEADLINE
TSC Target of Local APIC's TSC Deadline Mode (R/W)		If CPUID.01H:ECX[24] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:0	REGISTER_VALUE TSC-deadline value.	
Register Address: 6E1H, 1761		IA32_PKRS
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W)		If CPUID.07H:00H:ECX.PKS[31] = 1
31:0	For domain i (i between 0 and 15), bits 2i and 2i+1 contain the AD and WD permissions, respectively.	
63:32	Reserved.	
Register Address: 770H, 1904		IA32_PM_ENABLE
Enable/disable HWP (R/W)		If CPUID.06H:EAX[7] = 1
0	HWP_ENABLE (R/W) Note this bit can only be enabled once from the default value. Once set, writes to the HWP_ENABLE bit are ignored. Only RESET will clear this bit. Default = 0. See Section 17.4.2, "Enabling HWP."	If CPUID.06H:EAX[7] = 1
63:1	Reserved.	
Register Address: 771H, 1905		IA32_HWP_CAPABILITIES
HWP Performance Range Enumeration (R/O)		If CPUID.06H:EAX[7] = 1
7:0	Highest_Performance See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
15:8	Guaranteed_Performance See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
23:16	Most_Efficient_Performance See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
31:24	Lowest_Performance See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
63:32	Reserved.	
Register Address: 772H, 1906		IA32_HWP_REQUEST_PKG
Power Management Control Hints for All Logical Processors in a Package (R/W)		If CPUID.06H:EAX[11] = 1
7:0	Minimum_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
15:8	Maximum_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
23:16	Desired_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
31:24	Energy_Performance_Preference See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1 && CPUID.06H:EAX[10] = 1
41:32	Activity_Window See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1 && CPUID.06H:EAX[9] = 1
63:42	Reserved.	
Register Address: 773H, 1907		IA32_HWP_INTERRUPT
Control HWP Native Interrupts (R/W)		If CPUID.06H:EAX[8] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
0	EN_Guaranteed_Performance_Change See Section 17.4.6, "HWP Notifications."	If CPUID.06H:EAX[8] = 1
1	EN_Excursion_Minimum See Section 17.4.6, "HWP Notifications."	If CPUID.06H:EAX[8] = 1
63:2	Reserved.	
Register Address: 774H, 1908		IA32_HWP_REQUEST
Power Management Control Hints to a Logical Processor (R/W)		If CPUID.06H:EAX[7] = 1
7:0	Minimum_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
15:8	Maximum_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
23:16	Desired_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
31:24	Energy_Performance_Preference See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[10] = 1
41:32	Activity_Window See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[9] = 1
42	Package_Control See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[11] = 1
63:43	Reserved.	
Register Address: 775H, 1909		IA32_PECI_HWP_REQUEST_INFO
IA32_PECI_HWP_REQUEST_INFO		
7:0	Minimum Performance (MINIMUM_PERFORMANCE): Used by OS to read the latest value of Peci minimum performance input. Default value is 0.	
15:8	Maximum Performance (MAXIMUM_PERFORMANCE): Used by OS to read the latest value of Peci maximum performance input. Default value is 0.	
23:16	Reserved.	
31:24	Energy Performance Preference (ENERGY_PERFORMANCE_PREFERENCE): Used by OS to read the latest value of Peci Energy Performance Preference input. Default value is 0.	
59:32	Reserved.	
60	EPP Peci Override (EPP_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Energy Performance Preference input. If set to '1', Peci is overriding the Energy Performance Preference input. If clear (0), OS has control over Energy Performance Preference input. Default value is 0.	
61	Reserved.	
62	Max Peci Override (MAX_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Maximum Performance input. If set to '1', Peci is overriding the Maximum Performance input. If clear (0), OS has control over Maximum Performance input. Default value is 0.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63	Min PECL Override (MIN_PECI_OVERRIDE): Indicates whether PECL is currently overriding the Minimum Performance input. If set to '1', PECL is overriding the Minimum Performance input. If clear (0), OS has control over Minimum Performance input. Default value is 0.	
Register Address: 776H, 1910		IA32_HWP_CTL
IA32_HWP_CTL		If CPUID.06H:EAX[22] = 1
0	PKG_CTL_POLARITY Defines which HWP Request MSR is used whether Thread level or package level. When package MSR is used, the thread MSR valid bits define which thread MSR fields override the package. Default value is 0.	If CPUID.06H:EAX[22] = 1
63:1	Reserved.	
Register Address: 777H, 1911		IA32_HWP_STATUS
Log bits indicating changes to Guaranteed & excursions to Minimum (R/W)		If CPUID.06H:EAX[7] = 1
0	Guaranteed_Performance_Change (R/WCO) See Section 17.4.5, "HWP Feedback."	If CPUID.06H:EAX[7] = 1
1	Reserved.	
2	Excursion_To_Minimum (R/WCO) See Section 17.4.5, "HWP Feedback."	If CPUID.06H:EAX[7] = 1
63:3	Reserved.	
Register Address: 7A3H, 1955		IA32_MCU_EXT_SERVICE
MCU Extended Service (R/O)		If IA32_ARCH_CAPABILITIES[22] = 1
3:0	ALLOWED_PERIODS Value indicates the allowed periods for extended servicing. Value x means that all extended servicing periods are allowed till period x.	
63:4	Reserved.	
Register Address: 7A4H, 1956		IA32_MCU_ROLLBACK_MIN_ID
Minimal MCU Revision ID (R/O) Minimal MCU Revision ID that software can rollback to per boot.		If IA32_MCU_ENUMERATION[3] = 1
31:0	REVISION_ID Minimal MCU revision ID for rollback.	
63:32	Reserved for future use.	
Register Address: 7A5H, 1957		IA32_MCU_STAGING_MBOX_ADDR
IA32_MCU_STAGING_MBOX_ADDR (R/O) Reports MMIO address of MCU staging DOE mailbox.		
63:0	ADDR MMIO address base of MCU staging DOE mailbox.	
Register Address: 7B0H, 1968		IA32_ROLLBACK_SIGN_ID_0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Rollback ID 0 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B1H, 1969		IA32_ROLLBACK_SIGN_ID_1
Rollback ID 1 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B2H, 1970		IA32_ROLLBACK_SIGN_ID_2
Rollback ID 2 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B3H, 1971		IA32_ROLLBACK_SIGN_ID_3
Rollback ID 3 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B4H, 1972		IA32_ROLLBACK_SIGN_ID_4
Rollback ID 4 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 7B5H, 1973		IA32_ROLLBACK_SIGN_ID_5
Rollback ID 5 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B6H, 1974		IA32_ROLLBACK_SIGN_ID_6
Rollback ID 6 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B7H, 1975		IA32_ROLLBACK_SIGN_ID_7
Rollback ID 7 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B8H, 1976		IA32_ROLLBACK_SIGN_ID_8
Rollback ID 8 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B9H, 1977		IA32_ROLLBACK_SIGN_ID_9
Rollback ID 9 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:48	Reserved.	
Register Address: 7BAH, 1978		IA32_ROLLBACK_SIGN_ID_10
Rollback ID 10 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BBH, 1979		IA32_ROLLBACK_SIGN_ID_11
Rollback ID 11 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BCH, 1980		IA32_ROLLBACK_SIGN_ID_12
Rollback ID 12 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BDH, 1981		IA32_ROLLBACK_SIGN_ID_13
Rollback ID 13 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BEH, 1982		IA32_ROLLBACK_SIGN_ID_14
Rollback ID 14 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BFH, 1983		IA32_ROLLBACK_SIGN_ID_15
Rollback ID 15 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 802H, 2050		IA32_X2APIC_APICID
x2APIC ID Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 803H, 2051		IA32_X2APIC_VERSION
x2APIC Version Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 808H, 2056		IA32_X2APIC_TPR
x2APIC Task Priority Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80AH, 2058		IA32_X2APIC_PPR
x2APIC Processor Priority Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80BH, 2059		IA32_X2APIC_EOI
x2APIC EOI Register (W/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80DH, 2061		IA32_X2APIC_LDR
x2APIC Logical Destination Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80FH, 2063		IA32_X2APIC_SIVR
x2APIC Spurious Interrupt Vector Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 810H, 2064		IA32_X2APIC_ISR0
x2APIC In-Service Register Bits 31:0 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 811H, 2065		IA32_X2APIC_ISR1
x2APIC In-Service Register Bits 63:32 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 812H, 2066		IA32_X2APIC_ISR2
x2APIC In-Service Register Bits 95:64 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits 127:96 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits 159:128 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits 191:160 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits 223:192 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits 255:224 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits 31:0 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits 63:32 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits 95:64 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits 127:96 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits 159:128 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits 191:160 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits 223:192 (R/O)	If (CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1)	
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits 255:224 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 820H, 2080	IA32_X2APIC_IRRO	
x2APIC Interrupt Request Register Bits 31:0 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request Register Bits 63:32 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits 95:64 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits 127:96 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits 159:128 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits 191:160 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits 223:192 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits 255:224 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 836H, 2102		IA32_X2APIC_LVT_LINT1
x2APIC LVT LINT1 Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 837H, 2103		IA32_X2APIC_LVT_ERROR
x2APIC LVT Error Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 838H, 2104		IA32_X2APIC_INIT_COUNT
x2APIC Initial Count Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 839H, 2105		IA32_X2APIC_CUR_COUNT
x2APIC Current Count Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 83EH, 2110		IA32_X2APIC_DIV_CONF
x2APIC Divide Configuration Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 83FH, 2111		IA32_X2APIC_SELF_IPI
x2APIC Self IPI Register (W/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 981H, 2433		IA32_TME_CAPABILITY
Memory Encryption Capability MSR		If CPUID.07H.00H:ECX[13] = 1
0	Support for AES-XTS 128-bit encryption algorithm. (NIST standard)	
1	Support for AES-XTS 128-bit encryption with integrity algorithm.	
2	Support for AES-XTS 256-bit encryption algorithm.	
3	Support for AES-XTS 256-bit encryption with integrity algorithm.	
27:4	Reserved.	
28	Non-zero KeyIDs must be SEAM-private.	If set, any write to IA32_TME_ACTIVATE must write identical values to MK_TME_KEYID_BITS and TDX_RESERVED_KEYID_BITS.
29	Reserved.	
30	SUPPORT_IA32_TME_CLEAR_SAVED_KEY Support for the IA32_TME_CLEAR_SAVED_KEY MSR.	
31	TME encryption bypass supported.	
35:32	MK_TME_MAX_KEYID_BITS Number of bits which can be allocated for usage as key identifiers for multi-key memory encryption. 4 bits allow for a maximum value of 15, which could address 32K keys. Zero if TME-MK is not supported.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
50:36	MK_TME_MAX_KEYS Indicates the maximum number of keys which are available for usage. This value may not be a power of 2. KeyID 0 is specially reserved and is not accounted for in this field.	
63:51	Reserved.	
Register Address: 982H, 2434		IA32_TME_ACTIVATE
Memory Encryption Activation MSR This MSR is used to lock the MSRs listed below. Any write to the following MSRs will be ignored after they are locked. The lock is reset when CPU is reset. ▪ IA32_TME_ACTIVATE ▪ IA32_TME_EXCLUDE_MASK ▪ IA32_TME_EXCLUDE_BASE Note that IA32_TME_EXCLUDE_MASK and IA32_TME_EXCLUDE_BASE must be configured before IA32_TME_ACTIVATE.		If CPUID.07H.00H:ECX[13] = 1
0	Lock R/O – Will be set upon successful WRMSR (or first SMI); written value ignored.	
1	Hardware Encryption Enable This bit also enables TME-MK; TME-MK cannot be enabled without enabling encryption hardware. TME is enabled depending on the TME Encryption Bypass Enable bit (bit 31).	
2	Key Select 0: Create a new TME key (expected cold/warm boot). 1: Restore the TME key from storage (expected when resume from standby).	
3	Save TME Key for Standby Save key into storage to be used when resume from standby. Note: This may not be supported in all processors.	
7:4	TME Policy/Encryption Algorithm Only algorithms enumerated in IA32_TME_CAPABILITY MSR are allowed. Any other values are invalid and will result in #GP. Additionally, any algorithm that supports integrity checking is not allowed to be used for TME even if it is listed as allowed in the IA32_TME_CAPABILITY MSR and will result in #GP.	
30:8	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31	TME Encryption Bypass Enable When encryption hardware is enabled: - Total Memory Encryption is enabled using a CPU generated ephemeral key based on a hardware random number generator when this bit is set to 0. - Total Memory Encryption is bypassed (no encryption/decryption for KeyID0) when this bit is set to 1. On some processors, bypassing TME encryption can provide performance benefits to accesses made with KeyID 0 by avoiding the latency of decryption or encryption and decryption. Software must inspect Hardware Encryption Enable (bit 1) and TME Encryption Bypass Enable (bit 31) to determine if TME encryption is enabled.	
35:32	MK_TME_KEYID_BITS Reserved if TME-MK is not enumerated, otherwise: The number of key identifier bits to allocate to TME-MK usage. Similar to enumeration, this is an encoded value. Writing a value greater than MK_TME_MAX_KEYID_BITS will result in #GP. Writing a non-zero value to this field will #GP if bit 1 of EAX (Hardware Encryption Enable) is not also set to 1, as encryption hardware must be enabled to use TME-MK. Example: To support 255 keys, this field would be set to a value of 8.	
39:36	TDX_RESERVED_KEYID_BITS The number of key identifier bits to allocate to TDX usage, which are allocated from the most significant bit downward. Writing a value greater than MK_TME_KEYID_BITS will result in a #GP. Note: These bits are a subset of the overall KeyID bits which are declared by MK_TME_MAX_KEYID_BITS.	If IA32_TME_CAPABILITY[28] = 1, this value must be the same as that of MK_TME_KEYID_BITS.
47:40	Reserved.	
63:48	MK_TME_CRYPT0_ALGS Reserved if TME-MK is not enumerated, otherwise: Bit 48: AES-XTS 128. Bit 49: AES-XTS 128 with integrity. Bit 50: AES-XTS 256. Bit 51: AES-XTS-256 with integrity. Bit 63:52: Reserved (#GP) Bitmask for BIOS to set which encryption algorithms are allowed for TME-MK, would be later enforced by the key loading ISA ('1' = allowed).	
Register Address: 983H, 2435		IA32_TME_EXCLUDE_MASK
Memory Encryption Exclude Mask The IA32_TME_EXCLUDE_MASK MSR must define a contiguous region. WRMSR will #GP if the TMEEMASK field does not specify a contiguous region. This MSR is locked by the IA32_TME_ACTIVATE MSR. If lock=1, then WRMSR to IA32_TME_EXCLUDE_MASK/IA32_TME_EXCLUDE_BASE MSRs will result in #GP.		If CPUID.07H.00H:ECX[13] = 1
10:0	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
11	Enable: When set to '1', then TME_EXCLUDE_BASE and TME_EXCLUDE_MASK are used to define an exclusion region for TME/TME-MK (for KeyID=0).	
MAXPHYADDR-1:12	TMEEMASK: This field indicates the bits that must match TMEEBASE in order to qualify as a TME/TME-MK (for KeyID=0) exclusion memory range access.	
63:MAXPHYADDR	Reserved; must be zero.	
Register Address: 984H, 2436		IA32_TME_EXCLUDE_BASE
Memory Encryption Exclude Base Note: Writing '1' into bits greater than the max supported physical size will result in #GP. This MSRs is locked by the IA32_TME_ACTIVATE MSR. If lock=1, then WRMSR to IA32_TME_EXCLUDE_MASK/IA32_TME_EXCLUDE_BASE MSRs will result in #GP.		If CPUID.07H.00H:ECX[13] = 1
11:0	Reserved.	
MAXPHYADDR-1:12	TMEEBASE: Base physical address to be excluded for TME/TME-MK (for KeyID=0) encryption.	
63:MAXPHYADDR	Reserved; must be zero.	
Register Address: 985H, 2437		IA32_UINTR_RR
User Interrupt Request Register (R/W)		If CPUID.07H.01H:EDX[13] = 1
63:0	UIRR Bitmap of requested user interrupt vectors.	
Register Address: 986H, 2438		IA32_UINTR_HANDLER
User Interrupt Handler Address (R/W)		If CPUID.07H.01H:EDX[13] = 1
63:0	UIHANDLER User interrupt handler linear address.	
Register Address: 987H, 2439		IA32_UINTR_STACKADJUST
User Interrupt Stack Adjustment (R/W)		If CPUID.07H.01H:EDX[13] = 1
0	LOAD_RSP User interrupt stack mode.	
2:1	Reserved.	
63:3	STACK_ADJUST Stack adjust value.	
Register Address: 988H, 2440		IA32_UINTR_MISC
User-Interrupt Target-Table Size and Notification Vector (R/W)		If CPUID.07H.01H:EDX[13] = 1
31:0	UITTSZ The highest index of a valid entry in the user-interrupt target table. Valid entries are indices 0..UITTSZ (inclusive).	
39:32	UINV User-interrupt notification vector.	
63:40	Reserved.	
Register Address: 989H, 2441		IA32_UINTR_PD
User Interrupt PID Address (R/W)		If CPUID.07H.01H:EDX[13] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
5:0	Reserved.	
63:6	UPIDADDR User-interrupt notification processing accesses a UPID at this linear address.	
Register Address: 98AH, 2442		IA32_UINTR_TT
User-Interrupt Target Table (R/W)		If CPUID.07H.01H:EDX[13] = 1
0	SENDUIPI_ENABLE User-interrupt target table is valid.	
3:1	Reserved.	
63:4	UITTADDR User-interrupt target table base linear address.	
Register Address: 990H, 2448		IA32_COPY_STATUS ⁴
Status of Most Recent Platform to Local or Local to Platform Copies (R/O)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	IWKEY_COPY_SUCCESSFUL Status of most recent copy to or from IwKeyBackup.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	
Register Address: 991H, 2449		IA32_IWKEYBACKUP_STATUS ⁵
Information about IwKeyBackup Register (R/O)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	Backup/Restore Valid Cleared when a write to IwKeyBackup is initiated, and then set when the latest write of IwKeyBackup has been written to storage that persists across S3/S4 sleep state. If S3/S4 is entered between when an IwKeyBackup write occurs and when this bit is set, then IwKeyBackup may not be recovered after S3/S4 exit. During S3/S4 sleep state exit (system wake up), this bit is cleared. It is set again when IwKeyBackup is restored from persistent storage and thus available to be copied to IwKey using IA32_COPY_PLATFORM_TO_LOCAL MSR. Another write to IwKeyBackup (via IA32_COPY_LOCAL_TO_PLATFORM MSR) may fail if a previous write has not yet set this bit.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
1	Reserved.	
2	Backup Key Storage Read/Write Error Updated prior to backup/restore valid being set. Set when an error is encountered while backing up or restoring a key to persistent storage.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
3	IwKeyBackup Consumed Set after the previous backup operation has been consumed by the platform. This does not indicate that the system is ready for a second IwKeyBackup write as the previous IwKeyBackup write may still need to set Backup/restore valid.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:4	Reserved.	
Register Address: 9F1H, 2545		IA32_TSE_CAPABILITY
IA32_TSE_CAPABILITY		If CPUID.07H.01H:EBX.PBNDKB[1] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
15:0	Supported encryption algorithms.	
23:16	TSE engine key sources supported.	
35:24	Reserved.	
50:36	TSE_MAX_KEYS. Indicates the maximum number of keys available.	
63:51	Reserved.	
Register Address: 9FBH, 2555		IA32_TME_CLEAR_SAVED_KEY
IA32_TME_CLEAR_SAVED_KEY (W/O)		
0	TME_CLEAR_SAVED_KEY Clear saved TME keys.	
63:1	Reserved.	
Register Address: C80H, 3200		IA32_DEBUG_INTERFACE
Silicon Debug Feature Control (R/W)		If CPUID.01H:ECX[11] = 1
0	Enable (R/W) BIOS set 1 to enable Silicon debug features. Default is 0.	If CPUID.01H:ECX[11] = 1
29:1	Reserved.	
30	Lock (R/W): If 1, locks any further change to the MSR. The lock bit is set automatically on the first SMI assertion even if not explicitly set by BIOS. Default is 0.	If CPUID.01H:ECX[11] = 1
31	Debug Occurred (R/O): This “sticky bit” is set by hardware to indicate the status of bit 0. Default is 0.	If CPUID.01H:ECX[11] = 1
63:32	Reserved.	
Register Address: C81H, 3201		IA32_L3_QOS_CFG
L3 QOS Configuration (R/W)		If (CPUID.10H.01H:ECX[2] = 1)
0	Enable (R/W) Set 1 to enable L3 CAT masks and CLOS to operate in Code and Data Prioritization (CDP) mode.	
63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
Register Address: C82H, 3202		IA32_L2_QOS_CFG
L2 QOS Configuration (R/W)		If (CPUID.10H.02H:ECX[2] = 1)
0	Enable (R/W) Set 1 to enable L2 CAT masks and CLOS to operate in Code and Data Prioritization (CDP) mode.	
63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
Register Address: C83H, 3203		IA32_L3_IO_QOS_CFG
L3 I/O QOS Configuration (R/W) This MSR is used to enable the I/O RDT features.		If (CPUID.0FH.01H:EAX[10:9] = 1)
0	L3 I/O RDT Allocation Enable.	
1	L3 I/O RDT Monitoring Enable.	
63:2	Reserved.	
Register Address: C88H, 3208		IA32_RESOURCE_PRIORITY

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Thread Scope Resource Priority Enable (R/W) Cleared at system reset.		If CPUID.10H.06H:EAX[0] = 1
0	ENABLE When set for all logical processors, enables the Resource Priority feature.	
63:1	Reserved.	
Register Address: C89H, 3209		IA32_RESOURCE_PRIORITY_PKG
Package Scope Resource Priority Enable (R/W) Cleared at system reset.		If CPUID.10H.06H:EAX[1] = 1
0	ENABLE When set for each package, enables Resource Priority feature.	
63:1	Reserved.	
Register Address: C8DH, 3213		IA32_QM_EVTSEL
Monitoring Event Select Register (R/W)		If (CPUID.07H.00H:EBX[12] = 1)
7:0	Event ID: ID of a supported monitoring event to report via IA32_QM_CTR.	
31:8	Reserved.	
N+31:32	Resource Monitoring ID: ID for monitoring hardware to report monitored data via IA32_QM_CTR.	N = Ceil (Log ₂ (CPUID.0FH.00H:EBX[31:0] + 1))
63:N+32	Reserved.	
Register Address: C8EH, 3214		IA32_QM_CTR
Monitoring Counter Register (R/O)		If (CPUID.07H.00H:EBX[12] = 1)
61:0	Resource Monitored Data.	
62	Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
63	Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
Register Address: C8FH, 3215		IA32_PQR_ASSOC
Resource Association Register (R/W)		If ((CPUID.07H.00H:EBX[12] = 1) or (CPUID.07H.00H:EBX[15] = 1))
N-1:0	Resource Monitoring ID (R/W): ID for monitoring hardware to track internal operation, e.g., memory access.	N = Ceil (Log ₂ (CPUID.0FH.00H:EBX[31:0] + 1))
31:N	Reserved.	
M:32	CLOS (R/W): The class of service (CLOS) to enforce (on writes); returns the current CLOS when read.	CLOS_MAX = max of all valid RDT Allocation ResID.EDX[15:0] sub-leaves CLOS_WIDTH = Ceil(Log ₂ (CLOS_MAX + 1)) M = (32 + CLOS_WIDTH - 1)
63:M+1+32	Reserved.	
Register Address: C90H–D8FH, 3216–3471		Reserved MSR Address Space for CAT Mask Registers
See Section 20.19.4.1, “Enumeration and Detection Support of Cache Allocation Technology.”		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: C90H, 3216		IA32_L3_MASK_0
L3 CAT Mask for COS0 (R/W)		If (CPUID.10H.00H:EBX[1] != 0)
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: C90H+n, 3216+n		IA32_L3_MASK_n
L3 CAT Mask for COSn (R/W)		n = CPUID.10H.01H:EDX[15:0]
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D10H–D4FH, 3344–3407		Reserved MSR Address Space for L2 CAT Mask Registers
See Section 20.19.4.1, “Enumeration and Detection Support of Cache Allocation Technology.”		
Register Address: D10H, 3344		IA32_L2_MASK_0
L2 CAT Mask for COS0 (R/W)		If (CPUID.10H.00H:EBX[2] != 0)
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D10H+n, 3344+n		IA32_L2_MASK_n
L2 CAT Mask for COSn (R/W)		n = CPUID.10H.02H:EDX[15:0]
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D18H, 3352		IA32_L2_MASK_8
L2 CAT Mask for COS8 (R/W)		
15:0	WAY_MASK Capacity Bit Mask. Available ways vectors for class of service of IA core. '1' in bit indicates allocation to the way is allowed. '0' indicates allocation to the way is not allowed.	
63:16	Reserved.	
Register Address: D19H, 3353		IA32_L2_MASK_9
L2 CAT Mask for COS9 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1AH, 3354		IA32_L2_MASK_10
L2 CAT Mask for COS10 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1BH, 3355		IA32_L2_MASK_11
L2 CAT Mask for COS11 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1CH, 3356		IA32_L2_MASK_12
L2 CAT Mask for COS12 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1DH, 3357		IA32_L2_MASK_13

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
L2 CAT Mask for COS13 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1EH, 3358		IA32_L2_MASK_14
L2 CAT Mask for COS14 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1FH, 3359		IA32_L2_MASK_15
L2 CAT Mask for COS15 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D50H, 3408		IA32_L2_QOS_EXT_BW_THRTL_0
IA32_L2_QOS_EXT_BW_THRTL_0 (R/W) Memory Bandwidth enforcement for COS0.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 0
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D51H, 3409		IA32_L2_QOS_EXT_BW_THRTL_1
IA32_L2_QOS_EXT_BW_THRTL_1 (R/W) Memory Bandwidth enforcement for COS1.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 1
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D52H, 3410		IA32_L2_QOS_EXT_BW_THRTL_2
IA32_L2_QOS_EXT_BW_THRTL_2 (R/W) Memory Bandwidth enforcement for COS2.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 2
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D53H, 3411		IA32_L2_QOS_EXT_BW_THRTL_3
IA32_L2_QOS_EXT_BW_THRTL_3 (R/W) Memory Bandwidth enforcement for COS3.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 3
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D54H, 3412		IA32_L2_QOS_EXT_BW_THRTL_4
IA32_L2_QOS_EXT_BW_THRTL_4 (R/W) Memory Bandwidth enforcement for COS4.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 4
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D55H, 3413		IA32_L2_QOS_EXT_BW_THRTL_5

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_L2_QOS_EXT_BW_THRTL_5 (R/W) Memory Bandwidth enforcement for COS5.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 5
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D56H, 3414		IA32_L2_QOS_EXT_BW_THRTL_6
IA32_L2_QOS_EXT_BW_THRTL_6 (R/W) Memory Bandwidth enforcement for COS6.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 6
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D57H, 3415		IA32_L2_QOS_EXT_BW_THRTL_7
IA32_L2_QOS_EXT_BW_THRTL_7 (R/W) Memory Bandwidth enforcement for COS7.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 7
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D58H, 3416		IA32_L2_QOS_EXT_BW_THRTL_8
IA32_L2_QOS_EXT_BW_THRTL_8 (R/W) Memory Bandwidth enforcement for COS8.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 8
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D59H, 3417		IA32_L2_QOS_EXT_BW_THRTL_9
IA32_L2_QOS_EXT_BW_THRTL_9 (R/W) Memory Bandwidth enforcement for COS9.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 9
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5AH, 3418		IA32_L2_QOS_EXT_BW_THRTL_10
IA32_L2_QOS_EXT_BW_THRTL_10 (R/W) Memory Bandwidth enforcement for COS10.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 10
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5BH, 3419		IA32_L2_QOS_EXT_BW_THRTL_11
IA32_L2_QOS_EXT_BW_THRTL_11 (R/W) Memory Bandwidth enforcement for COS11.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 11

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5CH, 3420		IA32_L2_QOS_EXT_BW_THRTL_12
IA32_L2_QOS_EXT_BW_THRTL_12 (R/W) Memory Bandwidth enforcement for COS12.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 12
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5DH, 3421		IA32_L2_QOS_EXT_BW_THRTL_13
IA32_L2_QOS_EXT_BW_THRTL_13 (R/W) Memory Bandwidth enforcement for COS13.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 13
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5EH, 3422		IA32_L2_QOS_EXT_BW_THRTL_14
IA32_L2_QOS_EXT_BW_THRTL_14 (R/W) Memory Bandwidth enforcement for COS14.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 14
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D90H, 3472		IA32_BNDCFGS
Supervisor State of MPX Configuration (R/W)		If (CPUID.07H.00H:EBX[14] = 1)
0	EN: Enable Intel MPX in supervisor mode.	
1	BNDPRESERVE: Preserve the bounds registers for near branch instructions in the absence of the BND prefix.	
11:2	Reserved, must be zero.	
63:12	Base Address of Bound Directory.	
Register Address: D91H, 3473		IA32_COPY_LOCAL_TO_PLATFORM ⁵
Copy Local State to Platform State (W)		IF ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	lwKeyBackup Copy lwKey to lwKeyBackup.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	
Register Address: D92H, 3474		IA32_COPY_PLATFORM_TO_LOCAL ⁵
Copy Platform State to Local State (W)		IF ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	lwKeyBackup Copy lwKeyBackup to lwKey.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: D93H, 3475		IA32_PASID	
Process Address Space Identifier. (R/W)			
19:0	Process address space identifier (PASID). Specifies the PASID of the currently running software thread.		
30:20	Reserved.		
31	Valid. Execution of ENQCMD causes a #GP if this bit is clear.		
63:32	Reserved.		
Register Address: DA0H, 3488		IA32_XSS	
Extended Supervisor State Mask (R/W)			If (CPUID.0DH:01H:EAX[3] = 1
7:0	Reserved.		
8	PT State (R/W)		
9	Reserved.		
10	PASID State (R/W)		
11	CET_U State (R/W)		
12	CET_S State (R/W)		
13	HDC State (R/W)		
14	UINTR State (R/W)		
15	LBR State (R/W)		
16	HWP State (R/W)		
63:17	Reserved.		
Register Address: DB0H, 3504		IA32_PKG_HDC_CTL	
Package Level Enable/Disable HDC (R/W)			If CPUID.06H:EAX[13] = 1
0	HDC_Pkg_Enable (R/W) Force HDC idling or wake up HDC-idled logical processors in the package. See Section 17.5.2, "Package level Enabling HDC."		If CPUID.06H:EAX[13] = 1
63:1	Reserved.		
Register Address: DB1H, 3505		IA32_PM_CTL1	
Enable/Disable the HDC Thread Level Activity (R/W)			If CPUID.06H:EAX[13] = 1
0	SDC_ALLOWED (R/W) Set this bit to allow this thread to be forced into HDC idle state. Clearing this bit blocks HDC-enter (HW) request. Default value: 1. See Section 17.5.3.		If CPUID.06H:EAX[13] = 1
63:1	Reserved.		
Register Address: DB2H, 3506		IA32_THREAD_STALL	
Per-Logical_Processor_ID HDC Idle Residency (R/O)			If CPUID.06H:EAX[13] = 1
63:0	Stall_Cycle_Cnt (R/W) Stalled cycles due to HDC forced idle on this logical processor. See Section 17.5.4.1.		If CPUID.06H:EAX[13] = 1
Register Address: E00H, 3584		IA32_QOS_CORE_BW_THRTL_0	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
CBA Levels Based on COS for Bandwidth Throttling (R/W)		CPUID.10H.00H:EBX[5] = 1
3:0	COS0_LEVEL CBA Level for COS[0]. Levels are programmed from 0 to 15.	
7:4	Reserved.	
11:8	COS1_LEVEL CBA Level for COS[1]. Levels are programmed from 0 to 15.	
15:12	Reserved.	
19:16	COS2_LEVEL CBA Level for COS[2]. Levels are programmed from 0 to 15.	
25:20	Reserved.	
27:24	COS3_LEVEL CBA Level for COS[3]. Levels are programmed from 0 to 15.	
31:28	Reserved.	
35:32	COS4_LEVEL CBA Level for COS[4]. Levels are programmed from 0 to 15.	
39:36	Reserved.	
43:40	COS5_LEVEL CBA Level for COS[5]. Levels are programmed from 0 to 15.	
47:44	Reserved.	
51:48	COS6_LEVEL CBA Level for COS[6]. Levels are programmed from 0 to 15.	
Register Address: E01H, 3585		IA32_QOS_CORE_BW_THRTL_1
CBA Levels Based on COS for Bandwidth Throttling (R/W)		CPUID.10H.00H:EBX[5] = 1
3:0	COS8_LEVEL CBA Level for COS[8]. Levels are programmed from 0 to 15.	
7:4	Reserved.	
11:8	COS9_LEVEL CBA Level for COS[9]. Levels are programmed from 0 to 15.	
15:12	Reserved.	
19:16	COS10_LEVEL CBA Level for COS[10]. Levels are programmed from 0 to 15.	
25:20	Reserved.	
27:24	COS11_LEVEL CBA Level for COS[11]. Levels are programmed from 0 to 15.	
31:28	Reserved.	
35:32	COS12_LEVEL CBA Level for COS[12]. Levels are programmed from 0 to 15.	
39:36	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
43:40	COS13_LEVEL CBA Level for COS[13]. Levels are programmed from 0 to 15.	
47:44	Reserved.	
51:48	COS14_LEVEL CBA Level for COS[14]. Levels are programmed from 0 to 15.	
55:50	Reserved.	
59:56	COS15_LEVEL CBA Level for COS[15]. Levels are programmed from 0 to 15.	
63:60	Reserved	
Register Address: 1200H—121FH, 4608—4639		IA32_LBR_x_INFO
Last Branch Record Entry X Info Register (R/W) An attempt to read or write IA32_LBR_x_INFO such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		
15:0	CYC_CNT The elapsed CPU cycles (saturating) since the last LBR was recorded. See Section 18.1.3.3.	Reset Value: 0
55:16	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
59:56	BR_TYPE The branch type recorded by this LBR. Encodings: 0000B: COND 0001B: JMP Indirect 0010B: JMP Direct 0011B: CALL Indirect 0100B: CALL Direct 0101B: RET 011xB: Reserved 1xxxB: Other Branch	Reset Value: 0
60	CYC_CNT_VALID CYC_CNT value is valid. See Section 21.1.3.3.	Reset Value: 0
61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
63	MISPRED The recorded branch direction (conditional branch) or target (indirect branch) was mispredicted.	Reset Value: 0
Register Address: 1400H, 5120		IA32_SEAMRR_BASE

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
SEAM Memory Range Register for TDX - Base Address (R/W)		IA32_MTRRCAP.SEAMRR[15] = 1
2:0	Reserved.	
3	CONFIGURED When set to 1, the SEAM range is configured.	
24:4	Reserved.	
M-1:25	BASE SEAM Range Register BASE address.	M is the value enumerated by CPUID.80000008H:EAX[7:0] Bits M-1:M-k must be 0, where k = IA32_TME_ACTIVATE.MK_TME_KE YID_BITS
63:M	Reserved.	
Register Address: 1401H, 5121		IA32_SEAMRR_MASK
SEAM Memory Range Register - Address Mask (R/W)		IA32_MTRRCAP.SEAMRR[15] = 1
9:0	Reserved.	
10	LOCK	When the LOCK bit is set, the IA32_SEAMRR_BASE/MASK registers are non-writable.
11	VALID Returns 1 when SEAM range protections are active	Read-only. Must be written with 0.
24:12	Reserved.	
M-1:25	MASK Mask value for SEAMRR matching. Lowest granularity is 32MB.	M is the value enumerated by CPUID.80000008H:EAX[7:0]
63:M	Reserved.	
Register Address: 1406H, 5126		IA32_MCU_CONTROL
MCU Control (R/W) Controls the behavior of the Microcode Update Trigger MSR, IA32_BIOS_UPDT_TRIG.		If CPUID.07H.00H:EDX[29] = 1 && IA32_ARCH_CAPABILITIES.MCU_C ONTROL = 1
0	LOCK Once set, further writes to this MSR will cause a #GP(0) fault. Bypassed during SMM if EN_SMM_BYPASS (bit 2) is set.	
1	DIS_MCU_LOAD If this bit is set on a given logical processor, then any subsequent attempts to load a microcode update by that logical processor will be silently dropped (WRMSR 0x79 has no effect).	
2	EN_SMM_BYPASS If set, then writes to IA32_MCU_CONTROL are allowed during SMM regardless of the LOCK bit. This enables BIOS to Opt-In to the SMM Bypass functionality.	
63:3	Reserved.	
Register Address: 14CEH, 5326		IA32_LBR_CTL
Last Branch Record Enabling and Configuration Register (R/W)		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
0	LBREn When set, enables LBR recording.	Reset Value: 0
1	OS When set, allows LBR recording when CPL == 0.	Reset Value: 0
2	USR When set, allows LBR recording when CPL != 0.	Reset Value: 0
3	CALL_STACK When set, records branches in call-stack mode. See Section 21.1.2.4.	Reset Value: 0
15:4	Reserved.	Reset Value: 0
16	COND When set, records taken conditional branches. See Section 21.1.2.3.	
17	NEAR_REL_JMP When set, records near relative JMPs. See Section 21.1.2.3.	
18	NEAR_IND_JMP When set, records near indirect JMPs. See Section 21.1.2.3.	
19	NEAR_REL_CALL When set, records near relative CALLs. See Section 21.1.2.3.	
20	NEAR_IND_CALL When set, records near indirect CALLs. See Section 21.1.2.3.	
21	NEAR_RET When set, records near RETs. See Section 21.1.2.3.	
22	OTHER_BRANCH When set, records other branches. See Section 21.1.2.3.	
63:23	Reserved.	
Register Address: 14CFH, 5327		IA32_LBR_DEPTH
Last Branch Record Maximum Stack Depth Register (R/W)		
N:0	DEPTH The number of LBRs to be used for recording. Supported values are indicated by the bitmap in CPUID.1CH.00H:EAX[7:0]. The reset value will match the maximum supported by the CPU. Writes of unsupported values will #GP fault.	Reset Value: Varies
63:N+1	Reserved.	Reset Value: 0
Register Address: 1500H–151FH, 5376–5407		IA32_LBR_x_FROM_IP
Last Branch Record entry X source IP register (R/W). An attempt to read or write IA32_LBR_x_FROM_IP such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		
63:0	FROM_IP The source IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
Register Address: 1600H–161FH, 5632–5663		IA32_LBR_x_TO_IP

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Last Branch Record Entry X Destination IP Register (R/W) An attempt to read or write IA32_LBR_x_TO_IP such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		
63:0	TO_IP The destination IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
Register Address: 17D0H, 6096		IA32_HW_FEEDBACK_PTR
Hardware Feedback Interface Pointer		If CPUID.06H:EAX[19] = 1
0	Valid (R/W) When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.	
11:1	Reserved.	
MAXPHYADDR-1:12	ADDR (R/W) Physical address of the page frame of the first page of the hardware feedback interface structure.	
63:MAXPHYADDR	Reserved.	
Register Address: 17D1H, 6097		IA32_HW_FEEDBACK_CONFIG
Hardware Feedback Interface Configuration		If CPUID.06H:EAX[19] = 1
0	Enable (R/W) When set to 1, enables the hardware feedback interface.	
63:1	Reserved.	
Register Address: 17D2H, 6098		IA32_THREAD_FEEDBACK_CHAR
Thread Feedback Characteristics (R/O)		If CPUID.06H:EAX[23] = 1
7:0	Application Class ID, pointing into the Intel Thread Director structure.	
62:8	Reserved.	
63	Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.	
Register Address: 17D4H, 6100		IA32_HW_FEEDBACK_THREAD_CONFIG
Hardware Feedback Thread Configuration (R/W)		
0	Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).	
63:1	Reserved.	
Register Address: 17DAH, 6106		IA32_HRESET_ENABLE
History Reset Enable (R/W)		
0	Enable reset of the Intel Thread Director history.	
31:1	Reserved for other capabilities that can be reset by the HRESET instruction.	
63:32	Reserved.	
Register Address: 1900H, 6400		IA32_PMC_GPO_CTR

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Full Width Writable General Performance Counter 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0 and IA32_PERF_CAPABILITIES[13] = 1
47:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:48	Reserved.	
Register Address: 1901H, 6401		IA32_PMC_GPO_CFG_A
IA32_PMC_GPO_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 0.		If CPUID.0AH:EAX[15:8] > 0
7:0	EVENT_SELECT Selects a performance event logic unit.	
15:8	UMASK Qualifies the microarchitectural condition to detect on the selected event logic.	
16	USR When set, events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.	
17	OS When set, events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USER flag.	
18	EDGE When set, enables edge detection of events.	
19	Reserved.	
20	INT When set, the processor generates an exception through its local APIC on counter overflow for this counter's thread.	
21	ANYTHREAD If CPUID.0AH:EDX[15] is 1, then this bit is deprecated. When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
22	ENABLE When set, performance counting is enabled in the performance-monitoring counter; when clear, the counter is disabled.	
23	INVERT Inverts the result of the counter-mask (CMASK) comparison when set, so that both greater than equal to and less than comparisons can be made. 0: The comparison is: threshold is greater than or equal to the event 1: The comparison is inverted: threshold is less than event.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31:24	CMASK When CMASK is not zero, the corresponding performance counter increments by 1 each cycle if the event count is $\geq$ CMASK. This mask enables counting cycles in which multiple occurrences happen (for example, two or more instructions retired per clock).	
34:32	Reserved.	
35	EN_LBR_LOG When set enables updating LBRs with that counters event occurrences, if selected event is precise.	
36	EQUAL When EQ flag is set and the INV flag is clear, the comparison evaluates to true if the selected performance monitoring event (the event) is equal to the specified Counter Mask value (CMask). When EQ flag is set and INV flag is set, the comparison evaluates to true if the event is less-than the CMask value and the event is not zero. Note if CMask is zero, the EQ flag is ignored.	
39:37	Reserved.	
47:40	UMASK2 Unit mask 2 (UMASK2) field (bits 40 through 47) - These bits qualify the condition that the selected event logic unit detects. Valid UMASK2 values for each event logic unit are specific to the unit. The new UMASK2 field may also be used in conjunction with UMASK.	
63:48	Reserved.	
Register Address: 1903H, 6403		IA32_PMC_GPO_CFG_C
IA32_PMC_GPO_CFG_C (R/W) Extended Perf event selector for GP counter 0.		CPUID.23H.05H:EAX[0] = 1 or CPUID.23H.02H:EAX[0] = 1
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
34:32	Reserved.	
35	ALLOW_IN_RECORD (R/W) If 1, indicates that this counter's value will be included in the PEBS record if the associated CNTR subgroup is enabled.	
38:36	CNTR (R/W) When any bit is set, the counter group will be included in the PEBS record. - Bit 36: If 1, the general-purpose counters subgroup will be included. - Bit 37: If 1, the fixed-function counters subgroup will be included. - Bit 38: If 1, the performance metrics subgroup will be included and will be reset	
39	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
41:40	LBR (R/W) Identifies the number of Last Branch Records (LBRs) to record. The encodings of this field are as follows: 00 - LBR group will not be recorded. The group is disabled. 01 - 8 LBRs will be recorded. 10 - 16 LBRs will be recorded. 11 - The number of LBRs to be recorded is IA32_LBR_DEPTH.DEPTH.	
48:42	Reserved.	
55:49	XER (R/W) When set, enables particular XSAVE-Enabled Registers to be included in the PEBS record. The encodings for the register subgroup are: - Bit 49 - SSER: Record XMM0-XMM15. - Bit 50 - YMMHIR: Record the upper 128 bits of YMM0-YMM15. - Bit 51 - EGPR: Record R16-R31. - Bit 52 - Reserved and must be zero. - Bit 53 - OPMASKR: Record K0-K7. - Bit 54 - ZMMHIR: Record the upper 256 bits of ZMM0-ZMM15. - Bit 55 - Hi16ZMMR: Record ZMM16-ZMM31.	
60:56	Reserved.	
61	GPR (R/W) If 1, enables the General-Purpose Registers Group to be included in the PEBS record.	
62	AUX (R/W) If 1, enables the Auxiliary Group to be included in the PEBS record.	
63	PEBS (R/W) If 1, PEBS is enabled for this counter.	
Register Address: 1904H, 6404		IA32_PMC_GP1_CTR
Full Width Writable General Performance Counter 1 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 1 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1905H, 6405		IA32_PMC_GP1_CFG_A
IA32_PMC_GP1_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 1. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 1
Register Address: 1907H, 6407		IA32_PMC_GP1_CFG_C
IA32_PMC_GP1_CFG_C (R/W) Extended Perf event selector for GP counter 1. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[1] = 1 or CPUID.23H.02H:EAX[1] = 1
Register Address: 1908H, 6408		IA32_PMC_GP2_CTR
Full Width Writable General Performance Counter 2 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 2 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1909H, 6409		IA32_PMC_GP2_CFG_A
IA32_PMC_GP2_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 2. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 190AH, 6410		IA32_PMC_GP2_CFG_B	
IA32_PMC_GP2_CFG_B (R/W)			
GP counter reload configuration register.			
1:0	Reserved.		
2	RELOAD_PMC2 Reload GP2 when GP2 overflows.		
3	RELOAD_PMC3 Reload GP2 when GP3 overflows.		
4	RELOAD_PMC4 Reload GP2 when GP4 overflows.		
5	RELOAD_PMC5 Reload GP2 when GP5 overflows.		
6	RELOAD_PMC6 Reload GP2 when GP6 overflows.		
7	RELOAD_PMC7 Reload GP2 when GP7 overflows.		
31:8	Reserved.		
32	RELOAD_FC0 Reload GP2 when FC0 overflows.		
33	RELOAD_FC1 Reload GP2 when FC1 overflows.		
47:34	Reserved.		
48	METRICS_CLEAR Clear PERF_METRICS on overflow of GP2.		
63:49	Reserved.		
Register Address: 190BH, 6411		IA32_PMC_GP2_CFG_C	
IA32_PMC_GP2_CFG_C (R/W)			
Extended Perf event selector for GP counter 2. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[2] = 1 or CPUID.23H.02H:EAX[2] = 1	
Register Address: 190CH, 6412		IA32_PMC_GP3_CTR	
Full Width Writable General Performance Counter 3 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 3 and IA32_PERF_CAPABILITIES[13] = 1	
Register Address: 190DH, 6413		IA32_PMC_GP3_CFG_A	
IA32_PMC_GP3_CFG_A (R/W)			
Performance Event Select Register used to control the operation of the General Performance Counter 3. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 3	
Register Address: 190EH, 6414		IA32_PMC_GP3_CFG_B	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_PMC_GP3_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 190FH, 6415		IA32_PMC_GP3_CFG_C
IA32_PMC_GP3_CFG_C (R/W) Extended Perf event selector for GP counter 3. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[3] = 1 or CPUID.23H.02H:EAX[3] = 1
Register Address: 1910H, 6416		IA32_PMC_GP4_CTR
Full Width Writable General Performance Counter 4 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 4 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1911H, 6417		IA32_PMC_GP4_CFG_A
IA32_PMC_GP4_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 4. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 4
Register Address: 1912H, 6418		IA32_PMC_GP4_CFG_B
IA32_PMC_GP4_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1913H, 6419		IA32_PMC_GP4_CFG_C
IA32_PMC_GP4_CFG_C (R/W) Extended Perf event selector for GP counter 4. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[4] = 1 or CPUID.23H.02H:EAX[4] = 1
Register Address: 1914H, 6420		IA32_PMC_GP5_CTR
Full Width Writable General Performance Counter 5 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 5 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1915H, 6421		IA32_PMC_GP5_CFG_A
IA32_PMC_GP5_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 5. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 5
Register Address: 1916H, 6422		IA32_PMC_GP5_CFG_B
IA32_PMC_GP5_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1917H, 6423		IA32_PMC_GP5_CFG_C
IA32_PMC_GP5_CFG_C (R/W) Extended Perf event selector for GP counter 5. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[5] = 1 or CPUID.23H.02H:EAX[5] = 1
Register Address: 1918H, 6424		IA32_PMC_GP6_CTR

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Full Width Writable General Performance Counter 6 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 6 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1919H, 6425		IA32_PMC_GP6_CFG_A
IA32_PMC_GP6_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 6. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 6
Register Address: 191AH, 6426		IA32_PMC_GP6_CFG_B
IA32_PMC_GP6_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 191BH, 6427		IA32_PMC_GP6_CFG_C
IA32_PMC_GP6_CFG_C (R/W) Extended Perf event selector for GP counter 6. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[6] = 1 or CPUID.23H.02H:EAX[6] = 1
Register Address: 191CH, 6428		IA32_PMC_GP7_CTR
Full Width Writable General Performance Counter 7 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 7 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 191DH, 6429		IA32_PMC_GP7_CFG_A
IA32_PMC_GP7_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 7. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 7
Register Address: 191EH, 6430		IA32_PMC_GP7_CFG_B
IA32_PMC_GP7_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 191FH, 6431		IA32_PMC_GP7_CFG_C
IA32_PMC_GP7_CFG_C (R/W) Extended Perf event selector for GP counter 7. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[7] = 1 or CPUID.23H.02H:EAX[7] = 1
Register Address: 1920H, 6432		IA32_PMC_GP8_CTR
Full Width Writable General Performance Counter 8 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 8 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1921H, 6433		IA32_PMC_GP8_CFG_A
IA32_PMC_GP8_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 8. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 8
Register Address: 1922H, 6430		IA32_PMC_GP8_CFG_B

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_PMC_GP8_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1923H, 6431		IA32_PMC_GP8_CFG_C
IA32_PMC_GP8_CFG_C (R/W) Extended Perf event selector for GP counter 8. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[8] = 1 or CPUID.23H.02H:EAX[8] = 1
Register Address: 1924H, 6436		IA32_PMC_GP9_CTR
Full Width Writable General Performance Counter 9 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 9 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1925H, 6437		IA32_PMC_GP9_CFG_A
IA32_PMC_GP9_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 9. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 9
Register Address: 1926H, 6430		IA32_PMC_GP9_CFG_B
IA32_PMC_GP9_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1927H, 6431		IA32_PMC_GP9_CFG_C
IA32_PMC_GP9_CFG_C (R/W) Extended Perf event selector for GP counter 9. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[9] = 1 or CPUID.23H.02H:EAX[9] = 1
Register Address: 1980H, 6528		IA32_PMC_FX0_CTR
Fixed-Function Performance Counter 0 (R/W) Instructions retired.		If CPUID.0AH:EDX[4:0] > 0
47:0	FIXED_COUNTER Instructions Retired Counter.	
63:46	Reserved.	
Register Address: 1982H, 6530		IA32_PMC_FX0_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter0 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter0 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter0 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter0 when GP5overflows.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
6	RELOAD_PMC6 Reload Fixed-Function Counter0 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter0 when GP7 overflows.	
33:8	Reserved.	
32	RELOAD_FC0 Reload Fixed-Function Counter0 when FC0 overflows.	
33	RELOAD_FC1 Reload Fixed-Function Counter0 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 0.	
63:49	Reserved.	
Register Address: 1983H, 6531		IA32_PMC_FX0_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 0 (R/W)		CPUID.23H.05H:ECX[0] = 1 or CPUID.23H.02H:EBX[0] = 1
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
34:32	Reserved.	
35	ALLOW_IN_RECORD (R/W) If 1, indicates that this counter's value will be included in the PEBS record if the associated CNTR subgroup is enabled.	
38:36	CNTR (R/W) When any bit is set, the counter group will be included in the PEBS record. - Bit 36: If 1, the general-purpose counters subgroup will be included. - Bit 37: If 1, the fixed-function counters subgroup will be included. - Bit 38: If 1, the performance metrics subgroup will be included and will be reset	
39	Reserved.	
41:40	LBR (R/W) Identifies the number of Last Branch Records (LBRs) to record. The encodings of this field are as follows: 00 – LBR group will not be recorded. The group is disabled. 01 – 8 LBRs will be recorded. 10 – 16 LBRs will be recorded. 11 – The number of LBRs to be recorded is IA32_LBR_DEPTH.DEPTH.	
48:42	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
55:49	XER (R/W) When set, enables particular XSAVE-Enabled Registers to be included in the PEBS record. The encodings for the register subgroup are: ▪ Bit 49 – SSER: Record XMM0-XMM15. ▪ Bit 50 – YMMHIR: Record the upper 128 bits of YMM0-YMM15. ▪ Bit 51 – EGPR: Record R16-R31. ▪ Bit 52 – Reserved and must be zero. ▪ Bit 53 – OPMASKR: Record K0-K7. ▪ Bit 54 – ZMMHIR: Record the upper 256 bits of ZMM0-ZMM15. ▪ Bit 55 – Hi16ZMMR: Record ZMM16-ZMM31.	
60:56	Reserved.	
61	GPR (R/W) If 1, enables the General-Purpose Registers Group to be included in the PEBS record.	
62	AUX (R/W) If 1, enables the Auxiliary Group to be included in the PEBS record.	
63	PEBS (R/W) If 1, PEBS is enabled for this counter.	
Register Address: 1984H, 6532		IA32_PMC_FX1_CTR
Fixed-Function Performance Counter 1 (R/W) Unhalted core clock cycles.		If CPUID.0AH:EDX[4:0] > 1
47:0	FIXED_COUNTER Unhalted core clock cycles counter.	
63:46	Reserved.	
Register Address: 1986H, 6534		IA32_PMC_FX1_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter1 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter1 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter1 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter1 when GP5 overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter1 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter1 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FCO Reload Fixed-Function Counter1 when FCO overflows.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
33	RELOAD_FC1 Reload Fixed-Function Counter1 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 1.	
63:49	Reserved.	
Register Address: 1987H, 6532		IA32_PMC_FX1_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 1 (R/W) See IA32_PMC_FX0_CFG_C (1983H) for reference; similar format.		CPUID.23H.05H:ECX[1] = 1 or CPUID.23H.02H:EBX[1] = 1
Register Address: 1988H, 6536		IA32_PMC_FX2_CTR
Fixed-Function Performance Counter 2 (R/W) Unhalted core reference cycles.		If CPUID.0AH:EDX[4:0] > 2
47:0	FIXED_COUNTER Unhalted core reference cycles counter.	
63:48	Reserved.	
Register Address: 198AH, 6538		IA32_PMC_FX2_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter2 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter2 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter2 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter2 when GP5overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter2 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter2 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FC0 Reload Fixed-Function Counter2 when FC0 overflows.	
33	RELOAD_FC1 Reload Fixed-Function Counter2 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 2.	
63:49	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 198BH, 6539		IA32_PMC_FX2_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 2 (R/W) See IA32_PMC_FX0_CFG_C (1983H) for reference; similar format.		CPUID.23H.05H:ECX[2] = 1 or CPUID.23H.02H:EBX[2] = 1
Register Address: 198CH, 6540		IA32_PMC_FX3_CTR
Fixed-Function Performance Counter 3 (R/W) Top-down Microarchitecture Analysis unhalting number of available slots.		If CPUID.0AH:EDX[4:0] > 3
47:0	FIXED_COUNTER Top-down microarchitecture analysis unhalting number of available slots counter.	
63:48	Reserved.	
Register Address: 198EH, 6542		IA32_PMC_FX3_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter3 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter3 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter3 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter3 when GP5 overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter3 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter3 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FC0 Reload Fixed-Function Counter3 when FC0 overflows.	
33	RELOAD_FC1 Reload Fixed-Function Counter3 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 3.	
63:49	Reserved.	
Register Address: 198FH, 6543		IA32_PMC_FX3_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 3 (R/W) See IA32_PMC_FX0_CFG_C (1983H) for reference; similar format.		CPUID.23H.05H:ECX[3] = 1 or CPUID.23H.02H:EBX[3] = 1
Register Address: 1990H, 6544		IA32_PMC_FX4_CTR
Fixed-Function Performance Counter 4 (R/W) Top-down bad speculation.		If CPUID.0AH:EDX[4:0] > 4

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:0	FIXED_COUNTER Top-down bad speculation counter.	
63:48	Reserved.	
Register Address: 1993H, 6547		IA32_PMC_FX4_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 4 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1994H, 6548		IA32_PMC_FX5_CTR
Fixed-Function Performance Counter 5 (R/W) Top-down frontend bound.		If CPUID.0AH:EDX[4:0] > 5
47:0	FIXED_COUNTER Top-down frontend-bound counter.	
63:48	Reserved.	
Register Address: 1997H, 6551		IA32_PMC_FX5_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 5 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1998H, 6552		IA32_PMC_FX6_CTR
Fixed-Function Performance Counter 6 (R/W) Top-down retiring.		If CPUID.0AH:EDX[4:0] > 6
47:0	FIXED_COUNTER Top-down retiring counter.	
63:48	Reserved.	
Register Address: 199BH, 6555		IA32_PMC_FX6_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 6 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1B01H, 6913		IA32_UARCH_MISC_CTL
IA32_UARCH_MISC_CTL (R/W)		If IA32_ARCH_CAPABILITIES[12] = 1
0	Data Operand Independent Timing Mode (DOITM).	If IA32_ARCH_CAPABILITIES[12] = 1
63:1	Reserved.	
Register Address: 4000_0000H–4000_00FFH		Reserved MSR Address Space

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
All existing and future processors will not implement MSRs in this range.		
Register Address: C000_0080H		IA32_EFER
Extended Feature Enables		If (CPUID.80000001H:EDX[20] CPUID.80000001H:EDX[29])
0	SYSCALL Enable: IA32_EFER.SCE (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.	
7:1	Reserved.	
8	IA-32e Mode Enable: IA32_EFER.LME (R/W) Enables IA-32e mode operation.	
9	Reserved.	
10	IA-32e Mode Active: IA32_EFER.LMA (R) Indicates IA-32e mode is active when set.	
11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W)	
63:12	Reserved.	
Register Address: C000_0081H		IA32_STAR
System Call Target Address (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0082H		IA32_LSTAR
IA-32e Mode System Call Target Address (R/W) Target RIP for the called procedure when SYSCALL is executed in 64-bit mode.		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0083H		IA32_CSTAR
IA-32e Mode System Call Target Address (R/W) Not used, as the SYSCALL instruction is not recognized in compatibility mode.		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0084H		IA32_FMASK
System Call Flag Mask (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0100H		IA32_FS_BASE
Map of BASE Address of FS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0101H		IA32_GS_BASE
Map of BASE Address of GS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0102H		IA32_KERNEL_GS_BASE
Swap Target of BASE Address of GS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0103H		IA32_TSC_AUX
Auxiliary TSC (R/W)		If CPUID.80000001H:EDX[27] = 1 or CPUID.07H.00H:ECX[22] = 1
31:0	AUX: Auxiliary signature of TSC.	
63:32	Reserved.	

NOTES:

1. Some older processors may have supported this MSR as model-specific and do not enumerate it with CPUID.
2. In processors based on Intel NetBurst® microarchitecture, MSR addresses 180H-197H are supported, software must treat them as model-specific. Starting with Intel Core Duo processors, MSR addresses 180H-185H, 188H-197H are reserved.

3. The *_ADDR MSRs may or may not be present; this depends on flag settings in IA32_MCI_STATUS. See Section 18.3.2.3 and Section 18.3.2.4 for more information.
4. Further details on Key Locker and usage of this MSR can be found here:
<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.

2.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table 2-3 lists model-specific registers (MSRs) for the Intel Core 2 processor family and for Intel Xeon processors based on Intel Core microarchitecture, architectural MSR addresses are also included in Table 2-3. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_0FH, see Table 2-1.

MSRs listed in Table 2-2 and Table 2-3 are also supported by processors based on the Enhanced Intel Core microarchitecture. Processors based on the Enhanced Intel Core microarchitecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_17H.

The column “Shared/Unique” applies to multi-core processors based on Intel Core microarchitecture. “Unique” means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. “Shared” means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Unique
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, “MSRs in Pentium Processors.”		Unique
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, “Time-Stamp Counter,” and Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Shared
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Shared
7:0	Reserved.	
12:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
49:13	Reserved.	
52:50	See Table 2-2.	
63:53	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, “Local APIC Status and Location,” and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
3	MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
4	Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
5	Reserved.	
6	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
11	Intel TXT Capable Chipset. (R/O) 1 = Present; 0 = Not Present.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
13	Reserved.	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes.	
15	Reserved.	
17:16	APIC Cluster ID (R/O)	
18	N/2 Non-Integer Bus Ratio (R/O) 0 = Integer ratio; 1 = Non-integer ratio.	
19	Reserved.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
21:20	Symmetric Arbitration ID (R/O)	
26:22	Integer Bus Frequency Ratio (R/O)	
Register Address: 3AH, 58	MSR_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Unique
3	SMRR Enable (R/WL) When this bit is set and the lock bit is set, this makes the SMRR_PHYS_BASE and SMRR_PHYS_MASK registers read visible and writeable while in SMM.	Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of four pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.5.		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of four pairs of last branch record registers on the last branch record stack. This To_IP part of the stack contains pointers to the destination instruction.		Unique
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
BIOS Update Trigger Register (W) See Table 2-2.		Unique
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: A0H, 160	MSR_SMRR_PHYSBASE	
System Management Mode Base Address register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.		Unique
11:0	Reserved.	
31:12	PhysBase: SMRR physical Base Address.	
63:32	Reserved.	
Register Address: A1H, 161	MSR_SMRR_PHYSMASK	
System Management Mode Physical Address Mask register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.		Unique
10:0	Reserved.	
11	Valid: Physical address base and range mask are valid.	
31:12	PhysMask: SMRR physical address range mask.	
63:32	Reserved.	
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Core microarchitecture.		Shared
2:0	▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333)	
	133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
	266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.	
63:3	Reserved.	
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Enhanced Intel Core microarchitecture.		Shared
2:0	▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) ▪ 110B: 400 MHz (FSB 1600) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Unique
11	SMRR Capability Using MSR 0A0H and 0A1H (R)	Unique
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Shared
Register Address: 198H, 408	MSR_PERF_STATUS	
Current performance status. See Section 17.1.1, "Software Interface For Initiating Performance State Transitions."		Shared
15:0	Current Performance State Value	
30:16	Reserved.	
31	XE Operation (R/O). If set, XE operation is enabled. Default is cleared.	
39:32	Reserved.	
44:40	Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.	
45	Reserved.	
46	Non-Integer Bus Ratio (R/O) Indicates non-integer bus ratio is enabled. Applies processors based on Enhanced Intel Core microarchitecture.	
63:47	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Unique
15:0	Reserved.	
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.	
63:16	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
8	Reserved.	
9	Hardware Prefetcher Disable (R/W) When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Shared
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.01H:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line that contains data currently required by the processor. When set to 0, the processor fetches cache lines that comprise a cache line pair (128 bytes). Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.	Shared
20	Enhanced Intel SpeedStep Technology Select Lock (R/WO) When set, this bit causes the following bits to become read-only: - Enhanced Intel SpeedStep Technology Select Lock (this bit). - Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.	Shared
21	Reserved.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
22	Limit CPUID Maxval (R/W) See Table 2-2.	Shared
23	xTPR Message Disable (R/W) See Table 2-2.	Shared
33:24	Reserved.	
34	XD Bit Disable (R/W) When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H:EDX[20] =0). When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages. BIOS must not alter the contents of this bit location if XD bit is not supported. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.	Unique
36:35	Reserved.	
37	DCU Prefetcher Disable (R/W) When set to 1, the DCU L1 data cache prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The DCU prefetcher is an L1 data cache prefetcher. When the DCU prefetcher detects multiple loads from the same line done within a time limit, the DCU prefetcher assumes the next line will be required. The next line is prefetched in to the L1 data cache from memory or L2.	Unique
38	IDA Disable (R/W) When set to 1 on processors that support IDA, the Intel Dynamic Acceleration feature (IDA) is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H:EAX[1] =0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of IDA is enabled. Note: The power-on default value is used by BIOS to detect hardware support of IDA. If the power-on default value is 1, IDA is available in the processor. If the power-on default value is 0, IDA is not available.	Shared
39	IP Prefetcher Disable (R/W) When set to 1, the IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The IP prefetcher is an L1 data cache prefetcher. The IP prefetcher looks for sequential load history to determine whether to prefetch the next expected data into the L1 cache from memory or L2.	Unique
63:40	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Debug Control (R/W) See Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Unique
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Unique
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Unique
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Unique
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Unique
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Unique
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Unique
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Unique
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Unique
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Unique
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Unique
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Unique
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Unique

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Unique
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Unique
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Unique
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Unique
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Unique
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Unique
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Unique
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Unique
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Unique
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Unique
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Unique
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Unique
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Unique
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Unique
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Unique
Register Address: 309H, 777	IA32_FIXED_CTR0	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Unique
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Unique
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Unique
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Unique
Register Address: 345H, 837	MSR_PERF_CAPABILITIES	
R/O. This applies to processors that do not support architectural PerfMon version 2.		Unique
5:0	LBR Format. See Table 2-2.	
6	PEBS Record Format.	
7	PEBSSaveArchRegs. See Table 2-2.	
63:8	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Unique
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38EH, 910	MSR_PERF_GLOBAL_STATUS	
See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	MSR_PERF_GLOBAL_CTRL	
See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	MSR_PERF_GLOBAL_OVF_CTRL	
See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Unique
0	Enable PEBS on IA32_PMC0. (R/W)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 40CH, 1036	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 40DH, 1037	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 40EH, 1038	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 410H, 1040	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		
Register Address: 411H, 1041	IA32_MC3_STATUS	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS."		
Register Address: 412H, 1042	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRS." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 413H, 1043	IA32_MC3_MISC	
Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 414H, 1044	IA32_MC5_CTL	
Machine Check Error Reporting Register: Controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).		Unique
Register Address: 415H, 1045	IA32_MC5_STATUS	
Machine Check Error Reporting Register: Contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.		Unique
Register Address: 416H, 1046	IA32_MC5_ADDR	
Machine Check Error Reporting Register: Contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 417H, 1047	IA32_MC5_MISC	
Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 419H, 1045	IA32_MC6_STATUS	
Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 26.		Unique
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2. See Section 22.6.3.4, "Debug Store (DS) Mechanism."		Unique
Register Address: 107CCH, 67532	MSR_EMON_L3_CTR_CTL0	
GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107CDH, 67533	MSR_EMON_L3_CTR_CTL1	
GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107CEH, 67534	MSR_EMON_L3_CTR_CTL2	
GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107CFH, 67535	MSR_EMON_L3_CTR_CTL3	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
GSPNQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D0H, 67536	MSR_EMON_L3_CTR_CTL4	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D1H, 67537	MSR_EMON_L3_CTR_CTL5	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D2H, 67538	MSR_EMON_L3_CTR_CTL6	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D3H, 67539	MSR_EMON_L3_CTR_CTL7	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D8H, 67544	MSR_EMON_L3_GL_CTL	
L3/FSB Common Control Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Unique
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Unique
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Unique
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Unique

2.3 MSRS IN THE 45 NM AND 32 NM INTEL ATOM® PROCESSOR FAMILY

Table 2-4 lists model-specific registers (MSRs) for 45 nm and 32 nm Intel Atom processors, architectural MSR addresses are also included in Table 2-4. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_1CH, 06_26H, 06_27H, 06_35H, or 06_36H; see Table 2-1.

The column “Shared/Unique” applies to logical processors sharing the same core in processors based on the Intel Atom microarchitecture. “Unique” means each logical processor has a separate MSR, or a bit field in an MSR governs only a logical processor. “Shared” means the MSR or the bit field in an MSR address governs the operation of both logical processors in the same core.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Shared
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, “MSRs in Pentium Processors.”		Shared
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, “Time-Stamp Counter,” and see Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Shared
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Shared
7:0	Reserved.	
12:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
63:13	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, “Local APIC Status and Location,” and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
3	AERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
4	BERR# Enable for initiator bus requests (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
5	Reserved.	
6	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
8	Reserved.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	AERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0.	
11	Reserved.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0.	
13	Reserved.	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes.	
15	Reserved.	
17:16	APIC Cluster ID (R/O) Always 00B.	
19:18	Reserved.	
21:20	Symmetric Arbitration ID (R/O) Always 00B.	
26:22	Integer Bus Frequency Ratio (R/O)	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.5.		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 44H, 68	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 45H, 69	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 46H, 70	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 47H, 71	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.		Unique
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 64H, 100	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 65H, 101	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 66H, 102	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 67H, 103	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Shared
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: C1H, 193	IA32_PMC0	
Performance counter register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Atom microarchitecture.		Shared
2:0	111B: 083 MHz (FSB 333) 101B: 100 MHz (FSB 400) 001B: 133 MHz (FSB 533) 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R) See Table 2-2.		Shared
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Shared
0	L2 Hardware Enabled (R/O) 1 = Indicates the L2 is hardware-enabled. 0 = Indicates the L2 is hardware-disabled.	
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set, the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSELO	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Shared
Register Address: 198H, 408	MSR_PERF_STATUS	
Performance Status		Shared
15:0	Current Performance State Value.	
39:16	Reserved.	
44:40	Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.	
63:45	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Unique

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Shared
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.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		Unique
0	Fast-Strings Enable See Table 2-2.	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
8	Reserved.	
9	Reserved.	
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is cleared (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.01H:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Reserved.	
20	Enhanced Intel SpeedStep Technology Select Lock (R/W0) When set, this bit causes the following bits to become read-only: - Enhanced Intel SpeedStep Technology Select Lock (this bit). - Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.	Shared
21	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Unique
23	xTPR Message Disable (R/W) See Table 2-2.	Shared
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Unique
63:35	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Unique

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Shared
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Shared
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Shared
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Shared
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Shared
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Shared
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Shared
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Shared
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Shared
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Shared
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Shared
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Shared
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Shared
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Shared
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Shared
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Shared
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Shared
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Shared
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Shared
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Shared
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Shared
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Shared
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Shared
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Shared
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Shared
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Shared
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Unique
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Unique
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Unique
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Unique
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Unique
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Unique
0	Enable PEBS on IA32_PMC0 (R/W)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Shared
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2. See Section 22.6.3.4, "Debug Store (DS) Mechanism."		Unique
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Unique
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Unique
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Unique
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Unique

Table 2-5 lists model-specific registers (MSRs) that are specific to Intel Atom® processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_27H.

Table 2-5. MSRs Supported by Intel Atom® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_27H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3F8H, 1016	MSR_PKG_C2_RESIDENCY	
Package C2 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C2 Residency Counter (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.	Package
Register Address: 3F9H, 1017	MSR_PKG_C4_RESIDENCY	
Package C4 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.	Package
Register Address: 3FAH, 1018	MSR_PKG_C6_RESIDENCY	
Package C6 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.	Package

2.4 MSRS IN INTEL PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 2-6 lists model-specific registers (MSRs) common to Intel processors based on the Silvermont microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_37H, 06_4AH, 06_4DH, 06_5AH, or 06_5DH; see Table 2-1. The MSRs listed in Table 2-6 are also common to processors based on the Airmont microarchitecture and newer microarchitectures for next generation Intel Atom processors.

Table 2-7 lists MSRs common to processors based on the Silvermont and Airmont microarchitectures, but not newer microarchitectures.

Table 2-8, Table 2-9, and Table 2-10 lists MSRs that are model-specific across processors based on the Silvermont microarchitecture.

In the Silvermont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Silvermont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Core
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, "Time-Stamp Counter," and Table 2-2.		Core
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Core
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Writes ignored.		Module
63:0	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Core
31:0	SMI Count (R/O) Running count of SMI events since last RESET.	
63:32	Reserved.	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Core
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Core
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Core
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Module

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 100b - C4 is the max C-State to include 110b - C6 is the max C-State to include 111b - C7 is the max C-State to include	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Core
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Core
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction sets availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note: AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Core
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Core
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Core
Register Address: 179H, 377	IA32_MCG_CAP	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Core
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Core
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSELO	
See Table 2-2.		Core
7:0	Event Select	
15:8	UMask	
16	USR	
17	OS	
18	Edge	
19	PC	
20	INT	
21	Reserved.	
22	EN	
23	INV	
31:24	CMASK	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Core
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Module
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Core
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Core
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R) The default thermal throttling or PROCHOT# activation temperature in degrees C. The effective temperature for thermal throttling or PROCHOT# activation is "Temperature Target" + "Target Offset".	
29:24	Target Offset (R/W) Specifies an offset in degrees C to adjust the throttling and PROCHOT# activation temperature from the default target specified in TEMPERATURE_TARGET (bits 23:16).	
63:30	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
Offcore Response Event Select Register (R/W)		Module
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Module
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Core
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Core
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Core
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Core
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Core
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Core
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Core
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Core
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Core
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Core
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Core
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Core
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Core
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Core
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Core
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Core
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Core
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Core
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Core
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Core
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Core
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Core
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Core
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Core
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Core
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Core
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Core
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Core
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Core
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Core
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Core
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Core
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Core
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Core
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Core
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Core
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Core
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Counts at the TSC Frequency.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Module
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Module
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Module
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 410H, 1040	IA32_MC4_CTL	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Core
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Core
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Core
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."	Core	
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."	Core	
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."	Core	
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."	Core	
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."	Core	
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."	Core	
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.	Core	
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTL2	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.	Core	
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTL2	
Capability Reporting Register of Primary Processor-based VM-Execution Flex Controls (R/O) See Table 2-2.	Core	
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTL2	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.	Core	
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTL2	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.	Core	
Register Address: 491H, 1169	IA32_VMX_FMFUNC	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Core
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Core
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Core
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		Core
Register Address: 660H, 1632	MSR_CORE_C1_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C1 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C1 states. Counts at the TSC frequency.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Core
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Core
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Core
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Core
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Core
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Core
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Core
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2.		Core

Table 2-7 lists model-specific registers (MSRs) that are common to Intel Atom® processors based on the Silvermont and Airmont microarchitectures but not newer microarchitectures.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Module
7:0	Reserved.	
13:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
49:13	Reserved.	
52:50	See Table 2-2.	
63:33	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Reserved.	
2	Enable VMX outside SMX operation (R/WL)	
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.5 and record format in Section 20.4.8.1.		Core
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 44H, 68	MSR_LASTBRANCH_4_FROM_IP	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 45H, 69	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 46H, 70	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 47H, 71	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.		Core
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 64H, 100	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 65H, 101	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 66H, 102	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 67H, 103	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information: Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * Scalable Bus Frequency.	Package
63:16	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Module
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 100b: C4 110b: C6 111b: C7 (Silvermont only)	
9:3	Reserved.	
10	I/O MWait Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWait instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Module
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.	
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Core
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.	Module
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Core
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Core
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Core
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Module
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Core
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Core
23	xTPR Message Disable (R/W) See Table 2-2.	Module
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Core
37:35	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Module
63:39	Reserved.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS for precise event on IA32_PMC0 (R/W)	
Register Address: 3FAH, 1018	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Counts at the TSC Frequency.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Module
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	

2.4.1 MSRs with Model-Specific Behavior in the Silvermont Microarchitecture

Table 2-8 lists MSRs that are specific to the Intel Atom[®] processor E3000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_37H) and Intel Atom processors (CPUID Signature DisplayFamily_DisplayModel value of 06_4AH, 06_5AH, or 06_5DH).

Table 2-8. Specific MSRs Supported by Intel Atom[®] Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H, 06_4AH, 06_5AH, or 06_5DH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Silvermont microarchitecture.		Module
2:0	100B: 080.0 MHz 000B: 083.3 MHz 001B: 100.0 MHz 010B: 133.3 MHz 011B: 116.7 MHz	
63:3	Reserved.	
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
	Unit Multipliers used in RAPL Interfaces (R/O) See Section 17.10.1, "RAPL Interfaces."	Package
3:0	Power Units Power related information (in milliWatts) is based on the multiplier, 2^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.	
7:4	Reserved.	
12:8	Energy Status Units Energy related information (in microJoules) is based on the multiplier, 2^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.	
15:13	Reserved.	
19:16	Time Unit The value is 0000b, indicating time unit is in one second.	
63:20	Reserved.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W)		Package

Table 2-8. Specific MSRs Supported by Intel Atom® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H, 06_4AH, 06_5AH, or 06_5DH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	Package Power Limit #1 (R/W) See Section 17.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.	
15	Enable Power Limit #1 (R/W) See Section 17.10.3, "Package RAPL Domain."	
16	Package Clamping Limitation #1 (R/W) See Section 17.10.3, "Package RAPL Domain."	
23:17	Time Window for Power Limit #1 (R/W) In unit of second. If 0 is specified in bits [23:17], defaults to 1 second window.	
63:24	Reserved.	
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 17.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains," and MSR_RAPL_POWER_UNIT in Table 2-8.		Package

Table 2-9 lists model-specific registers (MSRs) that are specific to the Intel Atom® processor E3000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_37H).

Table 2-9. Specific MSRs Supported by the Intel Atom® Processor E3000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 668H, 1640	MSR_CC6_DEMOTION_POLICY_CONFIG	
Core C6 Demotion Policy Config MSR		Package
63:0	Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.	
Register Address: 669H, 1641	MSR_MC6_DEMOTION_POLICY_CONFIG	
Module C6 Demotion Policy Config MSR		Package
63:0	Controls module (i.e., two cores sharing the second-level cache) C6 demotion policy. Writing a value of 0 disables module level HW demotion policy.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Module
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	

Table 2-10 lists model-specific registers (MSRs) that are specific to Intel Atom® processor C2000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_4DH).

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	Reserved.	
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
63:3	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode (R/W)		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.	Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 17.10.1, "RAPL Interfaces."		Package
3:0	Power Units Power related information (in milliWatts) is based on the multiplier, 2^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.	
7:4	Reserved.	

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
12:8	Energy Status Units. Energy related information (in microJoules) is based on the multiplier, 2^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.	
15:13	Reserved.	
19:16	Time Unit The value is 0000b, indicating time unit is in one second.	
63:20	Reserved.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 66EH, 1646	MSR_PKG_POWER_INFO	
PKG RAPL Parameter (R/O)		Package
14:0	Thermal Spec Power (R/O) The unsigned integer value is the equivalent of the thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.	
63:15	Reserved.	

2.4.2 MSRs in Intel Atom® Processors Based on Airmont Microarchitecture

Intel Atom processor X7-Z8000 and X5-Z8000 series are based on the Airmont microarchitecture. These processors support MSRs listed in Table 2-6, Table 2-7, Table 2-8, and Table 2-11. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_4CH; see Table 2-1.

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Airmont microarchitecture.		Module
3:0	▪ 0000B: 083.3 MHz ▪ 0001B: 100.0 MHz ▪ 0010B: 133.3 MHz ▪ 0011B: 116.7 MHz ▪ 0100B: 080.0 MHz ▪ 0101B: 093.3 MHz ▪ 0110B: 090.0 MHz ▪ 0111B: 088.9 MHz ▪ 1000B: 087.5 MHz	
63:5	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Module
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: No limit 001b: C1 010b: C2 110b: C6 111b: C7	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/W0) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Module
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PMG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - Deep Power Down Technology is the max C-State. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
PPO RAPL Power Limit Control (R/W)		Package

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	PP0 Power Limit #1 (R/W) See Section 17.10.4, “PP0/PP1 RAPL Domains,” and MSR_RAPL_POWER_UNIT in Table 2-8.	
15	Enable Power Limit #1 (R/W) See Section 17.10.4, “PP0/PP1 RAPL Domains.”	
16	Reserved.	
23:17	Time Window for Power Limit #1 (R/W) Specifies the time duration over which the average power must remain below PP0_POWER_LIMIT #1(14:0). Supported Encodings: 0x0: 1 second time duration. 0x1: 5 second time duration (Default). 0x2: 10 second time duration. 0x3: 15 second time duration. 0x4: 20 second time duration. 0x5: 25 second time duration. 0x6: 30 second time duration. 0x7: 35 second time duration. 0x8: 40 second time duration. 0x9: 45 second time duration. 0xA: 50 second time duration. 0xB-0x7F - reserved.	
63:24	Reserved.	

2.5 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Intel Atom processors based on the Goldmont microarchitecture support MSRs listed in Table 2-6 and Table 2-12. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_5CH; see Table 2-1.

In the Goldmont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Goldmont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Module
49:0	Reserved.	
52:50	See Table 2-2.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:33	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Enable VMX inside SMX operation (R/WL)	
2	Enable VMX outside SMX operation (R/WL)	
14:8	SENTER local functions enables (R/WL)	
15	SENTER global functions enable (R/WL)	
18	SGX global functions enable (R/WL)	
63:19	Reserved.	
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Core TSC ADJUST (R/W) See Table 2-2.		Core
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Core
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
30	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.	Package
39:31	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: No limit 0001b: C1 0010b: C3 0011b: C6 0100b: C7 0101b: C7S 0110b: C8 0111b: C9 1000b: C10	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability enhancement. Accessible only while in SMM.		Core
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.	
59	Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.	
63:60	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Core
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Core
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Core
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.	Package
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Core
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Core
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Core
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Core
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Core
23	xTPR Message Disable (R/W) See Table 2-2.	Package
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Core
37:35	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	Reserved.	
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
63:3	Reserved.	
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org.		Package
0	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.	
21:1	Reserved.	
22	Thermal Interrupt Coordination Enable (R/W) If set, then thermal interrupt on one core is routed to all cores.	
63:23	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode by Core Groups (R/W) Specifies Maximum Ratio Limit for each Core Group. Max ratio for groups with more cores must decrease monotonically. For groups with less than 4 cores, the max ratio must be 32 or less. For groups with 4-5 cores, the max ratio must be 22 or less. For groups with more than 5 cores, the max ratio must be 16 or less.		Package
7:0	Maximum Ratio Limit for Active Cores in Group 0 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 0 threshold.	Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
15:8	Maximum Ratio Limit for Active Cores in Group 1 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 1 threshold, and greater than the Group 0 threshold.	Package
23:16	Maximum Ratio Limit for Active Cores in Group 2 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 2 threshold, and greater than the Group 1 threshold.	Package
31:24	Maximum Ratio Limit for Active Cores in Group 3 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 3 threshold, and greater than the Group 2 threshold.	Package
39:32	Maximum Ratio Limit for Active Cores in Group 4 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 4 threshold, and greater than the Group 3 threshold.	Package
47:40	Maximum Ratio Limit for Active Cores in Group 5 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 5 threshold, and greater than the Group 4 threshold.	Package
55:48	Maximum Ratio Limit for Active Cores in Group 6 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 6 threshold, and greater than the Group 5 threshold.	Package
63:56	Maximum Ratio Limit for Active Cores in Group 7 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 7 threshold, and greater than the Group 6 threshold.	Package
Register Address: 1AEH, 430	MSR_TURBO_GROUP_CORECNT	
Group Size of Active Cores for Turbo Mode Operation (R/W) Writes of 0 threshold is ignored.		Package
7:0	Group 0 Core Count Threshold Maximum number of active cores to operate under the Group 0 Max Turbo Ratio limit.	Package
15:8	Group 1 Core Count Threshold Maximum number of active cores to operate under the Group 1 Max Turbo Ratio limit. Must be greater than the Group 0 Core Count.	Package
23:16	Group 2 Core Count Threshold Maximum number of active cores to operate under the Group 2 Max Turbo Ratio limit. Must be greater than the Group 1 Core Count.	Package
31:24	Group 3 Core Count Threshold Maximum number of active cores to operate under the Group 3 Max Turbo Ratio limit. Must be greater than the Group 2 Core Count.	Package
39:32	Group 4 Core Count Threshold Maximum number of active cores to operate under the Group 4 Max Turbo Ratio limit. Must be greater than the Group 3 Core Count.	Package
47:40	Group 5 Core Count Threshold Maximum number of active cores to operate under the Group 5 Max Turbo Ratio limit. Must be greater than the Group 4 Core Count.	Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55:48	Group 6 Core Count Threshold Maximum number of active cores to operate under the Group 6 Max Turbo Ratio limit. Must be greater than the Group 5 Core Count.	Package
63:56	Group 7 Core Count Threshold Maximum number of active cores to operate under the Group 7 Max Turbo Ratio limit. Must be greater than the Group 6 Core Count, and not less than the total number of processor cores in the package. E.g., specify 255.	Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
9	EN_CALL_STACK	
63:10	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
63:2	Reserved.	
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Core
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Core
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
See Table 2-2.		Core

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address:	IA32_MTRR_PHYSMASK9	
213H, 531	See Table 2-2.	Core
Register Address:	IA32_MC0_CTL2	
280H, 640	See Table 2-2.	Module
Register Address:	IA32_MC1_CTL2	
281H, 641	See Table 2-2.	Module
Register Address:	IA32_MC2_CTL2	
282H, 642	See Table 2-2.	Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Module
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 300H, 768	MSR_SGXOWNEREPPOCH0	
Lower 64 Bit CR_SGXOWNEREPPOCH (W) Writes do not update CR_SGXOWNEREPPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 301H, 769	MSR_SGXOWNEREPPOCH1	
Upper 64 Bit CR_SGXOWNEREPPOCH (W) Writes do not update CR_SGXOWNEREPPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 22.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Ovf_PMC0	
1	Ovf_PMC1	
2	Ovf_PMC2	
3	Ovf_PMC3	
31:4	Reserved.	
32	Ovf_FixedCtr0	
33	Ovf_FixedCtr1	
34	Ovf_FixedCtr2	
54:35	Reserved.	
55	Trace_ToPA_PMI	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
57:56	Reserved.	
58	LBR_Frz	
59	CTR_Frz	
60	ASCI	
61	Ovf_Uncore	
62	Ovf_BufDSSAVE	
63	CondChgd	
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
See Table 2-2 and Section 22.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Set 1 to clear Ovf_PMC0.	
1	Set 1 to clear Ovf_PMC1.	
2	Set 1 to clear Ovf_PMC2.	
3	Set 1 to clear Ovf_PMC3.	
31:4	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	
33	Set 1 to clear Ovf_FixedCtr1.	
34	Set 1 to clear Ovf_FixedCtr2.	
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI.	
57:56	Reserved.	
58	Set 1 to clear LBR_Frz.	
59	Set 1 to clear CTR_Frz.	
60	Set 1 to clear ASCI.	
61	Set 1 to clear Ovf_Uncore.	
62	Set 1 to clear Ovf_BufDSSAVE.	
63	Set 1 to clear CondChgd.	
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
See Table 2-2 and Section 22.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Set 1 to cause Ovf_PMC0 = 1.	
1	Set 1 to cause Ovf_PMC1 = 1.	
2	Set 1 to cause Ovf_PMC2 = 1.	
3	Set 1 to cause Ovf_PMC3 = 1.	
31:4	Reserved.	
32	Set 1 to cause Ovf_FixedCtr0 = 1.	
33	Set 1 to cause Ovf_FixedCtr1 = 1.	
34	Set 1 to cause Ovf_FixedCtr2 = 1.	
54:35	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55	Set 1 to cause Trace_ToPA_PMI = 1.	
57:56	Reserved.	
58	Set 1 to cause LBR_Frz = 1.	
59	Set 1 to cause CTR_Frz = 1.	
60	Set 1 to cause ASCI = 1.	
61	Set 1 to cause Ovf_Uncore.	
62	Set 1 to cause Ovf_BufDSSAVE.	
63	Reserved.	
Register Address: 392H, 914	IA32_PERF_GLOBAL_INUSE	
See Table 2-2.		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2 and Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0. (R/W)	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 4C3H, 1219	IA32_A_PMC2	
See Table 2-2.		Core
Register Address: 4C4H, 1220	IA32_A_PMC3	
See Table 2-2.		Core
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.		Package
0	Lock (SMM-RW0) When set to '1' locks this register from further changes.	
1	Reserved.	
2	SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 4E2H, 1250	MSR_SMM_DELAYED	
SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.		Package
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 4E3H, 1251	MSR_SMM_BLOCKED	
SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is 0FFFH. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 500H, 1280	IA32_SGX_SVN_STATUS	
Status and SVN Threshold of SGX Support for ACM (R/O)		Core
0	Lock See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
15:1	Reserved.	
23:16	SGX_SVN_SINIT See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
63:24	Reserved.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W) See Table 2-2.		Core
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W) See Table 2-2.		Core
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Core
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	BranchEn	
17:14	MTCFreq	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
18	Reserved, must be zero.	
22:19	CycThresh	
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Core
0	FilterEn Writes ignored.	
1	ContextEn Writes ignored.	
2	TriggerEn Writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
31:6	Reserved, must be zero.	
48:32	PacketByteCnt	
63:49	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Core
4:0	Reserved	
63:5	CR3[63:5] value to match.	
Register Address: 580H, 1408	IA32_RTIT_ADDR0_A	
Region 0 Start Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 581H, 1409	IA32_RTIT_ADDR0_B	
Region 0 End Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 582H, 1410	IA32_RTIT_ADDR1_A	
Region 1 Start Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 583H, 1411	IA32_RTIT_ADDR1_B	
Region 1 End Address (R/W)		Core
63:0	See Table 2-2.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 17.10.1, "RAPL Interfaces."		Package
3:0	Power Units Power related information (in Watts) is in unit of $1W/2^{PU}$; where PU is an unsigned integer represented by bits 3:0. Default value is 1000b, indicating power unit is in 3.9 milliWatts increment.	
7:4	Reserved.	
12:8	Energy Status Units Energy related information (in Joules) is in unit of $1Joule/2^{ESU}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 01110b, indicating energy unit is in 61 microjoules.	
15:13	Reserved.	
19:16	Time Unit Time related information (in seconds) is in unit of $1S/2^{TU}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating power unit is in 0.977 millisecond.	
63:20	Reserved.	
Register Address: 60AH, 1546	MSR_PKGC3_IRTL	
Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60BH, 1547	MSR_PKGC_IRTL1	
Package C6/C7S Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7S state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7S state.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60CH, 1548	MSR_PKGC_IRTL2	
Package C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W)		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	Thermal Spec Power (R/W) See Section 17.10.3, "Package RAPL Domain."	
15	Reserved.	
30:16	Minimum Power (R/W) See Section 17.10.3, "Package RAPL Domain."	
31	Reserved.	
46:32	Maximum Power (R/W) See Section 17.10.3, "Package RAPL Domain."	
47	Reserved.	
54:48	Maximum Time Window (R/W) Specified by $2^Y * (1.0 + Z/4.0) * \text{Time\_Unit}$, where "Y" is the unsigned integer value represented by bits 52:48, "Z" is an unsigned integer represented by bits 54:53. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.	
63:55	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C10 Residency Counter (R/O) Value since last reset that the entire SOC is in an S0i3 state. Count at the same frequency as the TSC.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
3	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
8:4	Reserved.	
9	Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	
10	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
11	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
12	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
14	Maximum Efficiency Frequency Status (R0) When set, frequency is reduced below the maximum efficiency frequency.	
15	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24:20	Reserved.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Maximum Efficiency Frequency Log When set, indicates that the Maximum Efficiency Frequency Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:31	Reserved.	
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 0 From IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.6 and record format in Section 20.4.8.1.		Core
0:47	From Linear Address (R/W)	
62:48	Signed extension of bits 47:0.	
63	Mispred	
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 690H, 1680	MSR_LASTBRANCH_16_FROM_IP	
Last Branch Record 16 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 691H, 1681	MSR_LASTBRANCH_17_FROM_IP	
Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 692H, 1682	MSR_LASTBRANCH_18_FROM_IP	
Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 693H, 1683	MSR_LASTBRANCH_19_FROM_IP	
Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 694H, 1684	MSR_LASTBRANCH_20_FROM_IP	
Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 695H, 1685	MSR_LASTBRANCH_21_FROM_IP	
Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 696H, 1686	MSR_LASTBRANCH_22_FROM_IP	
Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 697H, 1687	MSR_LASTBRANCH_23_FROM_IP	
Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 698H, 1688	MSR_LASTBRANCH_24_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 699H, 1689	MSR_LASTBRANCH_25_FROM_IP	
Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69AH, 1690	MSR_LASTBRANCH_26_FROM_IP	
Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69BH, 1691	MSR_LASTBRANCH_27_FROM_IP	
Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69CH, 1692	MSR_LASTBRANCH_28_FROM_IP	
Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69DH, 1693	MSR_LASTBRANCH_29_FROM_IP	
Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69EH, 1694	MSR_LASTBRANCH_30_FROM_IP	
Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69FH, 1695	MSR_LASTBRANCH_31_FROM_IP	
Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the Destination instruction and elapsed cycles from last LBR update. See Section 20.6.		Core
0:47	Target Linear Address (R/W)	
63:48	Elapsed cycles from last update to the LBR.	
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DOH, 1744	MSR_LASTBRANCH_16_TO_IP	
Last Branch Record 16 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D1H, 1745	MSR_LASTBRANCH_17_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D2H, 1746	MSR_LASTBRANCH_18_TO_IP	
Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D3H, 1747	MSR_LASTBRANCH_19_TO_IP	
Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D4H, 1748	MSR_LASTBRANCH_20_TO_IP	
Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D5H, 1749	MSR_LASTBRANCH_21_TO_IP	
Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D6H, 1750	MSR_LASTBRANCH_22_TO_IP	
Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D7H, 1751	MSR_LASTBRANCH_23_TO_IP	
Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D8H, 1752	MSR_LASTBRANCH_24_TO_IP	
Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D9H, 1753	MSR_LASTBRANCH_25_TO_IP	
Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DAH, 1754	MSR_LASTBRANCH_26_TO_IP	
Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DBH, 1755	MSR_LASTBRANCH_27_TO_IP	
Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DCH, 1756	MSR_LASTBRANCH_28_TO_IP	
Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DDH, 1757	MSR_LASTBRANCH_29_TO_IP	
Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DEH, 1758	MSR_LASTBRANCH_30_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DFH, 1759	MSR_LASTBRANCH_31_TO_IP	
Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID register (R/O)		Core
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version register (R/O)		Core
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority register (R/W)		Core
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority register (R/O)		Core
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI register (W/O)		Core
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination register (R/O)		Core
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector register (R/W)		Core
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service register bits [31:0] (R/O)		Core
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service register bits [63:32] (R/O)		Core
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service register bits [95:64] (R/O)		Core
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service register bits [127:96] (R/O)		Core
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service register bits [159:128] (R/O)		Core
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service register bits [191:160] (R/O)		Core
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service register bits [223:192] (R/O)		Core
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service register bits [255:224] (R/O)		Core
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode register bits [31:0] (R/O)		Core
Register Address: 819H, 2073	IA32_X2APIC_TMR1	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Trigger Mode register bits [63:32] (R/O)		Core
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode register bits [95:64] (R/O)		Core
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode register bits [127:96] (R/O)		Core
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode register bits [159:128] (R/O)		Core
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode register bits [191:160] (R/O)		Core
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode register bits [223:192] (R/O)		Core
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode register bits [255:224] (R/O)		Core
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request register bits [31:0] (R/O)		Core
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request register bits [63:32] (R/O)		Core
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request register bits [95:64] (R/O)		Core
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request register bits [127:96] (R/O)		Core
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request register bits [159:128] (R/O)		Core
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request register bits [191:160] (R/O)		Core
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request register bits [223:192] (R/O)		Core
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request register bits [255:224] (R/O)		Core
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status register (R/W)		Core
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt register (R/W)		Core
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command register (R/W)		Core
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt register (R/W)		Core

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt register (R/W)		Core
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor register (R/W)		Core
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 register (R/W)		Core
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 register (R/W)		Core
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error register (R/W)		Core
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count register (R/W)		Core
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count register (R/O)		Core
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration register (R/W)		Core
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI register (W/O)		Core
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Core
31:0	Reserved.	
33:32	CLOS (R/W)	
63: 34	Reserved.	
Register Address: D10H, 3344	IA32_L2_QOS_MASK_0	
L2 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	
Register Address: D11H, 3345	IA32_L2_QOS_MASK_1	
L2 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	
Register Address: D12H, 3346	IA32_L2_QOS_MASK_2	
L2 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D13H, 3347	IA32_L2_QOS_MASK_3	
L2 Class Of Service Mask - CLOS 3 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L2 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: D90H, 3472	IA32_BNDCFGS	
See Table 2-2.		Core
Register Address: DA0H, 3488	IA32_XSS	
See Table 2-2.		Core
See Table 2-6, and Table 2-12 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_5CH.		

2.6 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Intel Atom processors based on the Goldmont Plus microarchitecture support MSRs listed in Table 2-6, Table 2-12, and Table 2-13. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_7AH; see Table 2-1. For an MSR listed in Table 2-13 that also appears in the model-specific tables of prior generations, Table 2-13 supersedes prior generation tables.

In the Goldmont Plus microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Goldmont Plus microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Enable VMX inside SMX operation (R/WL)	
2	Enable VMX outside SMX operation (R/WL)	
14:8	SENTER local functions enables (R/WL)	
15	SENTER global functions enable (R/WL)	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Valid if CPUID.07H.00H:ECX[30] = 1.	

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
18	SGX global functions enable (R/WL)	
63:19	Reserved.	
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	
See Table 2-2.		Core
Register Address: 8DH, 141	IA32_SGXLEPUBKEYHASH1	
See Table 2-2.		Core
Register Address: 8EH, 142	IA32_SGXLEPUBKEYHASH2	
See Table 2-2.		Core
Register Address: 8FH, 143	IA32_SGXLEPUBKEYHASH3	
See Table 2-2.		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
(R/W) See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0.	
1	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC1.	
2	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC2.	
3	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC3.	
31:4	Reserved.	
32	Enable PEBS trigger and recording for IA32_FIXED_CTR0.	
33	Enable PEBS trigger and recording for IA32_FIXED_CTR1.	
34	Enable PEBS trigger and recording for IA32_FIXED_CTR2.	
63:35	Reserved.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Core
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
4	PwrEvtEn	
5	FUPonPTW	
6	FabricEn	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
11	DisRETC	
12	PTWEn	
13	BranchEn	
17:14	MTCFreq	
18	Reserved, must be zero.	
22:19	CycThresh	
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. The From_IP part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.7, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture.”		Core
Register Address: 681H–69FH, 1665–1695	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP; <i>i</i> = 1-31.		Core
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. The To_IP part of the stack contains pointers to the Destination instruction. See also: ▪ Section 20.7, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture.”		Core
Register Address: 6C1H–6DFH, 1729–1759	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP; <i>i</i> = 1-31.		Core
Register Address: DCOH, 3520	MSR_LASTBRANCH_INFO_0	
Last Branch Record 0 Additional Information (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. This part of the stack contains flag and elapsed cycle information. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.9.1, “LBR Stack.”		Core
Register Address: DC1H, 3521	MSR_LASTBRANCH_INFO_1	
Last Branch Record 1 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DC2H, 3522	MSR_LASTBRANCH_INFO_2	
Last Branch Record 2 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC3H, 3523	MSR_LASTBRANCH_INFO_3	
Last Branch Record 3 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC4H, 3524	MSR_LASTBRANCH_INFO_4	
Last Branch Record 4 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC5H, 3525	MSR_LASTBRANCH_INFO_5	
Last Branch Record 5 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC6H, 3526	MSR_LASTBRANCH_INFO_6	
Last Branch Record 6 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC7H, 3527	MSR_LASTBRANCH_INFO_7	
Last Branch Record 7 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC8H, 3528	MSR_LASTBRANCH_INFO_8	
Last Branch Record 8 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC9H, 3529	MSR_LASTBRANCH_INFO_9	
Last Branch Record 9 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCAH, 3530	MSR_LASTBRANCH_INFO_10	
Last Branch Record 10 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCBH, 3531	MSR_LASTBRANCH_INFO_11	
Last Branch Record 11 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCCH, 3532	MSR_LASTBRANCH_INFO_12	
Last Branch Record 12 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCDH, 3533	MSR_LASTBRANCH_INFO_13	
Last Branch Record 13 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCEH, 3534	MSR_LASTBRANCH_INFO_14	
Last Branch Record 14 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DCFH, 3535	MSR_LASTBRANCH_INFO_15	
Last Branch Record 15 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDOH, 3536	MSR_LASTBRANCH_INFO_16	
Last Branch Record 16 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD1H, 3537	MSR_LASTBRANCH_INFO_17	
Last Branch Record 17 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD2H, 3538	MSR_LASTBRANCH_INFO_18	
Last Branch Record 18 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD3H, 3539	MSR_LASTBRANCH_INFO_19	
Last Branch Record 19 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD4H, 3520	MSR_LASTBRANCH_INFO_20	
Last Branch Record 20 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD5H, 3521	MSR_LASTBRANCH_INFO_21	
Last Branch Record 21 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD6H, 3522	MSR_LASTBRANCH_INFO_22	
Last Branch Record 22 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD7H, 3523	MSR_LASTBRANCH_INFO_23	
Last Branch Record 23 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD8H, 3524	MSR_LASTBRANCH_INFO_24	
Last Branch Record 24 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD9H, 3525	MSR_LASTBRANCH_INFO_25	
Last Branch Record 25 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDAH, 3526	MSR_LASTBRANCH_INFO_26	
Last Branch Record 26 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDBH, 3527	MSR_LASTBRANCH_INFO_27	
Last Branch Record 27 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DDCH, 3528	MSR_LASTBRANCH_INFO_28	
Last Branch Record 28 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDDH, 3529	MSR_LASTBRANCH_INFO_29	
Last Branch Record 29 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDEH, 3530	MSR_LASTBRANCH_INFO_30	
Last Branch Record 30 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDFH, 3531	MSR_LASTBRANCH_INFO_31	
Last Branch Record 31 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
See Table 2-6, Table 2-12, and Table 2-13 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_7AH.		

2.7 MSRS IN INTEL ATOM® PROCESSORS BASED ON TREMONT MICROARCHITECTURE

Processors based on the Tremont microarchitecture support MSRs listed in Table 2-6, Table 2-12, Table 2-13, and Table 2-14. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_86H, 06_96H, or 06_9CH; see Table 2-1. For an MSR listed in Table 2-14 that also appears in the model-specific tables of prior generations, Table 2-14 supersedes prior generation tables.

In the Tremont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Tremont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
28:0	Reserved.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
30	Reserved.	
31	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
IA32 Core Capabilities Register If CPUID.07H.00H:EDX[30] = 1.		Core
4:0	Reserved.	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL (MSR address 33H).	
63:6	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE: PRMRR BASE Memory Type.	
3	CONFIGURED: PRMRR BASE Configured.	
11:4	Reserved.	
51:12	BASE: PRMRR Base Address.	
63:52	Reserved.	
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
(R/W) See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
$n:0$	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMCx. The maximum value n can be determined from CPUID.0AH:EAX[15:8].	
31: $n+1$	Reserved.	
32: $m:32$	Enable PEBS trigger and recording for IA32_FIXED_CTRx. The maximum value m can be determined from CPUID.0AH:EDX[4:0].	
59:33+m	Reserved.	
60	Pend a PerfMon Interrupt (PMI) after each PEBS event.	
62:61	Specifies PEBS output destination. Encodings: 00B: DS Save Area. 01B: Intel PT trace output. Supported if IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] and CPUID.07H.00H:EBX[25] are set. 10B: Reserved. 11B: Reserved.	
63	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C4H, 5313–5316	MSR_RELOAD_PMCx	

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Reload value for IA32_PMCx (R/W)		Core
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	
See Table 2-6, Table 2-12, Table 2-13, and Table 2-14 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_86H.		

2.8 MSRS IN PROCESSORS BASED ON NEHALEM MICROARCHITECTURE

Table 2-15 lists model-specific registers (MSRs) that are common for Nehalem microarchitecture. These include the Intel Core i7 and i5 processor family. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, 06_1FH, or 06_2EH; see Table 2-1. Additional MSRs specific to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, or 06_1FH are listed in Table 2-16. Some MSRs listed in these tables are used by BIOS. More information about these MSR can be found at <http://biosbits.org>.

The column “Scope” represents the package/core/thread scope of individual bit field of an MSR. “Thread” means this bit field must be programmed on each logical processor independently. “Core” means the bit field must be programmed on each processor core independently, logical processors in the same core will be affected by change of this bit on the other logical processor in the same core. “Package” means the bit field must be programmed once for each physical package. Change of a bit filed with a package scope will affect all logical processors in that physical package.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Thread
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, “MSRs in Pentium Processors.”		Thread
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, “Time-Stamp Counter,” and Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Package
49:0	Reserved.	
52:50	See Table 2-2.	
63:53	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O) Running count of SMI events since last RESET.	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Thread
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Thread
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Programmable TDC-TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDC and TDP Limits for Turbo mode are programmable. When set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 133.33MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
23:16	Reserved.	
24	Interrupt filtering enable (R/W) When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.	
25	C3 state auto demotion enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
26	C1 state auto demotion enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Core
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Thread
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Thread
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Thread
7:0	Event Select	
15:8	UMask	
16	USR	
17	OS	
18	Edge	
19	PC	
20	INT	
21	AnyThread	
22	EN	
23	INV	
31:24	CMASK	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Thread
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Thread
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Core
15:0	Current Performance State Value.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Thread
0	Reserved.	
3:1	On demand Clock Modulation Duty Cycle (R/W)	
4	On demand Clock Modulation Enable (R/W)	
63:5	Reserved.	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Thread
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.	Thread
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Thread
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Thread
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Thread
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM. (R/W) See Table 2-2.	Thread
21:19	Reserved.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Limit CPUID Maxval (R/W) See Table 2-2.	Thread
23	xTPR Message Disable (R/W) See Table 2-2.	Thread
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Thread
37:35	Reserved.	
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Thread
15:0	Reserved.	
23:16	Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
63:24	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	Core
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
3	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	Core
63:4	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .		
0	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.	Package
1	Energy/Performance Bias Enable (R/W) This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.06H:ECX[3].	Thread
63:2	Reserved.	
Register Address: 1ACH, 428	MSR_TURBO_POWER_CURRENT_LIMIT	
See http://biosbits.org .		
14:0	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.	Package
15	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.	Package
30:16	TDC Limit (R/W) TDC limit in 1/8 Amp granularity.	Package
31	TDC Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.	Package
63:32	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
63:2	Reserved.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Thread
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Thread
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Thread
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Thread
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Thread
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Thread
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Thread
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Thread
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Thread
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Thread
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Thread
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Thread
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Thread
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Thread
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Thread
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Thread
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Thread
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Thread
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
See Table 2-2.		Thread
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Thread
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Thread
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Thread
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Thread
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Thread
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Thread
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Thread
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Thread
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Thread
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Thread
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Thread
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Thread
Register Address: 280H, 640	IA32_MC0_CTL2	
See Table 2-2.		Package
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Package
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Core

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Core
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Thread
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Thread
5:0	LBR Format See Table 2-2.	
6	PEBS Record Format	
7	PEBSSaveArchRegs See Table 2-2.	
11:8	PEBS_REC_FORMAT See Table 2-2.	
12	SMM_FREEZE See Table 2-2.	
63:13	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Thread
Register Address: 38EH, 910	MSR_PERF_GLOBAL_STATUS	
Provides single-bit status used by software to query the overflow condition of each performance counter. (R/O)		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
61	UNC_Ovf Uncore overflowed if 1.	
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities." Allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.		Thread
Register Address: 390H, 912	MSR_PERF_GLOBAL_OVF_CTRL	
(R/W)		Thread
61	CLR_UNC_Ovf Set 1 to clear UNC_Ovf.	
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 22.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
0	Enable PEBS on IA32_PMC0 (R/W)	
1	Enable PEBS on IA32_PMC1 (R/W)	
2	Enable PEBS on IA32_PMC2 (R/W)	
3	Enable PEBS on IA32_PMC3 (R/W)	
31:4	Reserved.	
32	Enable Load Latency on IA32_PMC0 (R/W)	
33	Enable Load Latency on IA32_PMC1 (R/W)	
34	Enable Load Latency on IA32_PMC2 (R/W)	
35	Enable Load Latency on IA32_PMC3 (R/W)	
63:36	Reserved.	
Register Address: 3F6H, 1014	MSR_PEBS_LD_LAT	
See Section 22.3.1.1.2, "Load Latency Performance Monitoring Facility."		Thread
15:0	Minimum threshold latency value of tagged load operation that will be counted. (R/W)	
63:36	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package
Register Address: 403H, 1027	IA32_MCO_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 413H, 1043	IA32_MC4_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Core
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Core
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		Thread
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2 and Appendix A.4, "VM-Exit Controls."		Thread
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2 and Appendix A.5, "VM-Entry Controls."		Thread
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2 and Appendix A.6, "Miscellaneous Data."		Thread
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2 and Appendix A.9, "VMCS Enumeration."		Thread
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		Thread
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - See Section 20.9.1 and record format in Section 20.4.8.1.		Thread
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.		Thread
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		Thread
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		Thread
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		Thread
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		Thread
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		Thread
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		Thread
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		Thread
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service Register Bits [31:0] (R/O)		Thread
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service Register Bits [63:32] (R/O)		Thread
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service Register Bits [95:64] (R/O)		Thread
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits [127:96] (R/O)		Thread
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits [159:128] (R/O)		Thread
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits [191:160] (R/O)		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits [223:192] (R/O)		Thread
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits [255:224] (R/O)		Thread
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits [31:0] (R/O)		Thread
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits [63:32] (R/O)		Thread
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits [95:64] (R/O)		Thread
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits [127:96] (R/O)		Thread
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits [159:128] (R/O)		Thread
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits [191:160] (R/O)		Thread
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits [223:192] (R/O)		Thread
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits [255:224] (R/O)		Thread
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request Register Bits [31:0] (R/O)		Thread
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request Register Bits [63:32] (R/O)		Thread
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits [95:64] (R/O)		Thread
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits [127:96] (R/O)		Thread
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits [159:128] (R/O)		Thread
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits [191:160] (R/O)		Thread
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits [223:192] (R/O)		Thread
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits [255:224] (R/O)		Thread
Register Address: 828H, 2088	IA32_X2APIC_ESR	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Error Status Register (R/W)		Thread
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		Thread
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		Thread
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		Thread
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		Thread
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Register (R/W)		Thread
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		Thread
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		Thread
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		Thread
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		Thread
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		Thread
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		Thread
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 20.17.2, "IA32_TSC_AUX Register and RDTSCP Support."		Thread

2.8.1 Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

The Intel Xeon Processor 5500 and 3400 series supports additional model-specific registers listed in Table 2-16. These MSRs also apply to the Intel Core i7 and i5 processor family with a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, or 06_1FH; see Table 2-1.

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Actual maximum turbo frequency is multiplied by 133.33MHz. (Not available in model 06_2EH.)		Package
7:0	Maximum Turbo Ratio Limit 1C (R/O) Maximum Turbo mode ratio limit with 1 core active.	
15:8	Maximum Turbo Ratio Limit 2C (R/O) Maximum Turbo mode ratio limit with 2 cores active.	
23:16	Maximum Turbo Ratio Limit 3C (R/O) Maximum Turbo mode ratio limit with 3 cores active.	
31:24	Maximum Turbo Ratio Limit 4C (R/O) Maximum Turbo mode ratio limit with 4 cores active.	
63:32	Reserved.	
Register Address: 301H, 769	MSR_GQ_SNOOP_MESF	
MSR_GQ_SNOOP_MESF		Package
0	From M to S (R/W)	
1	From E to S (R/W)	
2	From S to S (R/W)	
3	From F to S (R/W)	
4	From M to I (R/W)	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
5	From E to I (R/W)	
6	From S to I (R/W)	
7	From F to I (R/W)	
63:8	Reserved.	
Register Address: 391H, 913	MSR_UNCORE_PERF_GLOBAL_CTRL	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 392H, 914	MSR_UNCORE_PERF_GLOBAL_STATUS	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 393H, 915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 394H, 916	MSR_UNCORE_FIXED_CTR0	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 395H, 917	MSR_UNCORE_FIXED_CTR_CTRL	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 396H, 918	MSR_UNCORE_ADDR_OPCODE_MATCH	
See Section 22.3.1.2.3, "Uncore Address/Opcode Match MSR."		Package
Register Address: 3B0H, 960	MSR_UNCORE_PMC0	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B1H, 961	MSR_UNCORE_PMC1	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B2H, 962	MSR_UNCORE_PMC2	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B3H, 963	MSR_UNCORE_PMC3	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B4H, 964	MSR_UNCORE_PMC4	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B5H, 965	MSR_UNCORE_PMC5	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B6H, 966	MSR_UNCORE_PMC6	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B7H, 967	MSR_UNCORE_PMC7	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C0H, 944	MSR_UNCORE_PERFEVTSEL0	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C1H, 945	MSR_UNCORE_PERFEVTSEL1	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C2H, 946	MSR_UNCORE_PERFEVTSEL2	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C3H, 947	MSR_UNCORE_PERFEVTSEL3	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C4H, 948	MSR_UNCORE_PERFEVTSEL4	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C5H, 949	MSR_UNCORE_PERFEVTSEL5	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C6H, 950	MSR_UNCORE_PERFEVTSEL6	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C7H, 951	MSR_UNCORE_PERFEVTSEL7	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package

2.8.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

The Intel Xeon Processor 7500 series supports MSRs listed in Table 2-15 (except MSR address 1ADH) and additional model-specific registers listed in Table 2-17. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2EH.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Reserved. Attempt to read/write will cause #UD.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 394H, 816	MSR_W_PMON_FIXED_CTR	
Uncore W-box PerfMon fixed counter.		Package
Register Address: 395H, 817	MSR_W_PMON_FIXED_CTR_CTL	
Uncore U-box PerfMon fixed counter control MSR.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 454H, 1108	IA32_MC21_CTL	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: C00H, 3072	MSR_U_PMON_GLOBAL_CTRL	
Uncore U-box PerfMon global control MSR.		Package
Register Address: C01H, 3073	MSR_U_PMON_GLOBAL_STATUS	
Uncore U-box PerfMon global status MSR.		Package
Register Address: C02H, 3074	MSR_U_PMON_GLOBAL_OVF_CTRL	
Uncore U-box PerfMon global overflow control MSR.		Package
Register Address: C10H, 3088	MSR_U_PMON_EVNT_SEL	
Uncore U-box PerfMon event select MSR.		Package
Register Address: C11H, 3089	MSR_U_PMON_CTR	
Uncore U-box PerfMon counter MSR.		Package
Register Address: C20H, 3104	MSR_B0_PMON_BOX_CTRL	
Uncore B-box 0 PerfMon local box control MSR.		Package
Register Address: C21H, 3105	MSR_B0_PMON_BOX_STATUS	
Uncore B-box 0 PerfMon local box status MSR.		Package
Register Address: C22H, 3106	MSR_B0_PMON_BOX_OVF_CTRL	
Uncore B-box 0 PerfMon local box overflow control MSR.		Package
Register Address: C30H, 3120	MSR_B0_PMON_EVNT_SELO	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C31H, 3121	MSR_B0_PMON_CTR0	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C32H, 3122	MSR_B0_PMON_EVNT_SEL1	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C33H, 3123	MSR_B0_PMON_CTR1	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C34H, 3124	MSR_B0_PMON_EVNT_SEL2	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C35H, 3125	MSR_B0_PMON_CTR2	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C36H, 3126	MSR_B0_PMON_EVNT_SEL3	
Uncore B-box 0 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C37H, 3127	MSR_B0_PMON_CTR3	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C40H, 3136	MSR_S0_PMON_BOX_CTRL	
Uncore S-box 0 PerfMon local box control MSR.		Package
Register Address: C41H, 3137	MSR_S0_PMON_BOX_STATUS	
Uncore S-box 0 PerfMon local box status MSR.		Package
Register Address: C42H, 3138	MSR_S0_PMON_BOX_OVF_CTRL	
Uncore S-box 0 PerfMon local box overflow control MSR.		Package
Register Address: C50H, 3152	MSR_S0_PMON_EVNT_SELO	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C51H, 3153	MSR_S0_PMON_CTR0	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C52H, 3154	MSR_S0_PMON_EVNT_SEL1	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C53H, 3155	MSR_S0_PMON_CTR1	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C54H, 3156	MSR_S0_PMON_EVNT_SEL2	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C55H, 3157	MSR_S0_PMON_CTR2	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C56H, 3158	MSR_S0_PMON_EVNT_SEL3	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C57H, 3159	MSR_S0_PMON_CTR3	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C60H, 3168	MSR_B1_PMON_BOX_CTRL	
Uncore B-box 1 PerfMon local box control MSR.		Package
Register Address: C61H, 3169	MSR_B1_PMON_BOX_STATUS	
Uncore B-box 1 PerfMon local box status MSR.		Package
Register Address: C62H, 3170	MSR_B1_PMON_BOX_OVF_CTRL	
Uncore B-box 1 PerfMon local box overflow control MSR.		Package
Register Address: C70H, 3184	MSR_B1_PMON_EVNT_SELO	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C71H, 3185	MSR_B1_PMON_CTR0	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C72H, 3186	MSR_B1_PMON_EVNT_SEL1	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C73H, 3187	MSR_B1_PMON_CTR1	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C74H, 3188	MSR_B1_PMON_EVNT_SEL2	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C75H, 3189	MSR_B1_PMON_CTRL2	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C76H, 3190	MSR_B1_PMON_EVNT_SEL3	
Uncore B-box 1vPerfMon event select MSR.		Package
Register Address: C77H, 3191	MSR_B1_PMON_CTRL3	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C80H, 3120	MSR_W_PMON_BOX_CTRL	
Uncore W-box PerfMon local box control MSR.		Package
Register Address: C81H, 3121	MSR_W_PMON_BOX_STATUS	
Uncore W-box PerfMon local box status MSR.		Package
Register Address: C82H, 3122	MSR_W_PMON_BOX_OVF_CTRL	
Uncore W-box PerfMon local box overflow control MSR.		Package
Register Address: C90H, 3136	MSR_W_PMON_EVNT_SEL0	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C91H, 3137	MSR_W_PMON_CTRL0	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C92H, 3138	MSR_W_PMON_EVNT_SEL1	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C93H, 3139	MSR_W_PMON_CTRL1	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C94H, 3140	MSR_W_PMON_EVNT_SEL2	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C95H, 3141	MSR_W_PMON_CTRL2	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C96H, 3142	MSR_W_PMON_EVNT_SEL3	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C97H, 3143	MSR_W_PMON_CTRL3	
Uncore W-box PerfMon counter MSR.		Package
Register Address: CA0H, 3232	MSR_M0_PMON_BOX_CTRL	
Uncore M-box 0 PerfMon local box control MSR.		Package
Register Address: CA1H, 3233	MSR_M0_PMON_BOX_STATUS	
Uncore M-box 0 PerfMon local box status MSR.		Package
Register Address: CA2H, 3234	MSR_M0_PMON_BOX_OVF_CTRL	
Uncore M-box 0 PerfMon local box overflow control MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CA4H, 3236	MSR_M0_PMON_TIMESTAMP	
Uncore M-box 0 PerfMon time stamp unit select MSR.		Package
Register Address: CA5H, 3237	MSR_M0_PMON_DSP	
Uncore M-box 0 PerfMon DSP unit select MSR.		Package
Register Address: CA6H, 3238	MSR_M0_PMON_ISS	
Uncore M-box 0 PerfMon ISS unit select MSR.		Package
Register Address: CA7H, 3239	MSR_M0_PMON_MAP	
Uncore M-box 0 PerfMon MAP unit select MSR.		Package
Register Address: CA8H, 3240	MSR_M0_PMON_MSC_THR	
Uncore M-box 0 PerfMon MIC THR select MSR.		Package
Register Address: CA9H, 3241	MSR_M0_PMON_PGT	
Uncore M-box 0 PerfMon PGT unit select MSR.		Package
Register Address: CAAH, 3242	MSR_M0_PMON_PLD	
Uncore M-box 0 PerfMon PLD unit select MSR.		Package
Register Address: CABH, 3243	MSR_M0_PMON_ZDP	
Uncore M-box 0 PerfMon ZDP unit select MSR.		Package
Register Address: CBOH, 3248	MSR_M0_PMON_EVNT_SELO	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB1H, 3249	MSR_M0_PMON_CTR0	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB2H, 3250	MSR_M0_PMON_EVNT_SEL1	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB3H, 3251	MSR_M0_PMON_CTR1	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB4H, 3252	MSR_M0_PMON_EVNT_SEL2	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB5H, 3253	MSR_M0_PMON_CTR2	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB6H, 3254	MSR_M0_PMON_EVNT_SEL3	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB7H, 3255	MSR_M0_PMON_CTR3	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB8H, 3256	MSR_M0_PMON_EVNT_SEL4	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB9H, 3257	MSR_M0_PMON_CTR4	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CBAH, 3258	MSR_M0_PMON_EVNT_SEL5	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CBBH, 3259	MSR_M0_PMON_CTR5	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CC0H, 3264	MSR_S1_PMON_BOX_CTRL	
Uncore S-box 1 PerfMon local box control MSR.		Package
Register Address: CC1H, 3265	MSR_S1_PMON_BOX_STATUS	
Uncore S-box 1 PerfMon local box status MSR.		Package
Register Address: CC2H, 3266	MSR_S1_PMON_BOX_OVF_CTRL	
Uncore S-box 1 PerfMon local box overflow control MSR.		Package
Register Address: CD0H, 3280	MSR_S1_PMON_EVNT_SELO	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD1H, 3281	MSR_S1_PMON_CTR0	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD2H, 3282	MSR_S1_PMON_EVNT_SEL1	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD3H, 3283	MSR_S1_PMON_CTR1	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD4H, 3284	MSR_S1_PMON_EVNT_SEL2	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD5H, 3285	MSR_S1_PMON_CTR2	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD6H, 3286	MSR_S1_PMON_EVNT_SEL3	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD7H, 3287	MSR_S1_PMON_CTR3	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CE0H, 3296	MSR_M1_PMON_BOX_CTRL	
Uncore M-box 1 PerfMon local box control MSR.		Package
Register Address: CE1H, 3297	MSR_M1_PMON_BOX_STATUS	
Uncore M-box 1 PerfMon local box status MSR.		Package
Register Address: CE2H, 3298	MSR_M1_PMON_BOX_OVF_CTRL	
Uncore M-box 1 PerfMon local box overflow control MSR.		Package
Register Address: CE4H, 3300	MSR_M1_PMON_TIMESTAMP	
Uncore M-box 1 PerfMon time stamp unit select MSR.		Package
Register Address: CE5H, 3301	MSR_M1_PMON_DSP	
Uncore M-box 1 PerfMon DSP unit select MSR.		Package
Register Address: CE6H, 3302	MSR_M1_PMON_ISS	
Uncore M-box 1 PerfMon ISS unit select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CE7H, 3303	MSR_M1_PMON_MAP	
Uncore M-box 1 PerfMon MAP unit select MSR.		Package
Register Address: CE8H, 3304	MSR_M1_PMON_MSC_THR	
Uncore M-box 1 PerfMon MIC THR select MSR.		Package
Register Address: CE9H, 3305	MSR_M1_PMON_PGT	
Uncore M-box 1 PerfMon PGT unit select MSR.		Package
Register Address: CEAH, 3306	MSR_M1_PMON_PLD	
Uncore M-box 1 PerfMon PLD unit select MSR.		Package
Register Address: CEBH, 3307	MSR_M1_PMON_ZDP	
Uncore M-box 1 PerfMon ZDP unit select MSR.		Package
Register Address: CF0H, 3312	MSR_M1_PMON_EVNT_SELO	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF1H, 3313	MSR_M1_PMON_CTR0	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF2H, 3314	MSR_M1_PMON_EVNT_SEL1	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF3H, 3315	MSR_M1_PMON_CTR1	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF4H, 3316	MSR_M1_PMON_EVNT_SEL2	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF5H, 3317	MSR_M1_PMON_CTR2	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF6H, 3318	MSR_M1_PMON_EVNT_SEL3	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF7H, 3319	MSR_M1_PMON_CTR3	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF8H, 3320	MSR_M1_PMON_EVNT_SEL4	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF9H, 3321	MSR_M1_PMON_CTR4	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CFAH, 3322	MSR_M1_PMON_EVNT_SEL5	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CFBH, 3323	MSR_M1_PMON_CTR5	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: D00H, 3328	MSR_CO_PMON_BOX_CTRL	
Uncore C-box 0 PerfMon local box control MSR.		Package
Register Address: D01H, 3329	MSR_CO_PMON_BOX_STATUS	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 0 PerfMon local box status MSR.		Package
Register Address: D02H, 3330	MSR_CO_PMON_BOX_OVF_CTRL	
Uncore C-box 0 PerfMon local box overflow control MSR.		Package
Register Address: D10H, 3344	MSR_CO_PMON_EVNT_SELO	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D11H, 3345	MSR_CO_PMON_CTR0	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D12H, 3346	MSR_CO_PMON_EVNT_SEL1	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D13H, 3347	MSR_CO_PMON_CTR1	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D14H, 3348	MSR_CO_PMON_EVNT_SEL2	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D15H, 3349	MSR_CO_PMON_CTR2	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D16H, 3350	MSR_CO_PMON_EVNT_SEL3	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D17H, 3351	MSR_CO_PMON_CTR3	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D18H, 3352	MSR_CO_PMON_EVNT_SEL4	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D19H, 3353	MSR_CO_PMON_CTR4	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D1AH, 3354	MSR_CO_PMON_EVNT_SEL5	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D1BH, 3355	MSR_CO_PMON_CTR5	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D20H, 3360	MSR_C4_PMON_BOX_CTRL	
Uncore C-box 4 PerfMon local box control MSR.		Package
Register Address: D21H, 3361	MSR_C4_PMON_BOX_STATUS	
Uncore C-box 4 PerfMon local box status MSR.		Package
Register Address: D22H, 3362	MSR_C4_PMON_BOX_OVF_CTRL	
Uncore C-box 4 PerfMon local box overflow control MSR.		Package
Register Address: D30H, 3376	MSR_C4_PMON_EVNT_SELO	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D31H, 3377	MSR_C4_PMON_CTR0	
Uncore C-box 4 PerfMon counter MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D32H, 3378	MSR_C4_PMON_EVNT_SEL1	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D33H, 3379	MSR_C4_PMON_CTR1	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D34H, 3380	MSR_C4_PMON_EVNT_SEL2	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D35H, 3381	MSR_C4_PMON_CTR2	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D36H, 3382	MSR_C4_PMON_EVNT_SEL3	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D37H, 3383	MSR_C4_PMON_CTR3	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D38H, 3384	MSR_C4_PMON_EVNT_SEL4	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D39H, 3385	MSR_C4_PMON_CTR4	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D3AH, 3386	MSR_C4_PMON_EVNT_SEL5	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D3BH, 3387	MSR_C4_PMON_CTR5	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D40H, 3392	MSR_C2_PMON_BOX_CTRL	
Uncore C-box 2 PerfMon local box control MSR.		Package
Register Address: D41H, 3393	MSR_C2_PMON_BOX_STATUS	
Uncore C-box 2 PerfMon local box status MSR.		Package
Register Address: D42H, 3394	MSR_C2_PMON_BOX_OVF_CTRL	
Uncore C-box 2 PerfMon local box overflow control MSR.		Package
Register Address: D50H, 3408	MSR_C2_PMON_EVNT_SELO	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D51H, 3409	MSR_C2_PMON_CTR0	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D52H, 3410	MSR_C2_PMON_EVNT_SEL1	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D53H, 3411	MSR_C2_PMON_CTR1	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D54H, 3412	MSR_C2_PMON_EVNT_SEL2	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D55H, 3413	MSR_C2_PMON_CTR2	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D56H, 3414	MSR_C2_PMON_EVNT_SEL3	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D57H, 3415	MSR_C2_PMON_CTR3	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D58H, 3416	MSR_C2_PMON_EVNT_SEL4	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D59H, 3417	MSR_C2_PMON_CTR4	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D5AH, 3418	MSR_C2_PMON_EVNT_SEL5	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D5BH, 3419	MSR_C2_PMON_CTR5	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D60H, 3424	MSR_C6_PMON_BOX_CTRL	
Uncore C-box 6 PerfMon local box control MSR.		Package
Register Address: D61H, 3425	MSR_C6_PMON_BOX_STATUS	
Uncore C-box 6 PerfMon local box status MSR.		Package
Register Address: D62H, 3426	MSR_C6_PMON_BOX_OVF_CTRL	
Uncore C-box 6 PerfMon local box overflow control MSR.		Package
Register Address: D70H, 3440	MSR_C6_PMON_EVNT_SELO	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D71H, 3441	MSR_C6_PMON_CTR0	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D72H, 3442	MSR_C6_PMON_EVNT_SEL1	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D73H, 3443	MSR_C6_PMON_CTR1	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D74H, 3444	MSR_C6_PMON_EVNT_SEL2	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D75H, 3445	MSR_C6_PMON_CTR2	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D76H, 3446	MSR_C6_PMON_EVNT_SEL3	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D77H, 3447	MSR_C6_PMON_CTR3	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D78H, 3448	MSR_C6_PMON_EVNT_SEL4	
Uncore C-box 6 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D79H, 3449	MSR_C6_PMON_CTR4	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D7AH, 3450	MSR_C6_PMON_EVNT_SEL5	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D7BH, 3451	MSR_C6_PMON_CTR5	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D80H, 3456	MSR_C1_PMON_BOX_CTRL	
Uncore C-box 1 PerfMon local box control MSR.		Package
Register Address: D81H, 3457	MSR_C1_PMON_BOX_STATUS	
Uncore C-box 1 PerfMon local box status MSR.		Package
Register Address: D82H, 3458	MSR_C1_PMON_BOX_OVF_CTRL	
Uncore C-box 1 PerfMon local box overflow control MSR.		Package
Register Address: D90H, 3472	MSR_C1_PMON_EVNT_SELO	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D91H, 3473	MSR_C1_PMON_CTR0	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D92H, 3474	MSR_C1_PMON_EVNT_SEL1	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D93H, 3475	MSR_C1_PMON_CTR1	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D94H, 3476	MSR_C1_PMON_EVNT_SEL2	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D95H, 3477	MSR_C1_PMON_CTR2	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D96H, 3478	MSR_C1_PMON_EVNT_SEL3	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D97H, 3479	MSR_C1_PMON_CTR3	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D98H, 3480	MSR_C1_PMON_EVNT_SEL4	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D99H, 3481	MSR_C1_PMON_CTR4	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D9AH, 3482	MSR_C1_PMON_EVNT_SEL5	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D9BH, 3483	MSR_C1_PMON_CTR5	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: DA0H, 3488	MSR_C5_PMON_BOX_CTRL	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 5 PerfMon local box control MSR.		Package
Register Address: DA1H, 3489	MSR_C5_PMON_BOX_STATUS	
Uncore C-box 5 PerfMon local box status MSR.		Package
Register Address: DA2H, 3490	MSR_C5_PMON_BOX_OVF_CTRL	
Uncore C-box 5 PerfMon local box overflow control MSR.		Package
Register Address: DB0H, 3504	MSR_C5_PMON_EVNT_SELO	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB1H, 3505	MSR_C5_PMON_CTR0	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB2H, 3506	MSR_C5_PMON_EVNT_SEL1	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB3H, 3507	MSR_C5_PMON_CTR1	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB4H, 3508	MSR_C5_PMON_EVNT_SEL2	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB5H, 3509	MSR_C5_PMON_CTR2	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB6H, 3510	MSR_C5_PMON_EVNT_SEL3	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB7H, 3511	MSR_C5_PMON_CTR3	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB8H, 3512	MSR_C5_PMON_EVNT_SEL4	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB9H, 3513	MSR_C5_PMON_CTR4	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DBAH, 3514	MSR_C5_PMON_EVNT_SEL5	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DBBH, 3515	MSR_C5_PMON_CTR5	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DC0H, 3520	MSR_C3_PMON_BOX_CTRL	
Uncore C-box 3 PerfMon local box control MSR.		Package
Register Address: DC1H, 3521	MSR_C3_PMON_BOX_STATUS	
Uncore C-box 3 PerfMon local box status MSR.		Package
Register Address: DC2H, 3522	MSR_C3_PMON_BOX_OVF_CTRL	
Uncore C-box 3 PerfMon local box overflow control MSR.		Package
Register Address: DDOH, 3536	MSR_C3_PMON_EVNT_SELO	
Uncore C-box 3 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DD1H, 3537	MSR_C3_PMON_CTRL0	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD2H, 3538	MSR_C3_PMON_EVNT_SEL1	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD3H, 3539	MSR_C3_PMON_CTRL1	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD4H, 3540	MSR_C3_PMON_EVNT_SEL2	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD5H, 3541	MSR_C3_PMON_CTRL2	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD6H, 3542	MSR_C3_PMON_EVNT_SEL3	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD7H, 3543	MSR_C3_PMON_CTRL3	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD8H, 3544	MSR_C3_PMON_EVNT_SEL4	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD9H, 3545	MSR_C3_PMON_CTRL4	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DDAH, 3546	MSR_C3_PMON_EVNT_SEL5	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DDBH, 3547	MSR_C3_PMON_CTRL5	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DE0H, 3552	MSR_C7_PMON_BOX_CTRL	
Uncore C-box 7 PerfMon local box control MSR.		Package
Register Address: DE1H, 3553	MSR_C7_PMON_BOX_STATUS	
Uncore C-box 7 PerfMon local box status MSR.		Package
Register Address: DE2H, 3554	MSR_C7_PMON_BOX_OVF_CTRL	
Uncore C-box 7 PerfMon local box overflow control MSR.		Package
Register Address: DF0H, 3568	MSR_C7_PMON_EVNT_SELO	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF1H, 3569	MSR_C7_PMON_CTRL0	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF2H, 3570	MSR_C7_PMON_EVNT_SEL1	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF3H, 3571	MSR_C7_PMON_CTRL1	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF4H, 3572	MSR_C7_PMON_EVNT_SEL2	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF5H, 3573	MSR_C7_PMON_CTR2	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF6H, 3574	MSR_C7_PMON_EVNT_SEL3	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF7H, 3575	MSR_C7_PMON_CTR3	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF8H, 3576	MSR_C7_PMON_EVNT_SEL4	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF9H, 3577	MSR_C7_PMON_CTR4	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DFAH, 3578	MSR_C7_PMON_EVNT_SEL5	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DFBH, 3579	MSR_C7_PMON_CTR5	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: E00H, 3584	MSR_R0_PMON_BOX_CTRL	
Uncore R-box 0 PerfMon local box control MSR.		Package
Register Address: E01H, 3585	MSR_R0_PMON_BOX_STATUS	
Uncore R-box 0 PerfMon local box status MSR.		Package
Register Address: E02H, 3586	MSR_R0_PMON_BOX_OVF_CTRL	
Uncore R-box 0 PerfMon local box overflow control MSR.		Package
Register Address: E04H, 3588	MSR_R0_PMON_IPERF0_P0	
Uncore R-box 0 PerfMon IPERF0 unit Port 0 select MSR.		Package
Register Address: E05H, 3589	MSR_R0_PMON_IPERF0_P1	
Uncore R-box 0 PerfMon IPERF0 unit Port 1 select MSR.		Package
Register Address: E06H, 3590	MSR_R0_PMON_IPERF0_P2	
Uncore R-box 0 PerfMon IPERF0 unit Port 2 select MSR.		Package
Register Address: E07H, 3591	MSR_R0_PMON_IPERF0_P3	
Uncore R-box 0 PerfMon IPERF0 unit Port 3 select MSR.		Package
Register Address: E08H, 3592	MSR_R0_PMON_IPERF0_P4	
Uncore R-box 0 PerfMon IPERF0 unit Port 4 select MSR.		Package
Register Address: E09H, 3593	MSR_R0_PMON_IPERF0_P5	
Uncore R-box 0 PerfMon IPERF0 unit Port 5 select MSR.		Package
Register Address: E0AH, 3594	MSR_R0_PMON_IPERF0_P6	
Uncore R-box 0 PerfMon IPERF0 unit Port 6 select MSR.		Package
Register Address: E0BH, 3595	MSR_R0_PMON_IPERF0_P7	
Uncore R-box 0 PerfMon IPERF0 unit Port 7 select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E0CH, 3596	MSR_R0_PMON_QLX_P0	
Uncore R-box 0 PerfMon QLX unit Port 0 select MSR.		Package
Register Address: E0DH, 3597	MSR_R0_PMON_QLX_P1	
Uncore R-box 0 PerfMon QLX unit Port 1 select MSR.		Package
Register Address: E0EH, 3598	MSR_R0_PMON_QLX_P2	
Uncore R-box 0 PerfMon QLX unit Port 2 select MSR.		Package
Register Address: E0FH, 3599	MSR_R0_PMON_QLX_P3	
Uncore R-box 0 PerfMon QLX unit Port 3 select MSR.		Package
Register Address: E10H, 3600	MSR_R0_PMON_EVNT_SELO	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E11H, 3601	MSR_R0_PMON_CTR0	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E12H, 3602	MSR_R0_PMON_EVNT_SEL1	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E13H, 3603	MSR_R0_PMON_CTR1	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E14H, 3604	MSR_R0_PMON_EVNT_SEL2	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E15H, 3605	MSR_R0_PMON_CTR2	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E16H, 3606	MSR_R0_PMON_EVNT_SEL3	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E17H, 3607	MSR_R0_PMON_CTR3	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E18H, 3608	MSR_R0_PMON_EVNT_SEL4	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E19H, 3609	MSR_R0_PMON_CTR4	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1AH, 3610	MSR_R0_PMON_EVNT_SEL5	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1BH, 3611	MSR_R0_PMON_CTR5	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1CH, 3612	MSR_R0_PMON_EVNT_SEL6	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1DH, 3613	MSR_R0_PMON_CTR6	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1EH, 3614	MSR_R0_PMON_EVNT_SEL7	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1FH, 3615	MSR_R0_PMON_CTR7	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E20H, 3616	MSR_R1_PMON_BOX_CTRL	
Uncore R-box 1 PerfMon local box control MSR.		Package
Register Address: E21H, 3617	MSR_R1_PMON_BOX_STATUS	
Uncore R-box 1 PerfMon local box status MSR.		Package
Register Address: E22H, 3618	MSR_R1_PMON_BOX_OVF_CTRL	
Uncore R-box 1 PerfMon local box overflow control MSR.		Package
Register Address: E24H, 3620	MSR_R1_PMON_IPERF1_P8	
Uncore R-box 1 PerfMon IPERF1 unit Port 8 select MSR.		Package
Register Address: E25H, 3621	MSR_R1_PMON_IPERF1_P9	
Uncore R-box 1 PerfMon IPERF1 unit Port 9 select MSR.		Package
Register Address: E26H, 3622	MSR_R1_PMON_IPERF1_P10	
Uncore R-box 1 PerfMon IPERF1 unit Port 10 select MSR.		Package
Register Address: E27H, 3623	MSR_R1_PMON_IPERF1_P11	
Uncore R-box 1 PerfMon IPERF1 unit Port 11 select MSR.		Package
Register Address: E28H, 3624	MSR_R1_PMON_IPERF1_P12	
Uncore R-box 1 PerfMon IPERF1 unit Port 12 select MSR.		Package
Register Address: E29H, 3625	MSR_R1_PMON_IPERF1_P13	
Uncore R-box 1 PerfMon IPERF1 unit Port 13 select MSR.		Package
Register Address: E2AH, 3626	MSR_R1_PMON_IPERF1_P14	
Uncore R-box 1 PerfMon IPERF1 unit Port 14 select MSR.		Package
Register Address: E2BH, 3627	MSR_R1_PMON_IPERF1_P15	
Uncore R-box 1 PerfMon IPERF1 unit Port 15 select MSR.		Package
Register Address: E2CH, 3628	MSR_R1_PMON_QLX_P4	
Uncore R-box 1 PerfMon QLX unit Port 4 select MSR.		Package
Register Address: E2DH, 3629	MSR_R1_PMON_QLX_P5	
Uncore R-box 1 PerfMon QLX unit Port 5 select MSR.		Package
Register Address: E2EH, 3630	MSR_R1_PMON_QLX_P6	
Uncore R-box 1 PerfMon QLX unit Port 6 select MSR.		Package
Register Address: E2FH, 3631	MSR_R1_PMON_QLX_P7	
Uncore R-box 1 PerfMon QLX unit Port 7 select MSR.		Package
Register Address: E30H, 3632	MSR_R1_PMON_EVNT_SEL8	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E31H, 3633	MSR_R1_PMON_CTR8	
Uncore R-box 1 PerfMon counter MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E32H, 3634	MSR_R1_PMON_EVNT_SEL9	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E33H, 3635	MSR_R1_PMON_CTR9	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E34H, 3636	MSR_R1_PMON_EVNT_SEL10	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E35H, 3637	MSR_R1_PMON_CTR10	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E36H, 3638	MSR_R1_PMON_EVNT_SEL11	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E37H, 3639	MSR_R1_PMON_CTR11	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E38H, 3640	MSR_R1_PMON_EVNT_SEL12	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E39H, 3641	MSR_R1_PMON_CTR12	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3AH, 3642	MSR_R1_PMON_EVNT_SEL13	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3BH, 3643	MSR_R1_PMON_CTR13	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3CH, 3644	MSR_R1_PMON_EVNT_SEL14	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3DH, 3645	MSR_R1_PMON_CTR14	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3EH, 3646	MSR_R1_PMON_EVNT_SEL15	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3FH, 3647	MSR_R1_PMON_CTR15	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E45H, 3653	MSR_B0_PMON_MATCH	
Uncore B-box 0 PerfMon local box match MSR.		Package
Register Address: E46H, 3654	MSR_B0_PMON_MASK	
Uncore B-box 0 PerfMon local box mask MSR.		Package
Register Address: E49H, 3657	MSR_S0_PMON_MATCH	
Uncore S-box 0 PerfMon local box match MSR.		Package
Register Address: E4AH, 3658	MSR_S0_PMON_MASK	
Uncore S-box 0 PerfMon local box mask MSR.		Package
Register Address: E4DH, 3661	MSR_B1_PMON_MATCH	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore B-box 1 PerfMon local box match MSR.		Package
Register Address: E4EH, 3662	MSR_B1_PMON_MASK	
Uncore B-box 1 PerfMon local box mask MSR.		Package
Register Address: E54H, 3668	MSR_M0_PMON_MM_CONFIG	
Uncore M-box 0 PerfMon local box address match/mask config MSR.		Package
Register Address: E55H, 3669	MSR_M0_PMON_ADDR_MATCH	
Uncore M-box 0 PerfMon local box address match MSR.		Package
Register Address: E56H, 3670	MSR_M0_PMON_ADDR_MASK	
Uncore M-box 0 PerfMon local box address mask MSR.		Package
Register Address: E59H, 3673	MSR_S1_PMON_MATCH	
Uncore S-box 1 PerfMon local box match MSR.		Package
Register Address: E5AH, 3674	MSR_S1_PMON_MASK	
Uncore S-box 1 PerfMon local box mask MSR.		Package
Register Address: E5CH, 3676	MSR_M1_PMON_MM_CONFIG	
Uncore M-box 1 PerfMon local box address match/mask config MSR.		Package
Register Address: E5DH, 3677	MSR_M1_PMON_ADDR_MATCH	
Uncore M-box 1 PerfMon local box address match MSR.		Package
Register Address: E5EH, 3678	MSR_M1_PMON_ADDR_MASK	
Uncore M-box 1 PerfMon local box address mask MSR.		Package
Register Address: 3B5H, 965	MSR_UNCORE_PMC5	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package

2.9 MSRS IN THE INTEL® XEON® PROCESSOR 5600 SERIES BASED ON WESTMERE MICROARCHITECTURE

The Intel® Xeon® Processor 5600 Series is based on Westmere microarchitecture and supports the MSR interfaces listed in Table 2-15, Table 2-16, plus additional MSRs listed in Table 2-18. These MSRs apply to the Intel Core i7, i5, and i3 processor family with a CPUID Signature DisplayFamily_DisplayModel value of 06_25H or 06_2CH; see Table 2-1.

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
63:48	Reserved.	
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package

2.10 MSRS IN THE INTEL® XEON® PROCESSOR E7 FAMILY BASED ON WESTMERE MICROARCHITECTURE

The Intel® Xeon® Processor E7 Family is based on the Westmere microarchitecture and supports the MSR interfaces listed in Table 2-15 (except MSR address 1ADH), Table 2-16, plus additional MSRs listed in Table 2-19. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2FH.

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Reserved. Attempt to read/write will cause #UD.		Package
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package
Register Address: F40H, 3904	MSR_C8_PMON_BOX_CTRL	
Uncore C-box 8 PerfMon local box control MSR.		Package
Register Address: F41H, 3905	MSR_C8_PMON_BOX_STATUS	
Uncore C-box 8 PerfMon local box status MSR.		Package
Register Address: F42H, 3906	MSR_C8_PMON_BOX_OVF_CTRL	
Uncore C-box 8 PerfMon local box overflow control MSR.		Package
Register Address: F50H, 3920	MSR_C8_PMON_EVNT_SELO	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F51H, 3921	MSR_C8_PMON_CTR0	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F52H, 3922	MSR_C8_PMON_EVNT_SEL1	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F53H, 3923	MSR_C8_PMON_CTR1	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F54H, 3924	MSR_C8_PMON_EVNT_SEL2	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F55H, 3925	MSR_C8_PMON_CTR2	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F56H, 3926	MSR_C8_PMON_EVNT_SEL3	
Uncore C-box 8 PerfMon event select MSR.		Package

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: F57H, 3927	MSR_C8_PMON_CTR3	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F58H, 3928	MSR_C8_PMON_EVNT_SEL4	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F59H, 3929	MSR_C8_PMON_CTR4	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F5AH, 3930	MSR_C8_PMON_EVNT_SEL5	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F5BH, 3931	MSR_C8_PMON_CTR5	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: FC0H, 4032	MSR_C9_PMON_BOX_CTRL	
Uncore C-box 9 PerfMon local box control MSR.		Package
Register Address: FC1H, 4033	MSR_C9_PMON_BOX_STATUS	
Uncore C-box 9 PerfMon local box status MSR.		Package
Register Address: FC2H, 4034	MSR_C9_PMON_BOX_OVF_CTRL	
Uncore C-box 9 PerfMon local box overflow control MSR.		Package
Register Address: FDOH, 4048	MSR_C9_PMON_EVNT_SELO	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD1H, 4049	MSR_C9_PMON_CTR0	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD2H, 4050	MSR_C9_PMON_EVNT_SEL1	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD3H, 4051	MSR_C9_PMON_CTR1	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD4H, 4052	MSR_C9_PMON_EVNT_SEL2	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD5H, 4053	MSR_C9_PMON_CTR2	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD6H, 4054	MSR_C9_PMON_EVNT_SEL3	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD7H, 4055	MSR_C9_PMON_CTR3	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD8H, 4056	MSR_C9_PMON_EVNT_SEL4	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD9H, 4057	MSR_C9_PMON_CTR4	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FDAH, 4058	MSR_C9_PMON_EVNT_SEL5	

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FDBH, 4059	MSR_C9_PMON_CTR5	
Uncore C-box 9 PerfMon counter MSR.		Package

2.11 MSRS IN THE INTEL® PROCESSOR FAMILY BASED ON SANDY BRIDGE MICROARCHITECTURE

Table 2-20 lists model-specific registers (MSRs) that are common to the Intel® processor family based on Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH or 06_2DH; see Table 2-1. Additional MSRs specific to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH are listed in Table 2-21.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, "Time-Stamp Counter," and see Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O) Count SMIs.	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Thread
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Thread
Register Address: C5H, 197	IA32_PMC4	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C6H, 198	IA32_PMC5	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C7H, 199	IA32_PMC6	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C8H, 200	IA32_PMC7	
Performance Counter Register (if core not shared by threads)		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
28	Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.	
63:29	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Core
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-State Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Thread
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Thread
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFVTSEL0	
See Table 2-2.		Thread
Register Address: 187H, 391	IA32_PERFVTSEL1	
See Table 2-2.		Thread
Register Address: 188H, 392	IA32_PERFVTSEL2	
See Table 2-2.		Thread
Register Address: 189H, 393	IA32_PERFVTSEL3	
See Table 2-2.		Thread
Register Address: 18AH, 394	IA32_PERFVTSEL4	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 4.		Core
Register Address: 18BH, 395	IA32_PERFVTSEL5	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 5.		Core
Register Address: 18CH, 396	IA32_PERFVTSEL6	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 6.		Core
Register Address: 18DH, 397	IA32_PERFVTSEL7	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 7.		Core
Register Address: 198H, 408	IA32_PERF_STATUS	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Package
15:0	Current Performance State Value	
63:16	Reserved.	
Register Address: 198H, 408	MSR_PERF_STATUS	
Performance Status		Package
47:32	Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2 ¹³).	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Thread
3:0	On demand Clock Modulation Duty Cycle (R/W) In 6.25% increment.	
4	On demand Clock Modulation Enable (R/W)	
63:5	Reserved.	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
15:12	Reserved.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Thread
6:1	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Thread
10:8	Reserved	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Thread
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Thread
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Thread
21:19	Reserved.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Limit CPUID Maxval (R/W) See Table 2-2.	Thread
23	xTPR Message Disable (R/W) See Table 2-2.	Thread
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Thread
37:35	Reserved.	
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Unique
15:0	Reserved.	
23:16	Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
63:24	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	Core
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
3	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	Core
63:4	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .		
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
See Table 2-2.		Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
0	LBR: Last Branch Record	
1	BTF	
5:2	Reserved.	
6	TR: Branch Trace	
7	BTS: Log Branch Trace Message to BTS buffer	
8	BTINT	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
9	BTS_OFF_OS	
10	BTS_OFF_USER	
11	FREEZE_LBR_ON_PMI	
12	FREEZE_PERFMON_ON_PMI	
13	ENABLE_UNCORE_PMI	
14	FREEZE_WHILE_SMM	
63:15	Reserved.	
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
See http://biosbits.org .		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Thread
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Thread
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Thread
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Thread
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Thread
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Thread
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Thread
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Thread
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Thread
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Thread
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Thread
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Thread
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Thread
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Thread
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Thread
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Thread
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Thread
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
See Table 2-2.		Thread
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
See Table 2-2.		Thread
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Thread
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Thread
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Thread
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Thread
Register Address: 269H, 617	IA32_MTRR_FIX4K_80000	
See Table 2-2.		Thread
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Thread
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Thread
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Thread
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Thread
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Thread
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Thread
Register Address: 280H, 640	IA32_MCO_CTL2	
See Table 2-2.		Core
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Core
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
Always 0 (CMCI not supported).		Package
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Thread
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2 and Section 20.4.1, "IA32_DEBUGCTL MSR."		Thread
5:0	LBR Format See Table 2-2.	
6	PEBS Record Format.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7	PEBSSaveArchRegs See Table 2-2.	
11:8	PEBS_REC_FORMAT See Table 2-2.	
12	SMM_FREEZE See Table 2-2.	
63:13	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		
0	Ovf_PMC0	Thread
1	Ovf_PMC1	Thread
2	Ovf_PMC2	Thread
3	Ovf_PMC3	Thread
4	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	Core
5	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)	Core
6	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)	Core
7	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)	Core
31:8	Reserved.	
32	Ovf_FixedCtr0	Thread
33	Ovf_FixedCtr1	Thread
34	Ovf_FixedCtr2	Thread
60:35	Reserved.	
61	Ovf_Uncore	Thread
62	Ovf_BufDSSAVE	Thread
63	CondChgd	Thread
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		Thread
0	Set 1 to enable PMC0 to count.	Thread
1	Set 1 to enable PMC1 to count.	Thread
2	Set 1 to enable PMC2 to count.	Thread
3	Set 1 to enable PMC3 to count.	Thread
4	Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4).	Core
5	Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5).	Core
6	Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6).	Core
7	Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7).	Core

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
31:8	Reserved.	
32	Set 1 to enable FixedCtr0 to count.	Thread
33	Set 1 to enable FixedCtr1 to count.	Thread
34	Set 1 to enable FixedCtr2 to count.	Thread
63:35	Reserved.	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		
0	Set 1 to clear Ovf_PMC0.	Thread
1	Set 1 to clear Ovf_PMC1.	Thread
2	Set 1 to clear Ovf_PMC2.	Thread
3	Set 1 to clear Ovf_PMC3.	Thread
4	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).	Core
5	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).	Core
6	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).	Core
7	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).	Core
31:8	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	Thread
33	Set 1 to clear Ovf_FixedCtr1.	Thread
34	Set 1 to clear Ovf_FixedCtr2.	Thread
60:35	Reserved.	
61	Set 1 to clear Ovf_Uncore.	Thread
62	Set 1 to clear Ovf_BufDSSAVE.	Thread
63	Set 1 to clear CondChgd.	Thread
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 22.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
0	Enable PEBS on IA32_PMC0. (R/W)	
1	Enable PEBS on IA32_PMC1. (R/W)	
2	Enable PEBS on IA32_PMC2. (R/W)	
3	Enable PEBS on IA32_PMC3. (R/W)	
31:4	Reserved.	
32	Enable Load Latency on IA32_PMC0. (R/W)	
33	Enable Load Latency on IA32_PMC1. (R/W)	
34	Enable Load Latency on IA32_PMC2. (R/W)	
35	Enable Load Latency on IA32_PMC3. (R/W)	
62:36	Reserved.	
63	Enable Precise Store (R/W)	
Register Address: 3F6H, 1014	MSR_PEBS_LD_LAT	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 22.3.1.1.2, "Load Latency Performance Monitoring Facility."		Thread
15:0	Minimum threshold latency value of tagged load operation that will be counted. (R/W)	
63:36	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FEH, 1022	MSR_CORE_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C7 Residency Counter (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Core

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 403H, 1027	IA32_MCO_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
0	PCU Hardware Error (R/W) When set, enables signaling of PCU hardware detected errors.	
1	PCU Controller Error (R/W) When set, enables signaling of PCU controller detected errors.	
2	PCU Firmware Error (R/W) When set, enables signaling of PCU firmware detected errors.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:2	Reserved.	
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		Thread
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2 and Appendix A.4, "VM-Exit Controls."		Thread
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2 and Appendix A.5, "VM-Entry Controls."		Thread
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2 and Appendix A.6, "Miscellaneous Data."		Thread
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2 and Appendix A.9, "VMCS Enumeration."		Thread
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2		Thread
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2		Thread
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2		Thread
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTL	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2		Thread
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTL	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2		Thread
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Thread
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Thread
Register Address: 4C3H, 1219	IA32_A_PMC2	
See Table 2-2.		Thread
Register Address: 4C4H, 1220	IA32_A_PMC3	
See Table 2-2.		Thread
Register Address: 4C5H, 1221	IA32_A_PMC4	
See Table 2-2.		Core
Register Address: 4C6H, 1222	IA32_A_PMC5	
See Table 2-2.		Core
Register Address: 4C7H, 1223	IA32_A_PMC6	
See Table 2-2.		Core
Register Address: 4C8H, 1224	IA32_A_PMC7	
See Table 2-2.		Core
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		Thread
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 17.10.1, "RAPL Interfaces."		Package
Register Address: 60AH, 1546	MSR_PKG_C3_IRT	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60BH, 1547	MSR_PKG_C6_IRTL	
Package C6 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C6 to a C0 state, where an interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 638H, 1592	MSR_PP0_POWER_LIMIT	
PP0 RAPL Power Limit Control (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.9.1 and record format in Section 20.4.8.1.		Thread
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.		Thread
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
See Table 2-2.		Thread
Register Address: 802H–83FH, 2050–2111	X2APIC MSRs	
See Table 2-2.		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 20.17.2, “IA32_TSC_AUX Register and RDTSCP Support.”		Thread

2.11.1 MSRs in the 2nd Generation Intel® Core™ Processor Family Based on Sandy Bridge Microarchitecture

Table 2-21 and Table 2-22 list model-specific registers (MSRs) that are specific to the 2nd generation Intel® Core™ processor family based on the Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH; see Table 2-1.

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 60CH, 1548	MSR_PKG_C7_IRTL	
Package C7 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C7 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 63AH, 1594	MSR_PPO_POLICY	

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
PP0 Balance Policy (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 640H, 1600	MSR_PP1_POWER_LIMIT	
PP1 RAPL Power Limit Control (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 642H, 1602	MSR_PP1_POLICY	
PP1 Balance Policy (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-21, and Table 2-22 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH.		

Table 2-22 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH.

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4 select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 392H, 914	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Report the number of C-Box units with performance counters, including processor cores and processor graphics.	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 702H, 1794	MSR_UNC_CBO_0_PERFEVTSEL2	
Uncore C-Box 0, Counter 2 Event Select MSR		Package
Register Address: 703H, 1795	MSR_UNC_CBO_0_PERFEVTSEL3	
Uncore C-Box 0, Counter 3 Event Select MSR		Package
Register Address: 705H, 1797	MSR_UNC_CBO_0_UNIT_STATUS	
Uncore C-Box 0, Unit Status for Counter 0-3		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 708H, 1800	MSR_UNC_CBO_0_PERFCTR2	
Uncore C-Box 0, Performance Counter 2		Package
Register Address: 709H, 1801	MSR_UNC_CBO_0_PERFCTR3	
Uncore C-Box 0, Performance Counter 3		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 712H, 1810	MSR_UNC_CBO_1_PERFEVTSEL2	
Uncore C-Box 1, Counter 2 Event Select MSR		Package
Register Address: 713H, 1811	MSR_UNC_CBO_1_PERFEVTSEL3	
Uncore C-Box 1, Counter 3 Event Select MSR		Package
Register Address: 715H, 1813	MSR_UNC_CBO_1_UNIT_STATUS	
Uncore C-Box 1, Unit Status for Counter 0-3		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 718H, 1816	MSR_UNC_CBO_1_PERFCTR2	
Uncore C-Box 1, Performance Counter 2		Package
Register Address: 719H, 1817	MSR_UNC_CBO_1_PERFCTR3	
Uncore C-Box 1, Performance Counter 3		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 722H, 1826	MSR_UNC_CBO_2_PERFEVTSEL2	
Uncore C-Box 2, Counter 2 Event Select MSR		Package
Register Address: 723H, 1827	MSR_UNC_CBO_2_PERFEVTSEL3	
Uncore C-Box 2, Counter 3 Event Select MSR		Package
Register Address: 725H, 1829	MSR_UNC_CBO_2_UNIT_STATUS	
Uncore C-Box 2, Unit Status for Counter 0-3		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 728H, 1832	MSR_UNC_CBO_3_PERFCTR2	

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 3, Performance Counter 2		Package
Register Address: 729H, 1833	MSR_UNC_CBO_3_PERFCTR3	
Uncore C-Box 3, Performance Counter 3		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 732H, 1842	MSR_UNC_CBO_3_PERFEVTSEL2	
Uncore C-Box 3, Counter 2 Event Select MSR		Package
Register Address: 733H, 1843	MSR_UNC_CBO_3_PERFEVTSEL3	
Uncore C-Box 3, counter 3 Event Select MSR		Package
Register Address: 735H, 1845	MSR_UNC_CBO_3_UNIT_STATUS	
Uncore C-Box 3, Unit Status for Counter 0-3		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 738H, 1848	MSR_UNC_CBO_3_PERFCTR2	
Uncore C-Box 3, Performance Counter 2		Package
Register Address: 739H, 1849	MSR_UNC_CBO_3_PERFCTR3	
Uncore C-Box 3, Performance Counter 3		Package
Register Address: 740H, 1856	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 741H, 1857	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 742H, 1858	MSR_UNC_CBO_4_PERFEVTSEL2	
Uncore C-Box 4, Counter 2 Event Select MSR		Package
Register Address: 743H, 1859	MSR_UNC_CBO_4_PERFEVTSEL3	
Uncore C-Box 4, Counter 3 Event Select MSR		Package
Register Address: 745H, 1861	MSR_UNC_CBO_4_UNIT_STATUS	
Uncore C-Box 4, Unit status for Counter 0-3		Package
Register Address: 746H, 1862	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 747H, 1863	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 748H, 1864	MSR_UNC_CBO_4_PERFCTR2	
Uncore C-Box 4, Performance Counter 2		Package

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 749H, 1865	MSR_UNC_CBO_4_PERFCTR3	
Uncore C-Box 4, Performance Counter 3		Package

2.11.2 MSRs in the Intel® Xeon® Processor E5 Family Based on Sandy Bridge Microarchitecture

Table 2-23 lists additional model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family based on Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH, and also support MSRs listed in Table 2-20 and Table 2-24.

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
	MC Bank Error Configuration (R/W)	Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 cores active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 cores active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 cores active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 cores active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 cores active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 cores active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 cores active.	Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 39CH, 924	MSR_PEBS_NUM_ALT	
ENABLE_PEBS_NUM_ALT (R/W)		Package
0	ENABLE_PEBS_NUM_ALT (R/W) Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see https://perfmon-events.intel.com/ .	
63:1	Reserved, must be zero.	
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
Package RAPL Perf Status (R/O)		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-23, and Table 2-24 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH.		

2.11.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 Family

Intel Xeon Processor E5 family is based on the Sandy Bridge microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-24. For complete details of the uncore PMU, refer to the Intel Xeon Processor E5 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH.

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C08H, 3080	MSR_U_PMON_UCLK_FIXED_CTL	
Uncore U-box UCLK Fixed Counter Control		Package
Register Address: C09H, 3081	MSR_U_PMON_UCLK_FIXED_CTR	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore U-box UCLK Fixed Counter		Package
Register Address: C10H, 3088	MSR_U_PMON_EVTNSELO	
Uncore U-box PerfMon Event Select for U-box Counter 0		Package
Register Address: C11H, 3089	MSR_U_PMON_EVTNSEL1	
Uncore U-box PerfMon Event Select for U-box Counter 1		Package
Register Address: C16H, 3094	MSR_U_PMON_CTR0	
Uncore U-box PerfMon Counter 0		Package
Register Address: C17H, 3095	MSR_U_PMON_CTR1	
Uncore U-box PerfMon Counter 1		Package
Register Address: C24H, 3108	MSR_PCU_PMON_BOX_CTL	
Uncore PCU PerfMon for PCU-box-wide Control		Package
Register Address: C30H, 3120	MSR_PCU_PMON_EVTNSELO	
Uncore PCU PerfMon Event Select for PCU Counter 0		Package
Register Address: C31H, 3121	MSR_PCU_PMON_EVTNSEL1	
Uncore PCU PerfMon Event Select for PCU Counter 1		Package
Register Address: C32H, 3122	MSR_PCU_PMON_EVTNSEL2	
Uncore PCU PerfMon Event Select for PCU Counter 2		Package
Register Address: C33H, 3123	MSR_PCU_PMON_EVTNSEL3	
Uncore PCU PerfMon Event Select for PCU Counter 3		Package
Register Address: C34H, 3124	MSR_PCU_PMON_BOX_FILTER	
Uncore PCU PerfMon box-wide Filter		Package
Register Address: C36H, 3126	MSR_PCU_PMON_CTR0	
Uncore PCU PerfMon Counter 0		Package
Register Address: C37H, 3127	MSR_PCU_PMON_CTR1	
Uncore PCU PerfMon Counter 1		Package
Register Address: C38H, 3128	MSR_PCU_PMON_CTR2	
Uncore PCU PerfMon Counter 2		Package
Register Address: C39H, 3129	MSR_PCU_PMON_CTR3	
Uncore PCU PerfMon Counter 3		Package
Register Address: D04H, 3332	MSR_CO_PMON_BOX_CTL	
Uncore C-box 0 PerfMon Local Box Wide Control		Package
Register Address: D10H, 3344	MSR_CO_PMON_EVTNSELO	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 0		Package
Register Address: D11H, 3345	MSR_CO_PMON_EVTNSEL1	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 1		Package
Register Address: D12H, 3346	MSR_CO_PMON_EVTNSEL2	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 2		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D13H, 3347	MSR_CO_PMON_EVNTSEL3	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 3		Package
Register Address: D14H, 3348	MSR_CO_PMON_BOX_FILTER	
Uncore C-box 0 PerfMon Box Wide Filter		Package
Register Address: D16H, 3350	MSR_CO_PMON_CTR0	
Uncore C-box 0 PerfMon Counter 0		Package
Register Address: D17H, 3351	MSR_CO_PMON_CTR1	
Uncore C-box 0 PerfMon Counter 1		Package
Register Address: D18H, 3352	MSR_CO_PMON_CTR2	
Uncore C-box 0 PerfMon Counter 2		Package
Register Address: D19H, 3353	MSR_CO_PMON_CTR3	
Uncore C-box 0 PerfMon Counter 3		Package
Register Address: D24H, 3364	MSR_C1_PMON_BOX_CTL	
Uncore C-box 1 PerfMon Local Box Wide Control		Package
Register Address: D30H, 3376	MSR_C1_PMON_EVNTSELO	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 0		Package
Register Address: D31H, 3377	MSR_C1_PMON_EVNTSEL1	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 1		Package
Register Address: D32H, 3378	MSR_C1_PMON_EVNTSEL2	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 2		Package
Register Address: D33H, 3379	MSR_C1_PMON_EVNTSEL3	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 3		Package
Register Address: D34H, 3380	MSR_C1_PMON_BOX_FILTER	
Uncore C-box 1 PerfMon Box Wide Filter		Package
Register Address: D36H, 3382	MSR_C1_PMON_CTR0	
Uncore C-box 1 PerfMon Counter 0		Package
Register Address: D37H, 3383	MSR_C1_PMON_CTR1	
Uncore C-box 1 PerfMon Counter 1		Package
Register Address: D38H, 3384	MSR_C1_PMON_CTR2	
Uncore C-box 1 PerfMon Counter 2		Package
Register Address: D39H, 3385	MSR_C1_PMON_CTR3	
Uncore C-box 1 PerfMon Counter 3		Package
Register Address: D44H, 3396	MSR_C2_PMON_BOX_CTL	
Uncore C-box 2 PerfMon Local Box Wide Control		Package
Register Address: D50H, 3408	MSR_C2_PMON_EVNTSELO	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 0		Package
Register Address: D51H, 3409	MSR_C2_PMON_EVNTSEL1	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 1		Package
Register Address: D52H, 3410	MSR_C2_PMON_EVNTSEL2	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 2		Package
Register Address: D53H, 3411	MSR_C2_PMON_EVNTSEL3	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 3		Package
Register Address: D54H, 3412	MSR_C2_PMON_BOX_FILTER	
Uncore C-box 2 PerfMon Box Wide Filter		Package
Register Address: D56H, 3414	MSR_C2_PMON_CTR0	
Uncore C-box 2 PerfMon Counter 0		Package
Register Address: D57H, 3415	MSR_C2_PMON_CTR1	
Uncore C-box 2 PerfMon Counter 1		Package
Register Address: D58H, 3416	MSR_C2_PMON_CTR2	
Uncore C-box 2 PerfMon Counter 2		Package
Register Address: D59H, 3417	MSR_C2_PMON_CTR3	
Uncore C-box 2 PerfMon Counter 3		Package
Register Address: D64H, 3428	MSR_C3_PMON_BOX_CTL	
Uncore C-box 3 PerfMon Local Box Wide Control		Package
Register Address: D70H, 3440	MSR_C3_PMON_EVNTSEL0	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 0		Package
Register Address: D71H, 3441	MSR_C3_PMON_EVNTSEL1	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 1		Package
Register Address: D72H, 3442	MSR_C3_PMON_EVNTSEL2	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 2		Package
Register Address: D73H, 3443	MSR_C3_PMON_EVNTSEL3	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 3		Package
Register Address: D74H, 3444	MSR_C3_PMON_BOX_FILTER	
Uncore C-box 3 PerfMon Box Wide Filter		Package
Register Address: D76H, 3446	MSR_C3_PMON_CTR0	
Uncore C-box 3 PerfMon Counter 0		Package
Register Address: D77H, 3447	MSR_C3_PMON_CTR1	
Uncore C-box 3 PerfMon Counter 1		Package
Register Address: D78H, 3448	MSR_C3_PMON_CTR2	
Uncore C-box 3 PerfMon Counter 2		Package
Register Address: D79H, 3449	MSR_C3_PMON_CTR3	
Uncore C-box 3 PerfMon Counter 3		Package
Register Address: D84H, 3460	MSR_C4_PMON_BOX_CTL	
Uncore C-box 4 PerfMon Local Box Wide Control		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D90H, 3472	MSR_C4_PMON_EVNTSEL0	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 0		Package
Register Address: D91H, 3473	MSR_C4_PMON_EVNTSEL1	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 1		Package
Register Address: D92H, 3474	MSR_C4_PMON_EVNTSEL2	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 2		Package
Register Address: D93H, 3475	MSR_C4_PMON_EVNTSEL3	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 3		Package
Register Address: D94H, 3476	MSR_C4_PMON_BOX_FILTER	
Uncore C-box 4 PerfMon Box Wide Filter		Package
Register Address: D96H, 3478	MSR_C4_PMON_CTR0	
Uncore C-box 4 PerfMon Counter 0		Package
Register Address: D97H, 3479	MSR_C4_PMON_CTR1	
Uncore C-box 4 PerfMon Counter 1		Package
Register Address: D98H, 3480	MSR_C4_PMON_CTR2	
Uncore C-box 4 PerfMon Counter 2		Package
Register Address: D99H, 3481	MSR_C4_PMON_CTR3	
Uncore C-box 4 PerfMon Counter 3		Package
Register Address: DA4H, 3492	MSR_C5_PMON_BOX_CTL	
Uncore C-box 5 PerfMon Local Box Wide Control		Package
Register Address: DB0H, 3504	MSR_C5_PMON_EVNTSEL0	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 0		Package
Register Address: DB1H, 3505	MSR_C5_PMON_EVNTSEL1	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 1		Package
Register Address: DB2H, 3506	MSR_C5_PMON_EVNTSEL2	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 2		Package
Register Address: DB3H, 3507	MSR_C5_PMON_EVNTSEL3	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 3		Package
Register Address: DB4H, 3508	MSR_C5_PMON_BOX_FILTER	
Uncore C-box 5 PerfMon Box Wide Filter		Package
Register Address: DB6H, 3510	MSR_C5_PMON_CTR0	
Uncore C-box 5 PerfMon Counter 0		Package
Register Address: DB7H, 3511	MSR_C5_PMON_CTR1	
Uncore C-box 5 PerfMon Counter 1		Package
Register Address: DB8H, 3512	MSR_C5_PMON_CTR2	
Uncore C-box 5 PerfMon Counter 2		Package
Register Address: DB9H, 3513	MSR_C5_PMON_CTR3	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 5 PerfMon Counter 3		Package
Register Address: DC4H, 3524	MSR_C6_PMON_BOX_CTL	
Uncore C-box 6 PerfMon Local Box Wide Control		Package
Register Address: DD0H, 3536	MSR_C6_PMON_EVTNTSEL0	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 0		Package
Register Address: DD1H, 3537	MSR_C6_PMON_EVTNTSEL1	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 1		Package
Register Address: DD2H, 3538	MSR_C6_PMON_EVTNTSEL2	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 2		Package
Register Address: DD3H, 3539	MSR_C6_PMON_EVTNTSEL3	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 3		Package
Register Address: DD4H, 3540	MSR_C6_PMON_BOX_FILTER	
Uncore C-box 6 PerfMon Box Wide Filter		Package
Register Address: DD6H, 3542	MSR_C6_PMON_CTR0	
Uncore C-box 6 PerfMon Counter 0		Package
Register Address: DD7H, 3543	MSR_C6_PMON_CTR1	
Uncore C-box 6 PerfMon Counter 1		Package
Register Address: DD8H, 3544	MSR_C6_PMON_CTR2	
Uncore C-box 6 PerfMon Counter 2		Package
Register Address: DD9H, 3545	MSR_C6_PMON_CTR3	
Uncore C-box 6 PerfMon Counter 3		Package
Register Address: DE4H, 3556	MSR_C7_PMON_BOX_CTL	
Uncore C-box 7 PerfMon Local Box Wide Control		Package
Register Address: DF0H, 3568	MSR_C7_PMON_EVTNTSEL0	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 0		Package
Register Address: DF1H, 3569	MSR_C7_PMON_EVTNTSEL1	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 1		Package
Register Address: DF2H, 3570	MSR_C7_PMON_EVTNTSEL2	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 2		Package
Register Address: DF3H, 3571	MSR_C7_PMON_EVTNTSEL3	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 3		Package
Register Address: DF4H, 3572	MSR_C7_PMON_BOX_FILTER	
Uncore C-box 7 PerfMon Box Wide Filter		Package
Register Address: DF6H, 3574	MSR_C7_PMON_CTR0	
Uncore C-box 7 PerfMon Counter 0		Package
Register Address: DF7H, 3575	MSR_C7_PMON_CTR1	
Uncore C-box 7 PerfMon Counter 1		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DF8H, 3576	MSR_C7_PMON_CTR2	
Uncore C-box 7 PerfMon Counter 2		Package
Register Address: DF9H, 3577	MSR_C7_PMON_CTR3	
Uncore C-box 7 PerfMon Counter 3		Package

2.12 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY BASED ON IVY BRIDGE MICROARCHITECTURE

The 3rd generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v2 product family based on Ivy Bridge microarchitecture support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-25. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates that TDP Limit for Turbo mode is not programmable.	Package
31:30	Reserved.	
32	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.	Package
34:33	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved	Package
39:35	Reserved.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
55:48	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.	Package
63:56	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .	Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
24:16	Reserved	
25	C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
28	Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.	
63:29	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O)		Package
7:0	Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).	
63:8	Reserved.	
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
	ConfigTDP Level 1 ratio and power level (R/O)	Package
14:0	PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.	
15	Reserved.	
23:16	Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.	
47	Reserved.	
62:48	PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.	
63	Reserved.	
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 ratio and power level (R/O)		Package
14:0	PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.	
15	Reserved.	
23:16	Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.	
47	Reserved.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
62:48	PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.	
63	Reserved.	
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W)		Package
1:0	TDP_LEVEL (RW/L) System BIOS can program this field.	
30:2	Reserved.	
31	Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
See Table 2-20, Table 2-21, and Table 2-22 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.		

2.12.1 MSRs in the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture

Table 2-26 lists model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 v2 Product Family (based on Ivy Bridge-E microarchitecture). These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH; see Table 2-1. These processors supports the MSR interfaces listed in Table 2-20 and Table 2-26.

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/W0) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) When set to 1, indicates that Protected Processor Inventory Number (PPIN) capability can be enabled for a privileged system inventory agent to read PPIN from MSR_PPIN. When set to 0, PPIN capability is not supported. An attempt to access MSR_PPIN_CTL or MSR_PPIN will cause #GP.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
30	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	Reserved.	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
MC Bank Error Configuration (R/W)		Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
27:24	TCC Activation Offset (R/W) Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only if MSR_PLATFORM_INFO.[30] is set.	
63:28	Reserved.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package
63:32	Reserved.	
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 296H, 662	IA32_MC22_CTL2	
See Table 2-2.		Package
Register Address: 297H, 663	IA32_MC23_CTL2IA32_MC23_CTL2	
See Table 2-2.		Package
Register Address: 298H, 664	IA32_MC24_CTL2	
See Table 2-2.		Package
Register Address: 299H, 665	IA32_MC25_CTL2	
See Table 2-2.		Package
Register Address: 29AH, 666	IA32_MC26_CTL2	
See Table 2-2.		Package
Register Address: 29BH, 667	IA32_MC27_CTL2	
See Table 2-2.		Package
Register Address: 29CH, 668	IA32_MC28_CTL2	
See Table 2-2.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 454H, 1108	IA32_MC21_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 458H, 1112	IA32_MC22_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 459H, 1113	IA32_MC22_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45AH, 1114	IA32_MC22_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45BH, 1115	IA32_MC22_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45CH, 1116	IA32_MC23_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45DH, 1117	IA32_MC23_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45EH, 1118	IA32_MC23_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45FH, 1119	IA32_MC23_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 460H, 1120	IA32_MC24_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 461H, 1121	IA32_MC24_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 462H, 1122	IA32_MC24_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 463H, 1123	IA32_MC24_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 464H, 1124	IA32_MC25_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 465H, 1125	IA32_MC25_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 466H, 1126	IA32_MC25_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 467H, 1127	IA32_MC2MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 468H, 1128	IA32_MC26_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 469H, 1129	IA32_MC26_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46AH, 1130	IA32_MC26_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46BH, 1131	IA32_MC26_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46CH, 1132	IA32_MC27_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46DH, 1133	IA32_MC27_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46EH, 1134	IA32_MC27_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46FH, 1135	IA32_MC27_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 470H, 1136	IA32_MC28_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 471H, 1137	IA32_MC28_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 472H, 1138	IA32_MC28_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 473H, 1139	IA32_MC28_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
Package RAPL Perf Status (R/O)		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
See Table 2-20, for other MSR definitions applicable to Intel Xeon processor E5 v2 with a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH.		

2.12.2 Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family

The Intel® Xeon® processor E7 v2 family (based on Ivy Bridge-E microarchitecture) with a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH supports the MSR interfaces listed in Table 2-20, Table 2-26, and Table 2-27.

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
63:16	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
63:25	Reserved.	
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status (R/W)		Thread
0	RIPV	
1	EIPV	
2	MCIP	
3	LMCE Signaled	
63:4	Reserved.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
39:32	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.	Package
47:40	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.	Package
55:48	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.	Package
62:56	Reserved.	
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT and MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 29DH, 669	IA32_MC29_CTL2	
See Table 2-2.		Package
Register Address: 29EH, 670	IA32_MC30_CTL2	
See Table 2-2.		Package
Register Address: 29FH, 671	IA32_MC31_CTL2	
See Table 2-2.		Package
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 22.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
$n:0$	Enable PEBS on IA32_PMCx. (R/W)	
$31:n+1$	Reserved.	
$32+m:32$	Enable Load Latency on IA32_PMCx. (R/W)	
$63:33+m$	Reserved.	
Register Address: 41BH, 1051	IA32_MC6_MISC	
Misc MAC Information of Integrated I/O (R/O) See Section 18.3.2.4.		Package
5:0	Recoverable Address LSB	
8:6	Address Mode	
15:9	Reserved.	
31:16	PCI Express Requestor ID	
39:32	PCI Express Segment Number	
63:32	Reserved.	
Register Address: 474H, 1140	IA32_MC29_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 475H, 1141	IA32_MC29_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 476H, 1142	IA32_MC29_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 477H, 1143	IA32_MC29_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 478H, 1144	IA32_MC30_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 479H, 1145	IA32_MC30_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47AH, 1146	IA32_MC30_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47BH, 1147	IA32_MC30_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47CH, 1148	IA32_MC31_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47DH, 1149	IA32_MC31_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47EH, 1150	IA32_MC31_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47FH, 1147	IA32_MC31_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
See Table 2-20, Table 2-26 for other MSR definitions applicable to Intel Xeon processor E7 v2 with a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.12.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Intel Xeon Processor E5 v2 and E7 v2 families are based on the Ivy Bridge-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-24 and Table 2-28. For complete detail of the uncore PMU, refer to Intel

Xeon Processor E5 v2 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH.

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C00H, 3072	MSR_PMON_GLOBAL_CTL	
Uncore PerfMon Per-Socket Global Control		Package
Register Address: C01H, 3073	MSR_PMON_GLOBAL_STATUS	
Uncore PerfMon Per-Socket Global Status		Package
Register Address: C06H, 3078	MSR_PMON_GLOBAL_CONFIG	
Uncore PerfMon Per-Socket Global Configuration		Package
Register Address: C15H, 3093	MSR_U_PMON_BOX_STATUS	
Uncore U-box PerfMon U-Box Wide Status		Package
Register Address: C35H, 3125	MSR_PCU_PMON_BOX_STATUS	
Uncore PCU PerfMon Box Wide Status		Package
Register Address: D1AH, 3354	MSR_C0_PMON_BOX_FILTER1	
Uncore C-Box 0 PerfMon Box Wide Filter1		Package
Register Address: D3AH, 3386	MSR_C1_PMON_BOX_FILTER1	
Uncore C-Box 1 PerfMon Box Wide Filter1		Package
Register Address: D5AH, 3418	MSR_C2_PMON_BOX_FILTER1	
Uncore C-Box 2 PerfMon Box Wide Filter1		Package
Register Address: D7AH, 3450	MSR_C3_PMON_BOX_FILTER1	
Uncore C-Box 3 PerfMon Box Wide Filter1		Package
Register Address: D9AH, 3482	MSR_C4_PMON_BOX_FILTER1	
Uncore C-Box 4 PerfMon Box Wide Filter1		Package
Register Address: DBAH, 3514	MSR_C5_PMON_BOX_FILTER1	
Uncore C-Box 5 PerfMon Box Wide Filter1		Package
Register Address: DDAH, 3546	MSR_C6_PMON_BOX_FILTER1	
Uncore C-Box 6 PerfMon Box Wide Filter1		Package
Register Address: DFAH, 3578	MSR_C7_PMON_BOX_FILTER1	
Uncore C-Box 7 PerfMon Box Wide Filter1		Package
Register Address: E04H, 3588	MSR_C8_PMON_BOX_CTL	
Uncore C-Box 8 PerfMon Local Box Wide Control		Package
Register Address: E10H, 3600	MSR_C8_PMON_EVNTSELO	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 0		Package
Register Address: E11H, 3601	MSR_C8_PMON_EVNTSEL1	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 1		Package
Register Address: E12H, 3602	MSR_C8_PMON_EVNTSEL2	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 2		Package
Register Address: E13H, 3603	MSR_C8_PMON_EVNTSEL3	

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 3		Package
Register Address: E14H, 3604	MSR_C8_PMON_BOX_FILTER	
Uncore C-Box 8 PerfMon Box Wide Filter		Package
Register Address: E16H, 3606	MSR_C8_PMON_CTR0	
Uncore C-Box 8 PerfMon Counter 0		Package
Register Address: E17H, 3607	MSR_C8_PMON_CTR1	
Uncore C-Box 8 PerfMon Counter 1		Package
Register Address: E18H, 3608	MSR_C8_PMON_CTR2	
Uncore C-Box 8 PerfMon Counter 2		Package
Register Address: E19H, 3609	MSR_C8_PMON_CTR3	
Uncore C-Box 8 PerfMon Counter 3		Package
Register Address: E1AH, 3610	MSR_C8_PMON_BOX_FILTER1	
Uncore C-Box 8 PerfMon Box Wide Filter1		Package
Register Address: E24H, 3620	MSR_C9_PMON_BOX_CTL	
Uncore C-Box 9 PerfMon Local Box Wide Control		Package
Register Address: E30H, 3632	MSR_C9_PMON_EVNTSEL0	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 0		Package
Register Address: E31H, 3633	MSR_C9_PMON_EVNTSEL1	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 1		Package
Register Address: E32H, 3634	MSR_C9_PMON_EVNTSEL2	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 2		Package
Register Address: E33H, 3635	MSR_C9_PMON_EVNTSEL3	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 3		Package
Register Address: E34H, 3636	MSR_C9_PMON_BOX_FILTER	
Uncore C-Box 9 PerfMon Box Wide Filter		Package
Register Address: E36H, 3638	MSR_C9_PMON_CTR0	
Uncore C-Box 9 PerfMon Counter 0		Package
Register Address: E37H, 3639	MSR_C9_PMON_CTR1	
Uncore C-Box 9 PerfMon Counter 1		Package
Register Address: E38H, 3640	MSR_C9_PMON_CTR2	
Uncore C-Box 9 PerfMon Counter 2		Package
Register Address: E39H, 3641	MSR_C9_PMON_CTR3	
Uncore C-Box 9 PerfMon Counter 3		Package
Register Address: E3AH, 3642	MSR_C9_PMON_BOX_FILTER1	
Uncore C-Box 9 PerfMon Box Wide Filter1		Package
Register Address: E44H, 3652	MSR_C10_PMON_BOX_CTL	
Uncore C-Box 10 PerfMon Local Box Wide Control		Package

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E50H, 3664	MSR_C10_PMON_EVTNSELO	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 0		Package
Register Address: E51H, 3665	MSR_C10_PMON_EVTNSEL1	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 1		Package
Register Address: E52H, 3666	MSR_C10_PMON_EVTNSEL2	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 2		Package
Register Address: E53H, 3667	MSR_C10_PMON_EVTNSEL3	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 3		Package
Register Address: E54H, 3668	MSR_C10_PMON_BOX_FILTER	
Uncore C-Box 10 PerfMon Box Wide Filter		Package
Register Address: E56H, 3670	MSR_C10_PMON_CTR0	
Uncore C-Box 10 PerfMon Counter 0		Package
Register Address: E57H, 3671	MSR_C10_PMON_CTR1	
Uncore C-Box 10 PerfMon Counter 1		Package
Register Address: E58H, 3672	MSR_C10_PMON_CTR2	
Uncore C-Box 10 PerfMon Counter 2		Package
Register Address: E59H, 3673	MSR_C10_PMON_CTR3	
Uncore C-Box 10 PerfMon Counter 3		Package
Register Address: E5AH, 3674	MSR_C10_PMON_BOX_FILTER1	
Uncore C-Box 10 PerfMon Box Wide Filter1		Package
Register Address: E64H, 3684	MSR_C11_PMON_BOX_CTL	
Uncore C-Box 11 PerfMon Local Box Wide Control		Package
Register Address: E70H, 3696	MSR_C11_PMON_EVTNSELO	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 0		Package
Register Address: E71H, 3697	MSR_C11_PMON_EVTNSEL1	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 1		Package
Register Address: E72H, 3698	MSR_C11_PMON_EVTNSEL2	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 2		Package
Register Address: E73H, 3699	MSR_C11_PMON_EVTNSEL3	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 3		Package
Register Address: E74H, 3700	MSR_C11_PMON_BOX_FILTER	
Uncore C-Box 11 PerfMon Box Wide Filter		Package
Register Address: E76H, 3702	MSR_C11_PMON_CTR0	
Uncore C-Box 11 PerfMon Counter 0		Package
Register Address: E77H, 3703	MSR_C11_PMON_CTR1	
Uncore C-Box 11 PerfMon Counter 1		Package
Register Address: E78H, 3704	MSR_C11_PMON_CTR2	

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 11 PerfMon Counter 2		Package
Register Address: E79H, 3705	MSR_C11_PMON_CTR3	
Uncore C-Box 11 PerfMon Counter 3		Package
Register Address: E7AH, 3706	MSR_C11_PMON_BOX_FILTER1	
Uncore C-Box 11 PerfMon Box Wide Filter1		Package
Register Address: E84H, 3716	MSR_C12_PMON_BOX_CTL	
Uncore C-Box 12 PerfMon Local Box Wide Control		Package
Register Address: E90H, 3728	MSR_C12_PMON_EVNTSELO	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 0		Package
Register Address: E91H, 3729	MSR_C12_PMON_EVNTSEL1	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 1		Package
Register Address: E92H, 3730	MSR_C12_PMON_EVNTSEL2	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 2		Package
Register Address: E93H, 3731	MSR_C12_PMON_EVNTSEL3	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 3		Package
Register Address: E94H, 3732	MSR_C12_PMON_BOX_FILTER	
Uncore C-Box 12 PerfMon Box Wide Filter		Package
Register Address: E96H, 3734	MSR_C12_PMON_CTR0	
Uncore C-Box 12 PerfMon Counter 0		Package
Register Address: E97H, 3735	MSR_C12_PMON_CTR1	
Uncore C-Box 12 PerfMon Counter 1		Package
Register Address: E98H, 3736	MSR_C12_PMON_CTR2	
Uncore C-Box 12 PerfMon Counter 2		Package
Register Address: E99H, 3737	MSR_C12_PMON_CTR3	
Uncore C-Box 12 PerfMon Counter 3		Package
Register Address: E9AH, 3738	MSR_C12_PMON_BOX_FILTER1	
Uncore C-Box 12 PerfMon Box Wide Filter1		Package
Register Address: EA4H, 3748	MSR_C13_PMON_BOX_CTL	
Uncore C-Box 13 PerfMon Local Box Wide Control		Package
Register Address: EB0H, 3760	MSR_C13_PMON_EVNTSELO	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 0		Package
Register Address: EB1H, 3761	MSR_C13_PMON_EVNTSEL1	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 1		Package
Register Address: EB2H, 3762	MSR_C13_PMON_EVNTSEL2	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 2		Package
Register Address: EB3H, 3763	MSR_C13_PMON_EVNTSEL3	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 3		Package

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EB4H, 3764	MSR_C13_PMON_BOX_FILTER	
Uncore C-Box 13 PerfMon Box Wide Filter		Package
Register Address: EB6H, 3766	MSR_C13_PMON_CTR0	
Uncore C-Box 13 PerfMon Counter 0		Package
Register Address: EB7H, 3767	MSR_C13_PMON_CTR1	
Uncore C-Box 13 PerfMon Counter 1		Package
Register Address: EB8H, 3768	MSR_C13_PMON_CTR2	
Uncore C-Box 13 PerfMon Counter 2		Package
Register Address: EB9H, 3769	MSR_C13_PMON_CTR3	
Uncore C-Box 13 PerfMon Counter 3		Package
Register Address: EBAH, 3770	MSR_C13_PMON_BOX_FILTER1	
Uncore C-Box 13 PerfMon Box Wide Filter1		Package
Register Address: EC4H, 3780	MSR_C14_PMON_BOX_CTL	
Uncore C-Box 14 PerfMon Local Box Wide Control		Package
Register Address: ED0H, 3792	MSR_C14_PMON_EVTSELO	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 0		Package
Register Address: ED1H, 3793	MSR_C14_PMON_EVTSEL1	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 1		Package
Register Address: ED2H, 3794	MSR_C14_PMON_EVTSEL2	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 2		Package
Register Address: ED3H, 3795	MSR_C14_PMON_EVTSEL3	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 3		Package
Register Address: ED4H, 3796	MSR_C14_PMON_BOX_FILTER	
Uncore C-Box 14 PerfMon Box Wide Filter		Package
Register Address: ED6H, 3798	MSR_C14_PMON_CTR0	
Uncore C-Box 14 PerfMon Counter 0		Package
Register Address: ED7H, 3799	MSR_C14_PMON_CTR1	
Uncore C-Box 14 PerfMon Counter 1		Package
Register Address: ED8H, 3800	MSR_C14_PMON_CTR2	
Uncore C-Box 14 PerfMon Counter 2		Package
Register Address: ED9H, 3801	MSR_C14_PMON_CTR3	
Uncore C-Box 14 PerfMon Counter 3		Package
Register Address: EDAH, 3802	MSR_C14_PMON_BOX_FILTER1	
Uncore C-Box 14 PerfMon Box Wide Filter1		Package

2.13 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS BASED ON HASWELL MICROARCHITECTURE

The 4th generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v3 product family (based on Haswell microarchitecture), with a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH, 06_45H, or 06_46H, support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-29. For an MSR listed in Table 2-20 that also appears in Table 2-29, Table 2-29 supersedes Table 2-20.

The MSRs listed in Table 2-29 also apply to processors based on Haswell-E microarchitecture (see Section 2.14).

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
31:30	Reserved.	
32	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.	Package
34:33	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved.	Package
39:35	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
55:48	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.	Package
63:56	Reserved.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 186H, 390	IA32_PERFEVTSEL0	
Performance Event Select for Counter 0 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 22.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
Performance Event Select for Counter 1 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 22.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 188H, 392	IA32_PERFEVTSEL2	
Performance Event Select for Counter 2 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 22.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
33	IN_TXCP: See Section 22.3.6.5.1. When IN_TXCP=1 & IN_TX=1 and in sampling, a spurious PMI may occur and transactions may continuously abort near overflow conditions. Software should favor using IN_TXCP for counting over sampling. If sampling, software should use large "sample-after" value after clearing the counter configured to use IN_TXCP and also always reset the counter even when no overflow condition was reported.	
Register Address: 189H, 393	IA32_PERFEVTSEL3	
Performance Event Select for Counter 3 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 22.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W)		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
8	FAR_BRANCH	
9	EN_CALL_STACK	
63:9	Reserved.	
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
0	LBR: Last Branch Record	
1	BTF	
5:2	Reserved.	
6	TR: Branch Trace	
7	BTS: Log Branch Trace Message to BTS Buffer	
8	BTINT	
9	BTS_OFF_OS	
10	BTS_OFF_USER	
11	FREEZE_LBR_ON_PMI	
12	FREEZE_PERFMON_ON_PMI	
13	ENABLE_UNCORE_PMI	
14	FREEZE_WHILE_SMM	
15	RTM_DEBUG	
63:15	Reserved.	
Register Address: 491H, 1169	IA32_VMX_VMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Thread
Register Address: 60BH, 1548	MSR_PKG_C_IRT_L1	
Package C6/C7 Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the shorter-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 60CH, 1548	MSR_PKG_C6_C7_INTERRUPT_RESPONSE_LIMIT_2	
Package C6/C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the longer-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt response time limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O)		Package
7:0	Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).	
63:8	Reserved.	
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
ConfigTDP Level 1 Ratio and Power Level (R/O)		Package
14:0	PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.	
15	Reserved.	
23:16	Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.	
62:47	PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.	
63	Reserved.	
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 Ratio and Power Level (R/O)		Package
14:0	PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.	
15	Reserved.	
23:16	Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.	
62:47	PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.	
63	Reserved.	
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W)		Package
1:0	TDP_LEVEL (RW/L) System BIOS can program this field.	
30:2	Reserved.	
31	Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: C80H, 3200	IA32_DEBUG_INTERFACE	
Silicon Debug Feature Control (R/W) See Table 2-2.		Package

2.13.1 MSRs in the 4th Generation Intel® Core™ Processor Family Based on Haswell Microarchitecture

Table 2-30 lists model-specific registers (MSRs) that are specific to the 4th generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH, 06_45H, or 06_46H; see Table 2-1.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s Package C states C7 are not available to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH.	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved	
15	CFG Lock (R/WO)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
63:29	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.	
63:60	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Core 0 select.	
1	Core 1 select.	
2	Core 2 select.	
3	Core 3 select.	
18:4	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 392H, 914	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Encoded number of C-Box, derive value by "-1".	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.		Package
0	Lock (SMM-RW0) When set to '1' locks this register from further changes.	
1	Reserved.	
2	SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 4E2H, 1250	MSR_SMM_DELAYED	
SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.		Package

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 4E3H, 1251	MSR_SMM_BLOCKED	
SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.		Package
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is 0FFFH. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 640H, 1600	MSR_PP1_POWER_LIMIT	
PP1 RAPL Power Limit Control (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
PP1 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 642H, 1602	MSR_PP1_POLICY	
PP1 Balance Policy (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
12	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Graphics Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
15:12	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
5:2	Reserved.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
15:12	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:30	Reserved.	
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1824	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, and Table 2-29 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 063CH or 06_46H.		

2.13.2 Additional Residency MSRs Supported in 4th Generation Intel® Core™ Processors

The 4th generation Intel® Core™ processor family (based on Haswell microarchitecture) with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H supports the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-29, Table 2-30, and Table 2-31.

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_45H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/W0)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
63:29	Reserved.	
Register Address: 630H, 1584	MSR_PKG_C8_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C8 Residency Counter (R/O) Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 631H, 1585	MSR_PKG_C9_RESIDENCY	

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_45H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C9 Residency Counter (R/O) Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C10 Residency Counter (R/O) Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
See Table 2-20, Table 2-21, Table 2-22, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H.		

2.14 MSRS IN THE INTEL® XEON® PROCESSOR E5 V3 AND E7 V3 PRODUCT FAMILY

The Intel® Xeon® processor E5 v3 family and the Intel® Xeon® processor E7 v3 family are based on Haswell-E microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_3F). These processors support the MSR interfaces listed in Table 2-20, Table 2-29, and Table 2-32.

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 35H, 53	MSR_CORE_THREAD_COUNT	
Configured State of Enabled Processor Core Count and Logical Processor Count (R/O)		Package
- After a Power-On RESET, enumerates factory configuration of the number of processor cores and logical processors in the physical package. - Following the sequence of (i) BIOS modified a Configuration Mask which selects a subset of processor cores to be active post RESET and (ii) a RESET event after the modification, enumerates the current configuration of enabled processor core count and logical processor count in the physical package.		
15:0	THREAD_COUNT (R/O) The number of logical processors that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.	
31:16	Core_COUNT (R/O) The number of processor cores that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.	
63:32	Reserved.	
Register Address: 53H, 83	MSR_THREAD_ID_INFO	
A Hardware Assigned ID for the Logical Processor (R/O)		Thread

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7:0	Logical_Processor_ID (R/O) An implementation-specific numerical value physically assigned to each logical processor. This ID is not related to Initial APIC ID or x2APIC ID, it is unique within a physical package.	
63:8	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/WO)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
MC Bank Error Configuration (R/W)		Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.	Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package
39:32	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.	Package
47:40	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.	Package
55:48	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.	Package
63:56	Maximum Ratio Limit for 16C Maximum turbo ratio limit of 16 core active.	Package
Register Address: 1AFH, 431	MSR_TURBO_RATIO_LIMIT2	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 17C Maximum turbo ratio limit of 17 core active.	Package
15:8	Maximum Ratio Limit for 18C Maximum turbo ratio limit of 18 core active.	Package
62:16	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1, and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy Consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61EH, 1566	MSR_PCIE_PLL_RATIO	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Configuration of PCIE PLL Relative to BCLK(R/W)		Package
1:0	PCIE Ratio (R/W) 00b: Use 5:5 mapping for 100MHz operation (default). 01b: Use 5:4 mapping for 125MHz operation. 10b: Use 5:3 mapping for 166MHz operation. 11b: Use 5:2 mapping for 250MHz operation.	Package
2	LPLL Select (R/W) If 1, use configured setting of PCIE Ratio.	Package
3	LONG RESET (R/W) If 1, wait an additional time-out before re-locking Gen2/Gen3 PLLs.	Package
63:4	Reserved.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit	
3	Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit	
4	Reserved.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.	
12:11	Reserved.	
13	Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.	
14	Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.	
15	Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20	Reserved.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28:27	Reserved.	
29	Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
31	Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:32	Reserved.	
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x0: No monitoring. 0x1: L3 occupancy monitoring. All other encoding reserved.	
31:8	Reserved.	
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8EH, 3214	IA32_QM_CTR	
Monitoring Counter Register (R/O) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
61:0	Resource Monitored Data	
62	Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
63	Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
63: 10	Reserved.	
See Table 2-20 and Table 2-29 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3FH.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.14.1 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

The Intel Xeon Processor E5 v3 and E7 v3 families are based on Haswell-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-33. For complete details of the uncore PMU, refer to the Intel Xeon Processor E5 v3 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3FH.

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 700H, 1792	MSR_PMON_GLOBAL_CTL	
Uncore PerfMon Per-Socket Global Control		Package
Register Address: 701H, 1793	MSR_PMON_GLOBAL_STATUS	
Uncore PerfMon Per-Socket Global Status		Package
Register Address: 702H, 1794	MSR_PMON_GLOBAL_CONFIG	
Uncore PerfMon Per-Socket Global Configuration		Package
Register Address: 703H, 1795	MSR_U_PMON_UCLK_FIXED_CTL	
Uncore U-Box UCLK Fixed Counter Control		Package
Register Address: 704H, 1796	MSR_U_PMON_UCLK_FIXED_CTR	
Uncore U-Box UCLK Fixed Counter		Package
Register Address: 705H, 1797	MSR_U_PMON_EVNTSELO	
Uncore U-Box PerfMon Event Select for U-Box Counter 0		Package
Register Address: 706H, 1798	MSR_U_PMON_EVNTSEL1	
Uncore U-Box PerfMon Event Select for U-Box Counter 1		Package
Register Address: 708H, 1800	MSR_U_PMON_BOX_STATUS	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore U-Box PerfMon U-Box Wide Status		Package
Register Address: 709H, 1801	MSR_U_PMON_CTRL0	
Uncore U-Box PerfMon Counter 0		Package
Register Address: 70AH, 1802	MSR_U_PMON_CTRL1	
Uncore U-Box PerfMon Counter 1		Package
Register Address: 710H, 1808	MSR_PCU_PMON_BOX_CTL	
Uncore PCU PerfMon for PCU-Box-Wide Control		Package
Register Address: 711H, 1809	MSR_PCU_PMON_EVTSEL0	
Uncore PCU PerfMon Event Select for PCU Counter 0		Package
Register Address: 712H, 1810	MSR_PCU_PMON_EVTSEL1	
Uncore PCU PerfMon Event Select for PCU Counter 1		Package
Register Address: 713H, 1811	MSR_PCU_PMON_EVTSEL2	
Uncore PCU PerfMon Event Select for PCU Counter 2		Package
Register Address: 714H, 1812	MSR_PCU_PMON_EVTSEL3	
Uncore PCU PerfMon Event Select for PCU Counter 3		Package
Register Address: 715H, 1813	MSR_PCU_PMON_BOX_FILTER	
Uncore PCU PerfMon Box-Wide Filter		Package
Register Address: 716H, 1814	MSR_PCU_PMON_BOX_STATUS	
Uncore PCU PerfMon Box Wide Status		Package
Register Address: 717H, 1815	MSR_PCU_PMON_CTRL0	
Uncore PCU PerfMon Counter 0		Package
Register Address: 718H, 1816	MSR_PCU_PMON_CTRL1	
Uncore PCU PerfMon Counter 1		Package
Register Address: 719H, 1817	MSR_PCU_PMON_CTRL2	
Uncore PCU PerfMon Counter 2		Package
Register Address: 71AH, 1818	MSR_PCU_PMON_CTRL3	
Uncore PCU PerfMon Counter 3		Package
Register Address: 720H, 1824	MSR_S0_PMON_BOX_CTL	
Uncore SBo 0 PerfMon for SBo 0 Box-Wide Control		Package
Register Address: 721H, 1825	MSR_S0_PMON_EVTSEL0	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 0		Package
Register Address: 722H, 1826	MSR_S0_PMON_EVTSEL1	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 1		Package
Register Address: 723H, 1827	MSR_S0_PMON_EVTSEL2	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 2		Package
Register Address: 724H, 1828	MSR_S0_PMON_EVTSEL3	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 3		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 725H, 1829	MSR_S0_PMON_BOX_FILTER	
Uncore SBo 0 PerfMon Box-Wide Filter		Package
Register Address: 726H, 1830	MSR_S0_PMON_CTR0	
Uncore SBo 0 PerfMon Counter 0		Package
Register Address: 727H, 1831	MSR_S0_PMON_CTR1	
Uncore SBo 0 PerfMon Counter 1		Package
Register Address: 728H, 1832	MSR_S0_PMON_CTR2	
Uncore SBo 0 PerfMon Counter 2		Package
Register Address: 729H, 1833	MSR_S0_PMON_CTR3	
Uncore SBo 0 PerfMon Counter 3		Package
Register Address: 72AH, 1834	MSR_S1_PMON_BOX_CTL	
Uncore SBo 1 PerfMon for SBo 1 Box-Wide Control		Package
Register Address: 72BH, 1835	MSR_S1_PMON_EVTNSELO	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 0		Package
Register Address: 72CH, 1836	MSR_S1_PMON_EVTNSEL1	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 1		Package
Register Address: 72DH, 1837	MSR_S1_PMON_EVTNSEL2	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 2		Package
Register Address: 72EH, 1838	MSR_S1_PMON_EVTNSEL3	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 3		Package
Register Address: 72FH, 1839	MSR_S1_PMON_BOX_FILTER	
Uncore SBo 1 PerfMon Box-Wide Filter		Package
Register Address: 730H, 1840	MSR_S1_PMON_CTR0	
Uncore SBo 1 PerfMon Counter 0		Package
Register Address: 731H, 1841	MSR_S1_PMON_CTR1	
Uncore SBo 1 PerfMon Counter 1		Package
Register Address: 732H, 1842	MSR_S1_PMON_CTR2	
Uncore SBo 1 PerfMon Counter 2		Package
Register Address: 733H, 1843	MSR_S1_PMON_CTR3	
Uncore SBo 1 PerfMon Counter 3		Package
Register Address: 734H, 1844	MSR_S2_PMON_BOX_CTL	
Uncore SBo 2 PerfMon for SBo 2 Box-Wide Control		Package
Register Address: 735H, 1845	MSR_S2_PMON_EVTNSELO	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 0		Package
Register Address: 736H, 1846	MSR_S2_PMON_EVTNSEL1	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 1		Package
Register Address: 737H, 1847	MSR_S2_PMON_EVTNSEL2	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 2		Package
Register Address: 738H, 1848	MSR_S2_PMON_EVNTSEL3	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 3		Package
Register Address: 739H, 1849	MSR_S2_PMON_BOX_FILTER	
Uncore SBo 2 PerfMon Box-Wide Filter		Package
Register Address: 73AH, 1850	MSR_S2_PMON_CTR0	
Uncore SBo 2 PerfMon Counter 0		Package
Register Address: 73BH, 1851	MSR_S2_PMON_CTR1	
Uncore SBo 2 PerfMon Counter 1		Package
Register Address: 73CH, 1852	MSR_S2_PMON_CTR2	
Uncore SBo 2 PerfMon Counter 2		Package
Register Address: 73DH, 1853	MSR_S2_PMON_CTR3	
Uncore SBo 2 PerfMon Counter 3		Package
Register Address: 73EH, 1854	MSR_S3_PMON_BOX_CTL	
Uncore SBo 3 PerfMon for SBo 3 Box-Wide Control		Package
Register Address: 73FH, 1855	MSR_S3_PMON_EVNTSELO	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 0		Package
Register Address: 740H, 1856	MSR_S3_PMON_EVNTSEL1	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 1		Package
Register Address: 741H, 1857	MSR_S3_PMON_EVNTSEL2	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 2		Package
Register Address: 742H, 1858	MSR_S3_PMON_EVNTSEL3	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 3		Package
Register Address: 743H, 1859	MSR_S3_PMON_BOX_FILTER	
Uncore SBo 3 PerfMon Box-Wide Filter		Package
Register Address: 744H, 1860	MSR_S3_PMON_CTR0	
Uncore SBo 3 PerfMon Counter 0		Package
Register Address: 745H, 1861	MSR_S3_PMON_CTR1	
Uncore SBo 3 PerfMon Counter 1		Package
Register Address: 746H, 1862	MSR_S3_PMON_CTR2	
Uncore SBo 3 PerfMon Counter 2		Package
Register Address: 747H, 1863	MSR_S3_PMON_CTR3	
Uncore SBo 3 PerfMon Counter 3		Package
Register Address: E00H, 3584	MSR_CO_PMON_BOX_CTL	
Uncore C-Box 0 PerfMon for Box-Wide Control		Package
Register Address: E01H, 3585	MSR_CO_PMON_EVNTSELO	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 0		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E02H, 3586	MSR_CO_PMON_EVTSEL1	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 1		Package
Register Address: E03H, 3587	MSR_CO_PMON_EVTSEL2	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 2		Package
Register Address: E04H, 3588	MSR_CO_PMON_EVTSEL3	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 3		Package
Register Address: E05H, 3589	MSR_CO_PMON_BOX_FILTER0	
Uncore C-Box 0 PerfMon Box Wide Filter 0		Package
Register Address: E06H, 3590	MSR_CO_PMON_BOX_FILTER1	
Uncore C-Box 0 PerfMon Box Wide Filter 1		Package
Register Address: E07H, 3591	MSR_CO_PMON_BOX_STATUS	
Uncore C-Box 0 PerfMon Box Wide Status		Package
Register Address: E08H, 3592	MSR_CO_PMON_CTR0	
Uncore C-Box 0 PerfMon Counter 0		Package
Register Address: E09H, 3593	MSR_CO_PMON_CTR1	
Uncore C-Box 0 PerfMon Counter 1		Package
Register Address: E0AH, 3594	MSR_CO_PMON_CTR2	
Uncore C-Box 0 PerfMon Counter 2		Package
Register Address: E0BH, 3595	MSR_CO_PMON_CTR3	
Uncore C-Box 0 PerfMon Counter 3		Package
Register Address: E10H, 3600	MSR_C1_PMON_BOX_CTL	
Uncore C-Box 1 PerfMon for Box-Wide Control		Package
Register Address: E11H, 3601	MSR_C1_PMON_EVTSEL0	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 0		Package
Register Address: E12H, 3602	MSR_C1_PMON_EVTSEL1	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 1		Package
Register Address: E13H, 3603	MSR_C1_PMON_EVTSEL2	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 2		Package
Register Address: E14H, 3604	MSR_C1_PMON_EVTSEL3	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 3		Package
Register Address: E15H, 3605	MSR_C1_PMON_BOX_FILTER0	
Uncore C-Box 1 PerfMon Box Wide Filter 0		Package
Register Address: E16H, 3606	MSR_C1_PMON_BOX_FILTER1	
Uncore C-Box 1 PerfMon Box Wide Filter 1		Package
Register Address: E17H, 3607	MSR_C1_PMON_BOX_STATUS	
Uncore C-Box 1 PerfMon Box Wide Status		Package
Register Address: E18H, 3608	MSR_C1_PMON_CTR0	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 1 PerfMon Counter 0		Package
Register Address: E19H, 3609	MSR_C1_PMON_CTR1	
Uncore C-Box 1 PerfMon Counter 1		Package
Register Address: E1AH, 3610	MSR_C1_PMON_CTR2	
Uncore C-Box 1 PerfMon Counter 2		Package
Register Address: E1BH, 3611	MSR_C1_PMON_CTR3	
Uncore C-Box 1 PerfMon Counter 3		Package
Register Address: E20H, 3616	MSR_C2_PMON_BOX_CTL	
Uncore C-Box 2 PerfMon for Box-Wide Control		Package
Register Address: E21H, 3617	MSR_C2_PMON_EVNTSELO	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 0		Package
Register Address: E22H, 3618	MSR_C2_PMON_EVNTSEL1	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 1		Package
Register Address: E23H, 3619	MSR_C2_PMON_EVNTSEL2	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 2		Package
Register Address: E24H, 3620	MSR_C2_PMON_EVNTSEL3	
Uncore C-Box 2 PerfMon Event select for C-Box 2 Counter 3		Package
Register Address: E25H, 3621	MSR_C2_PMON_BOX_FILTER0	
Uncore C-Box 2 PerfMon Box Wide Filter 0		Package
Register Address: E26H, 3622	MSR_C2_PMON_BOX_FILTER1	
Uncore C-Box 2 PerfMon Box Wide Filter1		Package
Register Address: E27H, 3623	MSR_C2_PMON_BOX_STATUS	
Uncore C-Box 2 PerfMon Box Wide Status		Package
Register Address: E28H, 3624	MSR_C2_PMON_CTR0	
Uncore C-Box 2 PerfMon Counter 0		Package
Register Address: E29H, 3625	MSR_C2_PMON_CTR1	
Uncore C-Box 2 PerfMon Counter 1		Package
Register Address: E2AH, 3626	MSR_C2_PMON_CTR2	
Uncore C-Box 2 PerfMon Counter 2		Package
Register Address: E2BH, 3627	MSR_C2_PMON_CTR3	
Uncore C-Box 2 PerfMon Counter 3		Package
Register Address: E30H, 3632	MSR_C3_PMON_BOX_CTL	
Uncore C-Box 3 PerfMon for Box-Wide Control		Package
Register Address: E31H, 3633	MSR_C3_PMON_EVNTSELO	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 0		Package
Register Address: E32H, 3634	MSR_C3_PMON_EVNTSEL1	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 1		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E33H, 3635	MSR_C3_PMON_EVTSEL2	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 2		Package
Register Address: E34H, 3636	MSR_C3_PMON_EVTSEL3	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 3		Package
Register Address: E35H, 3637	MSR_C3_PMON_BOX_FILTER0	
Uncore C-Box 3 PerfMon Box Wide Filter 0		Package
Register Address: E36H, 3638	MSR_C3_PMON_BOX_FILTER1	
Uncore C-Box 3 PerfMon Box Wide Filter1		Package
Register Address: E37H, 3639	MSR_C3_PMON_BOX_STATUS	
Uncore C-Box 3 PerfMon Box Wide Status		Package
Register Address: E38H, 3640	MSR_C3_PMON_CTR0	
Uncore C-Box 3 PerfMon Counter 0		Package
Register Address: E39H, 3641	MSR_C3_PMON_CTR1	
Uncore C-Box 3 PerfMon Counter 1		Package
Register Address: E3AH, 3642	MSR_C3_PMON_CTR2	
Uncore C-Box 3 PerfMon Counter 2		Package
Register Address: E3BH, 3643	MSR_C3_PMON_CTR3	
Uncore C-Box 3 PerfMon Counter 3		Package
Register Address: E40H, 3648	MSR_C4_PMON_BOX_CTL	
Uncore C-Box 4 PerfMon for Box-Wide Control		Package
Register Address: E41H, 3649	MSR_C4_PMON_EVTSELO	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 0		Package
Register Address: E42H, 3650	MSR_C4_PMON_EVTSEL1	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 1		Package
Register Address: E43H, 3651	MSR_C4_PMON_EVTSEL2	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 2		Package
Register Address: E44H, 3652	MSR_C4_PMON_EVTSEL3	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 3		Package
Register Address: E45H, 3653	MSR_C4_PMON_BOX_FILTER0	
Uncore C-Box 4 PerfMon Box Wide Filter 0		Package
Register Address: E46H, 3654	MSR_C4_PMON_BOX_FILTER1	
Uncore C-Box 4 PerfMon Box Wide Filter1		Package
Register Address: E47H, 3655	MSR_C4_PMON_BOX_STATUS	
Uncore C-Box 4 PerfMon Box Wide Status		Package
Register Address: E48H, 3656	MSR_C4_PMON_CTR0	
Uncore C-Box 4 PerfMon Counter 0		Package
Register Address: E49H, 3657	MSR_C4_PMON_CTR1	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 4 PerfMon Counter 1		Package
Register Address: E4AH, 3658	MSR_C4_PMON_CTR2	
Uncore C-Box 4 PerfMon Counter 2		Package
Register Address: E4BH, 3659	MSR_C4_PMON_CTR3	
Uncore C-Box 4 PerfMon Counter 3		Package
Register Address: E50H, 3664	MSR_C5_PMON_BOX_CTL	
Uncore C-Box 5 PerfMon for Box-Wide Control		Package
Register Address: E51H, 3665	MSR_C5_PMON_EVNTSELO	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 0		Package
Register Address: E52H, 3666	MSR_C5_PMON_EVNTSEL1	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 1		Package
Register Address: E53H, 3667	MSR_C5_PMON_EVNTSEL2	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 2		Package
Register Address: E54H, 3668	MSR_C5_PMON_EVNTSEL3	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 3		Package
Register Address: E55H, 3669	MSR_C5_PMON_BOX_FILTER0	
Uncore C-Box 5 PerfMon Box Wide Filter 0		Package
Register Address: E56H, 3670	MSR_C5_PMON_BOX_FILTER1	
Uncore C-Box 5 PerfMon Box Wide Filter 1		Package
Register Address: E57H, 3671	MSR_C5_PMON_BOX_STATUS	
Uncore C-Box 5 PerfMon Box Wide Status		Package
Register Address: E58H, 3672	MSR_C5_PMON_CTR0	
Uncore C-Box 5 PerfMon Counter 0		Package
Register Address: E59H, 3673	MSR_C5_PMON_CTR1	
Uncore C-Box 5 PerfMon Counter 1		Package
Register Address: E5AH, 3674	MSR_C5_PMON_CTR2	
Uncore C-Box 5 PerfMon Counter 2		Package
Register Address: E5BH, 3675	MSR_C5_PMON_CTR3	
Uncore C-Box 5 PerfMon Counter 3		Package
Register Address: E60H, 3680	MSR_C6_PMON_BOX_CTL	
Uncore C-Box 6 PerfMon for Box-Wide Control		Package
Register Address: E61H, 3681	MSR_C6_PMON_EVNTSELO	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 0		Package
Register Address: E62H, 3682	MSR_C6_PMON_EVNTSEL1	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 1		Package
Register Address: E63H, 3683	MSR_C6_PMON_EVNTSEL2	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 2		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E64H, 3684	MSR_C6_PMON_EVTSEL3	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 3		Package
Register Address: E65H, 3685	MSR_C6_PMON_BOX_FILTER0	
Uncore C-Box 6 PerfMon Box Wide Filter 0		Package
Register Address: E66H, 3686	MSR_C6_PMON_BOX_FILTER1	
Uncore C-Box 6 PerfMon Box Wide Filter 1		Package
Register Address: E67H, 3687	MSR_C6_PMON_BOX_STATUS	
Uncore C-Box 6 PerfMon Box Wide Status		Package
Register Address: E68H, 3688	MSR_C6_PMON_CTR0	
Uncore C-Box 6 PerfMon Counter 0		Package
Register Address: E69H, 3689	MSR_C6_PMON_CTR1	
Uncore C-Box 6 PerfMon Counter 1		Package
Register Address: E6AH, 3690	MSR_C6_PMON_CTR2	
Uncore C-Box 6 PerfMon Counter 2		Package
Register Address: E6BH, 3691	MSR_C6_PMON_CTR3	
Uncore C-Box 6 PerfMon Counter 3		Package
Register Address: E70H, 3696	MSR_C7_PMON_BOX_CTL	
Uncore C-Box 7 PerfMon for Box-Wide Control		Package
Register Address: E71H, 3697	MSR_C7_PMON_EVTSELO	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 0		Package
Register Address: E72H, 3698	MSR_C7_PMON_EVTSEL1	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 1		Package
Register Address: E73H, 3699	MSR_C7_PMON_EVTSEL2	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 2		Package
Register Address: E74H, 3700	MSR_C7_PMON_EVTSEL3	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 3		Package
Register Address: E75H, 3701	MSR_C7_PMON_BOX_FILTER0	
Uncore C-Box 7 PerfMon Box Wide Filter 0		Package
Register Address: E76H, 3702	MSR_C7_PMON_BOX_FILTER1	
Uncore C-Box 7 PerfMon Box Wide Filter 1		Package
Register Address: E77H, 3703	MSR_C7_PMON_BOX_STATUS	
Uncore C-Box 7 PerfMon Box Wide Status		Package
Register Address: E78H, 3704	MSR_C7_PMON_CTR0	
Uncore C-Box 7 PerfMon Counter 0		Package
Register Address: E79H, 3705	MSR_C7_PMON_CTR1	
Uncore C-Box 7 PerfMon Counter 1		Package
Register Address: E7AH, 3706	MSR_C7_PMON_CTR2	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 7 PerfMon Counter 2		Package
Register Address: E7BH, 3707	MSR_C7_PMON_CTR3	
Uncore C-Box 7 PerfMon Counter 3		Package
Register Address: E80H, 3712	MSR_C8_PMON_BOX_CTL	
Uncore C-Box 8 PerfMon Local Box Wide Control		Package
Register Address: E81H, 3713	MSR_C8_PMON_EVNTSELO	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 0		Package
Register Address: E82H, 3714	MSR_C8_PMON_EVNTSEL1	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 1		Package
Register Address: E83H, 3715	MSR_C8_PMON_EVNTSEL2	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 2		Package
Register Address: E84H, 3716	MSR_C8_PMON_EVNTSEL3	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 3		Package
Register Address: E85H, 3717	MSR_C8_PMON_BOX_FILTER0	
Uncore C-Box 8 PerfMon Box Wide Filter 0		Package
Register Address: E86H, 3718	MSR_C8_PMON_BOX_FILTER1	
Uncore C-Box 8 PerfMon Box Wide Filter 1		Package
Register Address: E87H, 3719	MSR_C8_PMON_BOX_STATUS	
Uncore C-Box 8 PerfMon Box Wide Status		Package
Register Address: E88H, 3720	MSR_C8_PMON_CTR0	
Uncore C-Box 8 PerfMon Counter 0		Package
Register Address: E89H, 3721	MSR_C8_PMON_CTR1	
Uncore C-Box 8 PerfMon Counter 1		Package
Register Address: E8AH, 3722	MSR_C8_PMON_CTR2	
Uncore C-Box 8 PerfMon Counter 2		Package
Register Address: E8BH, 3723	MSR_C8_PMON_CTR3	
Uncore C-Box 8 PerfMon Counter 3		Package
Register Address: E90H, 3728	MSR_C9_PMON_BOX_CTL	
Uncore C-Box 9 PerfMon Local Box Wide Control		Package
Register Address: E91H, 3729	MSR_C9_PMON_EVNTSELO	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 0		Package
Register Address: E92H, 3730	MSR_C9_PMON_EVNTSEL1	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 1		Package
Register Address: E93H, 3731	MSR_C9_PMON_EVNTSEL2	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 2		Package
Register Address: E94H, 3732	MSR_C9_PMON_EVNTSEL3	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 3		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E95H, 3733	MSR_C9_PMON_BOX_FILTER0	
Uncore C-Box 9 PerfMon Box Wide Filter 0		Package
Register Address: E96H, 3734	MSR_C9_PMON_BOX_FILTER1	
Uncore C-Box 9 PerfMon Box Wide Filter 1		Package
Register Address: E97H, 3735	MSR_C9_PMON_BOX_STATUS	
Uncore C-Box 9 PerfMon Box Wide Status		Package
Register Address: E98H, 3736	MSR_C9_PMON_CTR0	
Uncore C-Box 9 PerfMon Counter 0		Package
Register Address: E99H, 3737	MSR_C9_PMON_CTR1	
Uncore C-Box 9 PerfMon Counter 1		Package
Register Address: E9AH, 3738	MSR_C9_PMON_CTR2	
Uncore C-Box 9 PerfMon Counter 2		Package
Register Address: E9BH, 3739	MSR_C9_PMON_CTR3	
Uncore C-Box 9 PerfMon Counter 3		Package
Register Address: EA0H, 3744	MSR_C10_PMON_BOX_CTL	
Uncore C-Box 10 PerfMon Local Box Wide Control		Package
Register Address: EA1H, 3745	MSR_C10_PMON_EVTNSELO	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 0		Package
Register Address: EA2H, 3746	MSR_C10_PMON_EVTNSEL1	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 1		Package
Register Address: EA3H, 3747	MSR_C10_PMON_EVTNSEL2	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 2		Package
Register Address: EA4H, 3748	MSR_C10_PMON_EVTNSEL3	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 3		Package
Register Address: EA5H, 3749	MSR_C10_PMON_BOX_FILTER0	
Uncore C-Box 10 PerfMon Box Wide Filter 0		Package
Register Address: EA6H, 3750	MSR_C10_PMON_BOX_FILTER1	
Uncore C-Box 10 PerfMon Box Wide Filter 1		Package
Register Address: EA7H, 3751	MSR_C10_PMON_BOX_STATUS	
Uncore C-Box 10 PerfMon Box Wide Status		Package
Register Address: EA8H, 3752	MSR_C10_PMON_CTR0	
Uncore C-Box 10 PerfMon Counter 0		Package
Register Address: EA9H, 3753	MSR_C10_PMON_CTR1	
Uncore C-Box 10 PerfMon Counter 1		Package
Register Address: EAAH, 3754	MSR_C10_PMON_CTR2	
Uncore C-Box 10 PerfMon Counter 2		Package
Register Address: EABH, 3755	MSR_C10_PMON_CTR3	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 10 PerfMon Counter 3		Package
Register Address: EB0H, 3760	MSR_C11_PMON_BOX_CTL	
Uncore C-Box 11 PerfMon Local Box Wide Control		Package
Register Address: EB1H, 3761	MSR_C11_PMON_EVTNSELO	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 0		Package
Register Address: EB2H, 3762	MSR_C11_PMON_EVTNSEL1	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 1		Package
Register Address: EB3H, 3763	MSR_C11_PMON_EVTNSEL2	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 2		Package
Register Address: EB4H, 3764	MSR_C11_PMON_EVTNSEL3	
Uncore C-box 11 PerfMon Event Select for C-Box 11 Counter 3		Package
Register Address: EB5H, 3765	MSR_C11_PMON_BOX_FILTER0	
Uncore C-Box 11 PerfMon Box Wide Filter 0		Package
Register Address: EB6H, 3766	MSR_C11_PMON_BOX_FILTER1	
Uncore C-Box 11 PerfMon Box Wide Filter 1		Package
Register Address: EB7H, 3767	MSR_C11_PMON_BOX_STATUS	
Uncore C-Box 11 PerfMon Box Wide Status		Package
Register Address: EB8H, 3768	MSR_C11_PMON_CTR0	
Uncore C-Box 11 PerfMon Counter 0		Package
Register Address: EB9H, 3769	MSR_C11_PMON_CTR1	
Uncore C-Box 11 PerfMon Counter 1		Package
Register Address: EBAH, 3770	MSR_C11_PMON_CTR2	
Uncore C-Box 11 PerfMon Counter 2		Package
Register Address: EBBH, 3771	MSR_C11_PMON_CTR3	
Uncore C-Box 11 PerfMon Counter 3		Package
Register Address: EC0H, 3776	MSR_C12_PMON_BOX_CTL	
Uncore C-Box 12 PerfMon Local Box Wide Control		Package
Register Address: EC1H, 3777	MSR_C12_PMON_EVTNSELO	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 0		Package
Register Address: EC2H, 3778	MSR_C12_PMON_EVTNSEL1	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 1		Package
Register Address: EC3H, 3779	MSR_C12_PMON_EVTNSEL2	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 2		Package
Register Address: EC4H, 3780	MSR_C12_PMON_EVTNSEL3	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 3		Package
Register Address: EC5H, 3781	MSR_C12_PMON_BOX_FILTER0	
Uncore C-Box 12 PerfMon Box Wide Filter 0		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EC6H, 3782	MSR_C12_PMON_BOX_FILTER1	
Uncore C-Box 12 PerfMon Box Wide Filter 1		Package
Register Address: EC7H, 3783	MSR_C12_PMON_BOX_STATUS	
Uncore C-Box 12 PerfMon Box Wide Status		Package
Register Address: EC8H, 3784	MSR_C12_PMON_CTR0	
Uncore C-Box 12 PerfMon Counter 0		Package
Register Address: EC9H, 3785	MSR_C12_PMON_CTR1	
Uncore C-Box 12 PerfMon Counter 1		Package
Register Address: ECAH, 3786	MSR_C12_PMON_CTR2	
Uncore C-Box 12 PerfMon Counter 2		Package
Register Address: ECBH, 3787	MSR_C12_PMON_CTR3	
Uncore C-Box 12 PerfMon Counter 3		Package
Register Address: ED0H, 3792	MSR_C13_PMON_BOX_CTL	
Uncore C-Box 13 PerfMon local box wide control.		Package
Register Address: ED1H, 3793	MSR_C13_PMON_EVTSELO	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 0		Package
Register Address: ED2H, 3794	MSR_C13_PMON_EVTSEL1	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 1		Package
Register Address: ED3H, 3795	MSR_C13_PMON_EVTSEL2	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 2		Package
Register Address: ED4H, 3796	MSR_C13_PMON_EVTSEL3	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 3		Package
Register Address: ED5H, 3797	MSR_C13_PMON_BOX_FILTER0	
Uncore C-Box 13 PerfMon Box Wide Filter 0		Package
Register Address: ED6H, 3798	MSR_C13_PMON_BOX_FILTER1	
Uncore C-Box 13 PerfMon Box Wide Filter 1		Package
Register Address: ED7H, 3799	MSR_C13_PMON_BOX_STATUS	
Uncore C-Box 13 PerfMon Box Wide Status		Package
Register Address: ED8H, 3800	MSR_C13_PMON_CTR0	
Uncore C-Box 13 PerfMon Counter 0		Package
Register Address: ED9H, 3801	MSR_C13_PMON_CTR1	
Uncore C-Box 13 PerfMon Counter 1		Package
Register Address: EDAH, 3802	MSR_C13_PMON_CTR2	
Uncore C-Box 13 PerfMon Counter 2		Package
Register Address: EDBH, 3803	MSR_C13_PMON_CTR3	
Uncore C-Box 13 PerfMon Counter 3		Package
Register Address: EE0H, 3808	MSR_C14_PMON_BOX_CTL	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 14 PerfMon Local Box Wide Control		Package
Register Address: EE1H, 3809	MSR_C14_PMON_EVNTSEL0	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 0		Package
Register Address: EE2H, 3810	MSR_C14_PMON_EVNTSEL1	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 1		Package
Register Address: EE3H, 3811	MSR_C14_PMON_EVNTSEL2	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 2		Package
Register Address: EE4H, 3812	MSR_C14_PMON_EVNTSEL3	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 3		Package
Register Address: EE5H, 3813	MSR_C14_PMON_BOX_FILTER	
Uncore C-Box 14 PerfMon Box Wide Filter 0		Package
Register Address: EE6H, 3814	MSR_C14_PMON_BOX_FILTER1	
Uncore C-Box 14 PerfMon Box Wide Filter 1		Package
Register Address: EE7H, 3815	MSR_C14_PMON_BOX_STATUS	
Uncore C-Box 14 PerfMon Box Wide Status		Package
Register Address: EE8H, 3816	MSR_C14_PMON_CTR0	
Uncore C-Box 14 PerfMon Counter 0		Package
Register Address: EE9H, 3817	MSR_C14_PMON_CTR1	
Uncore C-Box 14 PerfMon Counter 1		Package
Register Address: EEAH, 3818	MSR_C14_PMON_CTR2	
Uncore C-Box 14 PerfMon Counter 2		Package
Register Address: EEBH, 3819	MSR_C14_PMON_CTR3	
Uncore C-Box 14 PerfMon Counter 3		Package
Register Address: EF0H, 3824	MSR_C15_PMON_BOX_CTL	
Uncore C-Box 15 PerfMon Local Box Wide Control		Package
Register Address: EF1H, 3825	MSR_C15_PMON_EVNTSEL0	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 0		Package
Register Address: EF2H, 3826	MSR_C15_PMON_EVNTSEL1	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 1		Package
Register Address: EF3H, 3827	MSR_C15_PMON_EVNTSEL2	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 2		Package
Register Address: EF4H, 3828	MSR_C15_PMON_EVNTSEL3	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 3		Package
Register Address: EF5H, 3829	MSR_C15_PMON_BOX_FILTER0	
Uncore C-Box 15 PerfMon Box Wide Filter 0		Package
Register Address: EF6H, 3830	MSR_C15_PMON_BOX_FILTER1	
Uncore C-Box 15 PerfMon Box Wide Filter 1		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EF7H, 3831	MSR_C15_PMON_BOX_STATUS	
Uncore C-Box 15 PerfMon Box Wide Status		Package
Register Address: EF8H, 3832	MSR_C15_PMON_CTR0	
Uncore C-Box 15 PerfMon Counter 0		Package
Register Address: EF9H, 3833	MSR_C15_PMON_CTR1	
Uncore C-Box 15 PerfMon Counter 1		Package
Register Address: EFAH, 3834	MSR_C15_PMON_CTR2	
Uncore C-Box 15 PerfMon Counter 2		Package
Register Address: EFBH, 3835	MSR_C15_PMON_CTR3	
Uncore C-Box 15 PerfMon Counter 3		Package
Register Address: F00H, 3840	MSR_C16_PMON_BOX_CTL	
Uncore C-Box 16 PerfMon for Box-Wide Control		Package
Register Address: F01H, 3841	MSR_C16_PMON_EVTSELO	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 0		Package
Register Address: F02H, 3842	MSR_C16_PMON_EVTSEL1	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 1		Package
Register Address: F03H, 3843	MSR_C16_PMON_EVTSEL2	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 2		Package
Register Address: F04H, 3844	MSR_C16_PMON_EVTSEL3	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 3		Package
Register Address: F05H, 3845	MSR_C16_PMON_BOX_FILTER0	
Uncore C-Box 16 PerfMon Box Wide Filter 0		Package
Register Address: F06H, 3846	MSR_C16_PMON_BOX_FILTER1	
Uncore C-Box 16 PerfMon Box Wide Filter 1		Package
Register Address: F07H, 3847	MSR_C16_PMON_BOX_STATUS	
Uncore C-Box 16 PerfMon Box Wide Status		Package
Register Address: F08H, 3848	MSR_C16_PMON_CTR0	
Uncore C-Box 16 PerfMon Counter 0		Package
Register Address: F09H, 3849	MSR_C16_PMON_CTR1	
Uncore C-Box 16 PerfMon Counter 1		Package
Register Address: FOAH, 3850	MSR_C16_PMON_CTR2	
Uncore C-Box 16 PerfMon Counter 2		Package
Register Address: F0BH, 3851	MSR_C16_PMON_CTR3	
Uncore C-Box 16 PerfMon Counter 3		Package
Register Address: F10H, 3856	MSR_C17_PMON_BOX_CTL	
Uncore C-Box 17 PerfMon for Box-Wide Control		Package
Register Address: F11H, 3857	MSR_C17_PMON_EVTSELO	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 0		Package
Register Address: F12H, 3858	MSR_C17_PMON_EVTNSEL1	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 1		Package
Register Address: F13H, 3859	MSR_C17_PMON_EVTNSEL2	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 2		Package
Register Address: F14H, 3860	MSR_C17_PMON_EVTNSEL3	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 3		Package
Register Address: F15H, 3861	MSR_C17_PMON_BOX_FILTER0	
Uncore C-Box 17 PerfMon Box Wide Filter 0		Package
Register Address: F16H, 3862	MSR_C17_PMON_BOX_FILTER1	
Uncore C-Box 17 PerfMon Box Wide Filter 1		Package
Register Address: F17H, 3863	MSR_C17_PMON_BOX_STATUS	
Uncore C-Box 17 PerfMon Box Wide Status		Package
Register Address: F18H, 3864	MSR_C17_PMON_CTR0	
Uncore C-Box 17 PerfMon Counter 0		Package
Register Address: F19H, 3865	MSR_C17_PMON_CTR1	
Uncore C-Box 17 PerfMon Counter 1		Package
Register Address: F1AH, 3866	MSR_C17_PMON_CTR2	
Uncore C-Box 17 PerfMon Counter 2		Package
Register Address: F1BH, 3867	MSR_C17_PMON_CTR3	
Uncore C-Box 17 PerfMon Counter 3		Package

2.15 MSRS IN THE INTEL® CORE™ M PROCESSORS AND THE 5TH GENERATION INTEL® CORE™ PROCESSORS

The Intel® Core™ M-5xxx processors, 5th generation Intel® Core™ Processors, and the Intel® Xeon® Processor E3-1200 v4 family are based on Broadwell microarchitecture. The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH. The Intel® Xeon® Processor E3-1200 v4 family and 5th generation Intel® Core™ Processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_47H. Processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH or 06_47H support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, Table 2-34, and Table 2-35. For an MSR listed in Table 2-35 that also appears in the model-specific tables of prior generations, Table 2-35 supersedes prior generation tables.

Table 2-34 lists MSRs that are common to processors based on the Broadwell microarchitectures (including CPUID Signature DisplayFamily_DisplayModel values of 06_3DH, 06_47H, 06_4FH, and 06_56H).

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		Thread
0	Ovf_PMC0	
1	Ovf_PMC1	
2	Ovf_PMC2	
3	Ovf_PMC3	
31:4	Reserved	
32	Ovf_FixedCtr0	
33	Ovf_FixedCtr1	
34	Ovf_FixedCtr2	
54:35	Reserved.	
55	Trace_ToPA_PMI See Section 36.2.7.2, "Table of Physical Addresses (ToPA)."	
60:56	Reserved.	
61	Ovf_Uncore	
62	Ovf_BufDSSAVE	
63	CondChgd	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		Thread
0	Set 1 to clear Ovf_PMC0.	
1	Set 1 to clear Ovf_PMC1.	
2	Set 1 to clear Ovf_PMC2.	
3	Set 1 to clear Ovf_PMC3.	
31:4	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	
33	Set 1 to clear Ovf_FixedCtr1.	
34	Set 1 to clear Ovf_FixedCtr2	
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI. See Section 36.2.7.2, "Table of Physical Addresses (ToPA)."	
60:56	Reserved.	
61	Set 1 to clear Ovf_Uncore.	
62	Set 1 to clear Ovf_BufDSSAVE.	
63	Set 1 to clear CondChgd.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W)		Thread
6:0	Reserved.	
M-1:7	Base physical address. M is the value enumerated by CPUID.80000008H:EAX[7:0].	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:M	Reserved.	
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W)		Thread
6:0	Reserved.	
31:7	MaskOrTableOffset	
63:32	Output Offset.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Thread
0	TraceEn	
1	Reserved, must be zero.	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	Reserved, must be zero.	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	Reserved; writing 0 will #GP if also setting TraceEn.	
63:14	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Thread
0	Reserved, writes ignored.	
1	ContexEn, writes ignored.	
2	TriggerEn, writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
63:6	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Thread
4:0	Reserved.	
63:5	CR3[63:5] value to match.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	

Table 2-35 lists MSRs that are specific to Intel Core M processors and 5th Generation Intel Core Processors.

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/W0)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Enable Package C-State Auto-Demotion (R/W)	
30	Enable Package C-State Undemotion (R/W)	
63:31	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6core active.	Package
63:48	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, and Table 2-34 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH.		

2.16 MSRS IN THE INTEL® XEON® PROCESSOR E5 V4 FAMILY

The MSRs listed in Table 2-36 are available and common to the Intel® Xeon® Processor D Product Family (CPUID Signature DisplayFamily_DisplayModel value of 06_56H) and to the Intel Xeon processors E5 v4 and E7 v4 families (CPUID Signature DisplayFamily_DisplayModel value of 06_4FH). These processors are based on Broadwell microarchitecture.

See Section 2.16.1 for lists of tables of MSRs that are supported by the Intel® Xeon® Processor D Family.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Protected Processor Inventory Number	Enable Control (R/W)	Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number	(R/O)	Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information		Package
Contains power management and other model specific features enumeration. See http://biosbits.org .		
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) See Table 2-26.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) See Table 2-26.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.	Package
30	Programmable TJ OFFSET (R/O) See Table 2-26.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) See Table 2-26.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W)		Core
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/WO)	
16	Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).	
24:17	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Current Limit Log (R/WC0) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WC0) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) See Table 2-26.	
27:24	TCC Activation Offset (R/W) See Table 2-26.	
63:28	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C	Package
15:8	Maximum Ratio Limit for 2C	Package
23:16	Maximum Ratio Limit for 3C	Package
31:24	Maximum Ratio Limit for 4C	Package
39:32	Maximum Ratio Limit for 5C	Package
47:40	Maximum Ratio Limit for 6C	Package
55:48	Maximum Ratio Limit for 7C	Package
63:56	Maximum Ratio Limit for 8C	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C	Package
15:8	Maximum Ratio Limit for 10C	Package

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23:16	Maximum Ratio Limit for 11C	Package
31:24	Maximum Ratio Limit for 12C	Package
39:32	Maximum Ratio Limit for 13C	Package
47:40	Maximum Ratio Limit for 14C	Package
55:48	Maximum Ratio Limit for 15C	Package
63:56	Maximum Ratio Limit for 16C	Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
63:15	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
7	Reserved.	
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit.	
3	Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit.	
4	Reserved.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.	
12:11	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.	
14	Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.	
15	Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20	Reserved.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28:27	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
31	Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:32	Reserved.	
Register Address: 770H, 1904	IA32_PM_ENABLE	
See Section 17.4.2, "Enabling HWP."		Package
Register Address: 771H, 1905	IA32_HWP_CAPABILITIES	
See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."		Thread
Register Address: 774H, 1908	IA32_HWP_REQUEST	
See Section 17.4.4, "Managing HWP."		Thread
7:0	Minimum Performance (R/W)	
15:8	Maximum Performance (R/W)	
23:16	Desired Performance (R/W)	
63:24	Reserved.	
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Section 17.4.5, "HWP Feedback."		Thread
1:0	Reserved.	
2	Excursion to Minimum (R/O)	
63:3	Reserved.	
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.	
31:8	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
31:10	Reserved.	
51:32	CLOS (R/W)	
63: 52	Reserved.	
Register Address: C90H, 3216	IA32_L3_QOS_MASK_0	
L3 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 0 enforcement.	
63:20	Reserved.	
Register Address: C91H, 3217	IA32_L3_QOS_MASK_1	
L3 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 1 enforcement.	
63:20	Reserved.	
Register Address: C92H, 3218	IA32_L3_QOS_MASK_2	
L3 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 2 enforcement.	
63:20	Reserved.	
Register Address: C93H, 3219	IA32_L3_QOS_MASK_3	
L3 Class Of Service Mask - CLOS 3 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: C94H, 3220	IA32_L3_QOS_MASK_4	
L3 Class Of Service Mask - CLOS 4 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 4.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 4 enforcement.	
63:20	Reserved.	
Register Address: C95H, 3221	IA32_L3_QOS_MASK_5	
L3 Class Of Service Mask - CLOS 5 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 5.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 5 enforcement.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: C96H, 3222	IA32_L3_QOS_MASK_6	
L3 Class Of Service Mask - CLOS 6 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 6.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 6 enforcement.	
63:20	Reserved.	
Register Address: C97H, 3223	IA32_L3_QOS_MASK_7	
L3 Class Of Service Mask - CLOS 7 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 7.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 7 enforcement.	
63:20	Reserved.	
Register Address: C98H, 3224	IA32_L3_QOS_MASK_8	
L3 Class Of Service Mask - CLOS 8 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 8.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 8 enforcement.	
63:20	Reserved.	
Register Address: C99H, 3225	IA32_L3_QOS_MASK_9	
L3 Class Of Service Mask - CLOS 9 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 9.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 9 enforcement.	
63:20	Reserved.	
Register Address: C9AH, 3226	IA32_L3_QOS_MASK_10	
L3 Class Of Service Mask - CLOS 10 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 10.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 10 enforcement.	
63:20	Reserved.	
Register Address: C9BH, 3227	IA32_L3_QOS_MASK_11	
L3 Class Of Service Mask - CLOS 11 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 11.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 11 enforcement.	
63:20	Reserved.	
Register Address: C9CH, 3228	IA32_L3_QOS_MASK_12	
L3 Class Of Service Mask - CLOS 12 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 12.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 12 enforcement.	
63:20	Reserved.	
Register Address: C9DH, 3229	IA32_L3_QOS_MASK_13	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
L3 Class Of Service Mask - CLOS 13 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 13.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 13 enforcement.	
63:20	Reserved.	
Register Address: C9EH, 3230	IA32_L3_QOS_MASK_14	
L3 Class Of Service Mask - CLOS 14 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 14.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 14 enforcement.	
63:20	Reserved.	
Register Address: C9FH, 3231	IA32_L3_QOS_MASK_15	
L3 Class Of Service Mask - CLOS 15 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 15.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 15 enforcement.	
63:20	Reserved.	

2.16.1 Additional MSRs Supported in the Intel® Xeon® Processor D Product Family

The MSRs listed in Table 2-37 are available to Intel® Xeon® Processor D Product Family (CPUID Signature DisplayFamily_DisplayModel value of 06_56H). The Intel® Xeon® processor D product family is based on Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-20, Table 2-29, Table 2-34, Table 2-36, and Table 2-37.

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ACH, 428	MSR_TURBO_RATIO_LIMIT3	
Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
62:0	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 449H, 1097	IA32_MC18_STATUS	

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
See Table 2-20, Table 2-29, Table 2-34, and Table 2-36 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_56H.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.16.2 Additional MSRs Supported in Intel® Xeon® Processors E5 v4 and E7 v4 Families

The MSRs listed in Table 2-37 are available to the Intel® Xeon® Processor E5 v4 and E7 v4 Families (CPUID Signature DisplayFamily_DisplayModel value of 06_4FH). The Intel® Xeon® processor E5 v4 family is based on Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-29, Table 2-34, Table 2-36, and Table 2-38.

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ACH, 428	MSR_TURBO_RATIO_LIMIT3	
Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
62:0	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1, and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: C81H, 3201	IA32_L3_QOS_CFG	
Cache Allocation Technology Configuration (R/W)		Package
0	CAT Enable. Set 1 to enable Cache Allocation Technology.	
63:1	Reserved.	
See Table 2-20, Table 2-21, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.17 MSRS IN THE 6TH—13TH GENERATION INTEL® CORE™ PROCESSORS, 1ST—5TH GENERATION INTEL® XEON® SCALABLE PROCESSOR FAMILIES, INTEL® CORE™ ULTRA 7 PROCESSORS, 8TH GENERATION INTEL® CORE™ I3 PROCESSORS, INTEL® XEON® E PROCESSORS, INTEL® XEON® 6 P-CORE PROCESSORS, INTEL® XEON® 6 E-CORE PROCESSORS, AND INTEL® SERIES 2 CORE™ ULTRA PROCESSORS

6th generation Intel® Core™ processors are based on Skylake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_4EH or 06_5EH.

The Intel® Xeon® Scalable Processor Family based on the Skylake microarchitecture, the 2nd generation Intel® Xeon® Scalable Processor Family based on the Cascade Lake product, and the 3rd generation Intel® Xeon® Scalable Processor Family based on the Cooper Lake product all have a CPUID Signature DisplayFamily_DisplayModel value of 06_55H.

7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture, 8th generation and 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on Coffee Lake microarchitecture; these processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_8EH or 06_9EH.

8th generation Intel® Core™ i3 processors are based on Cannon Lake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_66H.

10th generation Intel® Core™ processors are based on Comet Lake microarchitecture (with a CPUID Signature DisplayFamily_DisplayModel value of 06_A5H or 06_A6H) and Ice Lake microarchitecture (with a CPUID Signature DisplayFamily_DisplayModel value of 06_7EH).

11th generation Intel® Core™ processors are based on Tiger Lake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_8CH or 06_8DH.

The 3rd generation Intel® Xeon® Scalable Processor Family is based on Ice Lake microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_6AH or 06_6CH.

12th generation Intel® Core™ processors supporting the Alder Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_97H or 06_9AH.

13th generation Intel® Core™ processors supporting the Raptor Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_BAH, 06_B7H, or 06_BFH.

The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_8FH.

The 5th generation Intel® Xeon® Scalable Processor Family is based on Emerald Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_CFH.

The Intel® Core™ Ultra 7 processors supporting the Meteor Lake hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_AAH.

The Intel® Xeon® 6 P-core processor is based on the Granite Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_ADH or 06_AEH.

The Intel® Xeon® 6 E-core processor is based on the Sierra Forest microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_AFH.

The Intel® Series 2 Core™ Ultra processors supporting the Lunar Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_BDH.

These processors support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-25, Table 2-29, Table 2-35, and Table 2-39¹. For an MSR listed in Table 2-39 that also appears in the model-specific tables of prior generations, Table 2-39 supersedes prior generation tables.

Tables 2-40 through 2-60 list additional supported MSR interfaces introduced in specific processors; see each table for additional details.

The notation of “Platform” in the Scope column (with respect to MSR_PLATFORM_ENERGY_COUNTER and MSR_PLATFORM_POWER_LIMIT) is limited to the power-delivery domain and the specifics of the power delivery integration may vary by platform vendor’s implementation.

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
MTRR Capability (R/O, Architectural) See Table 2-2		Thread
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal threshold #1 Status (R/O) See Table 2-2.	
7	Thermal threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	

1. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core: 3F7H. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core or P-core: 652H, 653H, 655H, 656H, DB0H, DB1H, DB2H, and D90H.

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
9	Thermal Threshold #2 Log (R/WCO) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WCO) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	
13	Current Limit Log (R/WCO) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WCO) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.		Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
18:2	Reserved.	
19	Disable Energy Efficiency Optimization (R/W) Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.	
20	Disable Race to Halt Optimization (R/W) Setting this bit disables the Race to Halt optimization and avoids this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization.	
63:21	Reserved.	
Register Address: 300H, 768	MSR_SGXOWNEREPOCH0	
Lower 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 301H, 769	MSR_SGXOWNEREPOCH1	
Upper 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 22.2.4, "Architectural Performance Monitoring Version 4."		
0	Ovf_PMC0	Thread
1	Ovf_PMC1	Thread
2	Ovf_PMC2	Thread
3	Ovf_PMC3	Thread
4	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	Thread
5	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)	Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)	Thread
7	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)	Thread
31:8	Reserved.	
32	Ovf_FixedCtr0	Thread
33	Ovf_FixedCtr1	Thread
34	Ovf_FixedCtr2	Thread
54:35	Reserved	
55	Trace_ToPA_PMI	Thread
57:56	Reserved.	
58	LBR_Frz	Thread
59	CTR_Frz	Thread
60	ASCI	Thread
61	Ovf_Uncore	Thread
62	Ovf_BufDSSAVE	Thread
63	CondChgd	Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
See Table 2-2 and Section 22.2.4, “Architectural Performance Monitoring Version 4.”		
0	Set 1 to clear Ovf_PMC0.	Thread
1	Set 1 to clear Ovf_PMC1.	Thread
2	Set 1 to clear Ovf_PMC2.	Thread
3	Set 1 to clear Ovf_PMC3.	Thread
4	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).	Thread
5	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).	Thread
6	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).	Thread
7	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).	Thread
31:8	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	Thread
33	Set 1 to clear Ovf_FixedCtr1.	Thread
34	Set 1 to clear Ovf_FixedCtr2.	Thread
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI.	Thread
57:56	Reserved.	
58	Set 1 to clear LBR_Frz.	Thread
59	Set 1 to clear CTR_Frz.	Thread
60	Set 1 to clear ASCI.	Thread

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
61	Set 1 to clear Ovf_Uncore.	Thread
62	Set 1 to clear Ovf_BufDSSAVE.	Thread
63	Set 1 to clear CondChgd.	Thread
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
See Table 2-2 and Section 22.2.4, “Architectural Performance Monitoring Version 4.”		
0	Set 1 to cause Ovf_PMC0 = 1.	Thread
1	Set 1 to cause Ovf_PMC1 = 1.	Thread
2	Set 1 to cause Ovf_PMC2 = 1.	Thread
3	Set 1 to cause Ovf_PMC3 = 1.	Thread
4	Set 1 to cause Ovf_PMC4 =1 (if CPUID.0AH:EAX[15:8] > 4).	Thread
5	Set 1 to cause Ovf_PMC5 =1 (if CPUID.0AH:EAX[15:8] > 5).	Thread
6	Set 1 to cause Ovf_PMC6 =1 (if CPUID.0AH:EAX[15:8] > 6).	Thread
7	Set 1 to cause Ovf_PMC7 =1 (if CPUID.0AH:EAX[15:8] > 7).	Thread
31:8	Reserved.	
32	Set 1 to cause Ovf_FixedCtr0 = 1.	Thread
33	Set 1 to cause Ovf_FixedCtr1 = 1.	Thread
34	Set 1 to cause Ovf_FixedCtr2 = 1.	Thread
54:35	Reserved.	
55	Set 1 to cause Trace_ToPA_PMI = 1.	Thread
57:56	Reserved.	
58	Set 1 to cause LBR_Frz = 1.	Thread
59	Set 1 to cause CTR_Frz = 1.	Thread
60	Set 1 to cause ASCI = 1.	Thread
61	Set 1 to cause Ovf_Uncore.	Thread
62	Set 1 to cause Ovf_BufDSSAVE.	Thread
63	Reserved.	
Register Address: 392H, 914	IA32_PERF_GLOBAL_INUSE	
See Table 2-2.		Thread
Register Address: 3F7H, 1015	MSR_PEBS_FRONTEND	
FrontEnd Precise Event Condition Select (R/W)		Thread
2:0	Event Code Select	
3	Reserved	
4	Event Code Select High	
7:5	Reserved.	
19:8	IDQ_Bubble_Length Specifier	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
22:20	IDQ_Bubble_Width Specifier	
63:23	Reserved.	
Register Address: 500H, 1280	IA32_SGX_SVN_STATUS	
Status and SVN Threshold of SGX Support for ACM (R/O)		Thread
0	Lock See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
15:1	Reserved.	
23:16	SGX_SVN_SINIT See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
63:24	Reserved.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W) See Table 2-2.		Thread
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W) See Table 2-2.		Thread
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Thread
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	BranchEn	
17:14	MTCFreq	
18	Reserved, must be zero.	
22:19	CycThresh	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Thread
0	FilterEn, writes ignored.	
1	ContexEn, writes ignored.	
2	TriggerEn, writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
31:6	Reserved, must be zero.	
48:32	PacketByteCnt	
63:49	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Thread
4:0	Reserved	
63:5	CR3[63:5] value to match	
Register Address: 580H, 1408	IA32_RTIT_ADDR0_A	
Region 0 Start Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 581H, 1409	IA32_RTIT_ADDR0_B	
Region 0 End Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 582H, 1410	IA32_RTIT_ADDR1_A	
Region 1 Start Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 583H, 1411	IA32_RTIT_ADDR1_B	
Region 1 End Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
PP0 Energy Status (R/O) See Section 17.10.4, “PP0/PP1 RAPL Domains.”		Package
Register Address: 64DH, 1613	MSR_PLATFORM_ENERGY_COUNTER	
Platform Energy Counter (R/O) This MSR is valid only if both platform vendor hardware implementation and BIOS enablement support it. This MSR will read 0 if not valid.		Platform
31:0	Total energy consumed by all devices in the platform that receive power from integrated power delivery mechanism, included platform devices are processor cores, SOC, memory, add-on or peripheral devices that get powered directly from the platform power delivery means. The energy units are specified in the MSR_RAPL_POWER_UNIT.Energy_Status_Unit.	
63:32	Reserved.	
Register Address: 64EH, 1614	MSR_PPERF	
Productive Performance Count (R/O)		Thread
63:0	Hardware’s view of workload scalability. See Section 17.4.5.1.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)	Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Residency State Regulation Status (R0) When set, frequency is reduced below the operating system request due to residency state regulation limit.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).	
7	VR Therm Design Current Status (R0) When set, frequency is reduced below the operating system request due to VR thermal design current limit.	
8	Other Status (R0) When set, frequency is reduced below the operating system request due to electrical or other constraints.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.	
12	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Residency State Regulation Log When set, indicates that the Residency State Regulation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR Thermal Design Current Log When set, indicates that the VR TDC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
24	Other Log When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 652H, 1618	MSR_PKG_HDC_CONFIG	
HDC Configuration (R/W)		Package
2:0	PKG_Cx_Monitor Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY.	
63:3	Reserved.	
Register Address: 653H, 1619	MSR_CORE_HDC_RESIDENCY	
Core HDC Idle Residency (R/O)		Core
63:0	Core_Cx_Duty_Cycle_Cnt	
Register Address: 655H, 1621	MSR_PKG_HDC_SHALLOW_RESIDENCY	
Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle (R/O)		Package
63:0	Pkg_C2_Duty_Cycle_Cnt	
Register Address: 656H, 1622	MSR_PKG_HDC_DEEP_RESIDENCY	
Package Cx HDC Idle Residency (R/O)		Package
63:0	Pkg_Cx_Duty_Cycle_Cnt	
Register Address: 658H, 1624	MSR_WEIGHTED_CORE_CO	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Core-count Weighted C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 659H, 1625	MSR_ANY_CORE_C0	
Any Core C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if any processor core in the package is in C0.	
Register Address: 65AH, 1626	MSR_ANY GFXE_C0	
Any Graphics Engine C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.	
Register Address: 65BH, 1627	MSR_CORE GFXE_OVERLAP_C0	
Core and Graphics Engine Overlapped C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W-L) Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor. The processor implements an exponential-weighted algorithm in the placement of the time windows. Note that the bit field widths differ depending on the platform.		Platform
MSR_PLATFORM_POWER_LIMIT for 4th and 5th generation Intel® Xeon® Scalable Processor Families, Intel® Xeon® 6 E-Core Processors, and Intel® Xeon 6 P-core Processors		
16:0	Platform Power Limit #1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
17	Enable Platform Power Limit #1 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #1 over the time window specified by Power Limit #1 Time Window.	
18	Platform Clamping Limitation #1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #1 value. This bit is writeable only when CPUID.06H:EAX[4] is set.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
25:19	Time Window for Platform Power Limit #1 Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[25:24] Y = POWER_LIMIT_1_TIME[23:19] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:26	Reserved.	
48:32	Platform Power Limit #2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit # 1).	
49	Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.	
50	Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.	
62:51	Reserved.	
63	Lock. Setting this bit will lock all other bits of this MSR until system RESET.	
MSR_PLATFORM_POWER_LIMIT for 6th—13th Generation Intel® Core™ Processors, 1st—3rd Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, and Intel® Series 2 Core™ Ultra Processors		
14:0	Platform Power Limit #1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
15	Enable Platform Power Limit #1 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #1 over the time window specified by Power Limit #1 Time Window.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
16	Platform Clamping Limitation #1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #1 value. This bit is writeable only when CPUID.06H:EAX[4] is set.	
23:17	Time Window for Platform Power Limit #1 Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:24	Reserved.	
46:32	Platform Power Limit #2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit # 1).	
47	Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.	
48	Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.	
62:49	Reserved.	
63	Lock. Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 690H, 1680	MSR_LASTBRANCH_16_FROM_IP	
Last Branch Record 16 From IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.12.		Thread
Register Address: 691H, 1681	MSR_LASTBRANCH_17_FROM_IP	
Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 692H, 1682	MSR_LASTBRANCH_18_FROM_IP	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 693H, 1683	MSR_LASTBRANCH_19_FROM_IP	
Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 694H, 1684	MSR_LASTBRANCH_20_FROM_IP	
Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 695H, 1685	MSR_LASTBRANCH_21_FROM_IP	
Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 696H, 1686	MSR_LASTBRANCH_22_FROM_IP	
Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 697H, 1687	MSR_LASTBRANCH_23_FROM_IP	
Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 698H, 1688	MSR_LASTBRANCH_24_FROM_IP	
Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 699H, 1689	MSR_LASTBRANCH_25_FROM_IP	
Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69AH, 1690	MSR_LASTBRANCH_26_FROM_IP	
Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69BH, 1691	MSR_LASTBRANCH_27_FROM_IP	
Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69CH, 1692	MSR_LASTBRANCH_28_FROM_IP	
Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69DH, 1693	MSR_LASTBRANCH_29_FROM_IP	
Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69EH, 1694	MSR_LASTBRANCH_30_FROM_IP	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69FH, 1695	MSR_LASTBRANCH_31_FROM_IP	
Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced due to running average thermal limit.	
6	VR Therm Alert Status (R0) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR Thermal Design Current Status (R0) When set, frequency is reduced due to VR TDC limit.	
8	Other Status (R0) When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
12	Inefficient Operation Status (R0) When set, processor graphics frequency is operating below target frequency.	
15:13	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Inefficient Operation Log When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:29	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced due to assertion of external PROCHOT.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Thermal Status (R0) When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced due to running average thermal limit.	
6	VR Therm Alert Status (R0) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR Thermal Design Current Status (R0) When set, frequency is reduced due to VR TDC limit.	
8	Other Status (R0) When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced due to package/Platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced due to package/Platform-level power limiting PL2/PL3.	
15:12	Reserved	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
23	VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:28	Reserved.	
Register Address: 6D0H, 1744	MSR_LASTBRANCH_16_TO_IP	
Last Branch Record 16 To IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.12.		Thread
Register Address: 6D1H, 1745	MSR_LASTBRANCH_17_TO_IP	
Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D2H, 1746	MSR_LASTBRANCH_18_TO_IP	
Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D3H, 1747	MSR_LASTBRANCH_19_TO_IP	
Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D4H, 1748	MSR_LASTBRANCH_20_TO_IP	
Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D5H, 1749	MSR_LASTBRANCH_21_TO_IP	
Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 6D6H, 1750	MSR_LASTBRANCH_22_TO_IP	
Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D7H, 1751	MSR_LASTBRANCH_23_TO_IP	
Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D8H, 1752	MSR_LASTBRANCH_24_TO_IP	
Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D9H, 1753	MSR_LASTBRANCH_25_TO_IP	
Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DAH, 1754	MSR_LASTBRANCH_26_TO_IP	
Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DBH, 1755	MSR_LASTBRANCH_27_TO_IP	
Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DCH, 1756	MSR_LASTBRANCH_28_TO_IP	
Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DDH, 1757	MSR_LASTBRANCH_29_TO_IP	
Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DEH, 1758	MSR_LASTBRANCH_30_TO_IP	
Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DFH, 1759	MSR_LASTBRANCH_31_TO_IP	
Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 770H, 1904	IA32_PM_ENABLE	
See Section 17.4.2, “Enabling HWP.”		Package
Register Address: 771H, 1905	IA32_HWP_CAPABILITIES	
See Section 17.4.3, “HWP Performance Range and Dynamic Capabilities.”		Thread
Register Address: 772H, 1906	IA32_HWP_REQUEST_PKG	
See Section 17.4.4, “Managing HWP.”		Package

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 773H, 1907	IA32_HWP_INTERRUPT	
See Section 17.4.6, “HWP Notifications.”		Thread
Register Address: 774H, 1908	IA32_HWP_REQUEST	
See Section 17.4.4, “Managing HWP.”		Thread
7:0	Minimum Performance (R/W)	
15:8	Maximum Performance (R/W)	
23:16	Desired Performance (R/W)	
31:24	Energy/Performance Preference (R/W)	
41:32	Activity Window (R/W)	
42	Package Control (R/W)	
63:43	Reserved.	
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Section 17.4.5, “HWP Feedback.”		Thread
Register Address: D90H, 3472	IA32_BNDCFGS	
See Table 2-2.		Thread
Register Address: DAOH, 3488	IA32_XSS	
See Table 2-2.		Thread
Register Address: DBOH, 3504	IA32_PKG_HDC_CTL	
See Section 17.5.2, “Package level Enabling HDC.”		Package
Register Address: DB1H, 3505	IA32_PM_CTL1	
See Section 17.5.3, “Logical-Processor Level HDC Control.”		Thread
Register Address: DB2H, 3506	IA32_THREAD_STALL	
See Section 17.5.4.1, “IA32_THREAD_STALL.”		Thread
Register Address: DCOH, 3520	MSR_LBR_INFO_0	
Last Branch Record 0 Additional Information (R/W) One of 32 triplet of last branch record registers on the last branch record stack. This part of the stack contains flag, TSX-related and elapsed cycle information. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.9.1, “LBR Stack.”		Thread
Register Address: DC1H, 3521	MSR_LBR_INFO_1	
Last Branch Record 1 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC2H, 3522	MSR_LBR_INFO_2	
Last Branch Record 2 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC3H, 3523	MSR_LBR_INFO_3	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 3 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC4H, 3524	MSR_LBR_INFO_4	
Last Branch Record 4 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC5H, 3525	MSR_LBR_INFO_5	
Last Branch Record 5 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC6H, 3526	MSR_LBR_INFO_6	
Last Branch Record 6 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC7H, 3527	MSR_LBR_INFO_7	
Last Branch Record 7 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC8H, 3528	MSR_LBR_INFO_8	
Last Branch Record 8 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC9H, 3529	MSR_LBR_INFO_9	
Last Branch Record 9 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCAH, 3530	MSR_LBR_INFO_10	
Last Branch Record 10 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCBH, 3531	MSR_LBR_INFO_11	
Last Branch Record 11 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCCH, 3532	MSR_LBR_INFO_12	
Last Branch Record 12 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCDH, 3533	MSR_LBR_INFO_13	
Last Branch Record 13 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCEH, 3534	MSR_LBR_INFO_14	
Last Branch Record 14 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCFH, 3535	MSR_LBR_INFO_15	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 15 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDOH, 3536	MSR_LBR_INFO_16	
Last Branch Record 16 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD1H, 3537	MSR_LBR_INFO_17	
Last Branch Record 17 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD2H, 3538	MSR_LBR_INFO_18	
Last Branch Record 18 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD3H, 3539	MSR_LBR_INFO_19	
Last Branch Record 19 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD4H, 3540	MSR_LBR_INFO_20	
Last Branch Record 20 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD5H, 3541	MSR_LBR_INFO_21	
Last Branch Record 21 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD6H, 3542	MSR_LBR_INFO_22	
Last Branch Record 22 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD7H, 3543	MSR_LBR_INFO_23	
Last Branch Record 23 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD8H, 3544	MSR_LBR_INFO_24	
Last Branch Record 24 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD9H, 3545	MSR_LBR_INFO_25	
Last Branch Record 25 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDAH, 3546	MSR_LBR_INFO_26	
Last Branch Record 26 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDBH, 3547	MSR_LBR_INFO_27	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 27 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDCH, 3548	MSR_LBR_INFO_28	
Last Branch Record 28 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDDH, 3549	MSR_LBR_INFO_29	
Last Branch Record 29 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDEH, 3550	MSR_LBR_INFO_30	
Last Branch Record 30 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDFH, 3551	MSR_LBR_INFO_31	
Last Branch Record 31 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread

Table 2-40 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_4EH, 06_5EH, 06_8EH, 06_9EH, or 06_66H.

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
43:0	Current count.	
63:44	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).	
63:4	Reserved.	

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTRO	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTRO	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTRO	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTRO	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTRO	

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: E01H, 3585	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: E02H, 3586	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.1 MSRs Introduced in 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Table 2-41 lists additional MSRs for 7th generation and 8th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_8EH or 06_9EH. For an MSR listed in Table 2-41 that also appears in the model-specific tables of prior generations, Table 2-41 supersedes prior generation tables.

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 80H, 128	MSR_TRACE_HUB_STH ACPIBAR_BASE	
NPK Address Used by AET Messages (R/W)		Package
0	Lock Bit If set, this MSR cannot be re-written anymore. Lock bit has to be set in order for the AET packets to be directed to NPK MMIO.	

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
17:1	Reserved.	
63:18	ACPIBAR_BASE_ADDRESS AET target address in NPK MMIO space.	
Register Address: 1F4H, 500	MSR_PRMRR_PHYS_BASE	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MemType PRMRR BASE MemType.	
11:3	Reserved.	
45:12	Base PRMRR Base Address.	
63:46	Reserved.	
Register Address: 1F5H, 501	MSR_PRMRR_PHYS_MASK	
Processor Reserved Memory Range Register - Physical Mask Control Register (R/W)		Core
9:0	Reserved.	
10	Lock Lock bit for the PRMRR.	
11	VLD Enable bit for the PRMRR.	
45:12	Mask PRMRR MASK bits.	
63:46	Reserved.	
Register Address: 1FBH, 507	MSR_PRMRR_VALID_CONFIG	
Valid PRMRR Configurations (R/W)		Core
0	1M supported MEE size.	
4:1	Reserved.	
5	32M supported MEE size.	
6	64M supported MEE size.	
7	128M supported MEE size.	
31:8	Reserved.	
Register Address: 2F4H, 756	MSR_UNCORE_PRMRR_PHYS_BASE ¹	
(R/W)	The PRMRR range is used to protect the processor reserved memory from unauthorized reads and writes. Any IO access to this range is aborted. This register controls the location of the PRMRR range by indicating its starting address. It functions in tandem with the PRMRR mask register.	
11:0	Reserved.	Package
PAWIDTH-1:12	Range Base This field corresponds to bits PAWIDTH-1:12 of the base address memory range which is allocated to PRMRR memory.	
63:PAWIDTH	Reserved.	

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2F5H, 757	MSR_UNCORE_PRMRR_PHYS_MASK ¹	
(R/W) This register controls the size of the PRMRR range by indicating which address bits must match the PRMRR base register value.		Package
9:0	Reserved.	
10	Lock Setting this bit locks all writeable settings in this register, including itself.	
11	Range_En Indicates whether the PRMRR range is enabled and valid.	
38:12	Range_Mask This field indicates which address bits must match PRMRR base in order to qualify as an PRMRR access.	
63:39	Reserved.	
Register Address: 620H, 1568	MSR_RING_RATIO_LIMIT	
Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.		Package
6:0	MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	

NOTES:

1. This MSR is specific to 7th generation and 8th generation Intel® Core™ processors.

2.17.2 MSRs Specific to 8th Generation Intel® Core™ i3 Processors

Table 2-42 lists additional MSRs for 8th generation Intel Core i3 processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_66H. For an MSR listed in Table 2-42 that also appears in the model-specific tables of prior generations, Table 2-42 supersedes prior generation tables.

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Available only if CPUID.07H.00H:ECX[30] = 1.	
18	SGX Global Functions Enable (R/WL)	
63:21	Reserved.	
Register Address: 350H, 848	MSR_BR_DETECT_CTRL	
Branch Monitoring Global Control (R/W)		
0	EnMonitoring Global enable for branch monitoring.	
1	EnExcept Enable branch monitoring event signaling on threshold trip. The branch monitoring event handler is signaled via the existing PMI signaling mechanism as programmed from the corresponding local APIC LVT entry.	
2	EnLBRFrz Enable LBR freeze on threshold trip. This will cause the LBR frozen bit 58 to be set in IA32_PERF_GLOBAL_STATUS when a triggering condition occurs and this bit is enabled.	
3	DisableInGuest When set to '1', branch monitoring, event triggering and LBR freeze actions are disabled when operating at VMX non-root operation.	
7:4	Reserved.	
17:8	WindowSize Window size defined by WindowCntSel. Values 0 - 1023 are supported. Once the Window counter reaches the WindowSize count both the Window Counter and all Branch Monitoring Counters are cleared.	
23:18	Reserved.	
25:24	WindowCntSel Window event count select: '00 = Instructions retired. '01 = Branch instructions retired '10 = Return instructions retired. '11 = Indirect branch instructions retired.	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
26	CntAndMode When set to '1', the overall branch monitoring event triggering condition is true only if all enabled counters' threshold conditions are true. When '0', the threshold tripping condition is true if any enabled counters' threshold is true.	
63:27	Reserved.	
Register Address: 351H, 849	MSR_BR_DETECT_STATUS	
Branch Monitoring Global Status (R/W)		
0	Branch Monitoring Event Signaled When set to '1', Branch Monitoring event signaling is blocked until this bit is cleared by software.	
1	LBRsValid This status bit is set to '1' if the LBR state is considered valid for sampling by branch monitoring software.	
7:2	Reserved.	
8	CntrHit0 Branch monitoring counter #0 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.	
9	CntrHit1 Branch monitoring counter #1 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.	
15:10	Reserved. Reserved for additional branch monitoring counters threshold hit status.	
25:16	CountWindow The current value of the window counter. The count value is frozen on a valid branch monitoring triggering condition. This is a 10-bit unsigned value.	
31:26	Reserved. Reserved for future extension of CountWindow.	
39:32	Count0 The current value of counter 0 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit0 will also be set). This is an 8-bit signed value (2's complement). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
47:40	Count1 The current value of counter 1 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit1 will also be set). This is an 8-bit signed value (2's complement). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).	
63:48	Reserved.	
Register Address: 354H–355H, 852–853	MSR_BR_DETECT_COUNTER_CONFIG_i	
Branch Monitoring Detect Counter Configuration (R/W)		
0	CntrEn Enable counter.	
7:1	CntrEvSel Event select (other values #GP) '0000000 = RETs. '0000001 = RET-CALL bias. '0000010 = RET mispredicts. '0000011 = Branch (all) mispredicts. '0000100 = Indirect branch mispredicts. '0000101 = Far branch instructions.	
14:8	CntrThreshold Threshold (an unsigned value of 0 to 127 supported). The value 0 of counter threshold will result in event signaled after every instruction. #GP if threshold is < 2.	
15	MispredEventCnt Mispredict events counting behavior: '0 = Mispredict events are counted in a window. '1 = Mispredict events are counted based on a consecutive occurrence. CntrThreshold is treated as # of consecutive mispredicts. This control bit only applies to events specified by CntrEvSel that involve a prediction (0000010, 0000011, 0000100). Setting this bit for other events is ignored.	
63:16	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Package C3 Residency Counter (R/O)		Package
63:0	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.	
Register Address: 620H, 1568	MSR_RING_RATIO_LIMIT	
Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.		Package

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6:0	MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	
Register Address: 660H, 1632	MSR_CORE_C1_RESIDENCY	
Core C1 Residency Counter (R/O)		Core
63:0	Value since last reset for the Core C1 residency. Counter rate is the Max Non-Turbo frequency (same as TSC). This counter counts in case both of the core's threads are in an idle state and at least one of the core's thread residency is in a C1 state or in one of its sub states. The counter is updated only after a core C state exit. Note: Always reads 0 if core C1 is unsupported. A value of zero indicates that this processor does not support core C1 or never entered core C1 level state.	
Register Address: 662H, 1634	MSR_CORE_C3_RESIDENCY	
Core C3 Residency Counter (R/O)		Core
63:0	Will always return 0.	

Table 2-43 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_66H.

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Report the number of C-Box units with performance counters, including processor cores and processor graphics.	

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 702H, 1794	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 703H, 1795	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 708H, 1800	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 709H, 1801	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 70AH, 1802	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 70BH, 1803	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 712H, 1810	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 713H, 1811	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 718H, 1816	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 719H, 1817	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 71AH, 1818	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 71BH, 1819	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 722H, 1826	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 723H, 1827	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 728H, 1832	MSR_UNC_CBO_5_PERFEVTSELO	
Uncore C-Box 5, Counter 0 Event Select MSR		Package
Register Address: 729H, 1833	MSR_UNC_CBO_5_PERFEVTSEL1	
Uncore C-Box 5, Counter 1 Event Select MSR		Package
Register Address: 72AH, 1834	MSR_UNC_CBO_5_PERFCTR0	
Uncore C-Box 5, Performance Counter 0		Package
Register Address: 72BH, 1835	MSR_UNC_CBO_5_PERFCTR1	
Uncore C-Box 5, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_6_PERFEVTSELO	
Uncore C-Box 6, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_6_PERFEVTSEL1	
Uncore C-Box 6, Counter 1 Event Select MSR		Package
Register Address: 732H, 1842	MSR_UNC_CBO_6_PERFCTR0	
Uncore C-Box 6, Performance Counter 0		Package
Register Address: 733H, 1843	MSR_UNC_CBO_6_PERFCTR1	
Uncore C-Box 6, Performance Counter 1		Package
Register Address: 738H, 1848	MSR_UNC_CBO_7_PERFEVTSELO	
Uncore C-Box 7, Counter 0 Event Select MSR		Package
Register Address: 739H, 1849	MSR_UNC_CBO_7_PERFEVTSEL1	
Uncore C-Box 7, Counter 1 Event Select MSR		Package
Register Address: 73AH, 1850	MSR_UNC_CBO_7_PERFCTR0	
Uncore C-Box 7, Performance Counter 0		Package
Register Address: 73BH, 1851	MSR_UNC_CBO_7_PERFCTR1	
Uncore C-Box 7, Performance Counter 1		Package
Register Address: E01H, 3585	MSR_UNC_PERF_GLOBAL_CTRL	

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: E02H, 3586	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.3 MSRs Introduced in 10th Generation Intel® Core™ Processors

Table 2-44 lists additional MSRs for 10th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_7EH. For an MSR listed in Table 2-44 that also appears in the model-specific tables of prior generations, Table 2-44 supersedes prior generation tables.

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
28:0	Reserved.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, "Features to Disable Bus Locks."	
30	Reserved.	
31	Reserved.	
Register Address: 48H, 72	IA32_SPEC_CTRL	
See Table 2-2.		Core
Register Address: 49H, 73	IA32_PREDICT_CMD	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	
See Table 2-2.		Thread
Register Address: 8DH, 141	IA32_SGXLEPUBKEYHASH1	
See Table 2-2.		Thread
Register Address: 8EH, 142	IA32_SGXLEPUBKEYHASH2	
See Table 2-2.		Thread
Register Address: 8FH, 143	IA32_SGXLEPUBKEYHASH3	
See Table 2-2.		Thread
Register Address: A0H, 160	MSR_BIOS_MCU_ERRORCODE	
BIOS MCU ERRORCODE (R/O) This MSR indicates if WRMSR 0x79 failed to configure PRM memory and gives a hint to debug BIOS.		Package
15:0	Error Codes (R/O)	Package
30:16	Reserved.	
31	MCU Partial Success (R/O) When set to 1, WRMSR 0x79 skipped part of the functionality during BIOS.	Thread
Register Address: A5H, 165	MSR_FIT_BIOS_ERROR	
FIT BIOS ERROR (R/W) Report error codes for debug in case the processor failed to parse the Firmware Table in BIOS. Can also be used to log BIOS information.		Thread
7:0	Error Codes (R/W) Error codes for debug.	
15:8	Entry Type (R/W) Failed FIT entry type.	
16	FIT MCU Entry (R/W) FIT contains MCU entry.	
62:17	Reserved.	
63	LOCK (R/W) When set to 1, writes to this MSR will be skipped.	
Register Address: 10BH, 267	IA32_FLUSH_CMD	
See Table 2-2.		Thread
Register Address: 151H, 337	MSR_BIOS_DONE	
BIOS Done (R/WO)		Thread
0	BIOS Done Indication (R/WO) Set by BIOS when it finishes programming the processor and wants to lock the memory configuration from changes by software that is running on this thread. Writes to the bit will be ignored if EAX[0] is 0.	Thread

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Package BIOS Done Indication (R/O) When set to 1, all threads in the package have bit 0 of this MSR set.	Package
31:2	Reserved.	
Register Address: 1F1H, 497	MSR_CRASHLOG_CONTROL	
Write Data to a Crash Log Configuration		Thread
0	CDDIS: CrashDump_Disable If set, indicates that Crash Dump is disabled.	
63:1	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE: PRMRR BASE Memory Type.	
3	CONFIGURED: PRMRR BASE Configured.	
11:4	Reserved.	
51:12	BASE: PRMRR Base Address.	
63:52	Reserved.	
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter Register 3 (R/W) Bit definitions are the same as found in IA32_FIXED_CTR0, offset 309H. See Table 2-2.		Thread
Register Address: 329H, 809	MSR_PERF_METRICS	
Performance Metrics (R/W) Reports metrics directly. Software can check (and/or expose to its guests) the availability of PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15).		Thread
7:0	Retiring. Percent of utilized slots by uops that eventually retire (commit).	
15:8	Bad Speculation. Percent of wasted slots due to incorrect speculation, covering utilized by uops that do not retire, or recovery bubbles (unutilized slots).	
23:16	Frontend Bound. Percent of unutilized slots where front-end did not deliver a uop while back-end is ready.	
31:24	Backend Bound. Percent of unutilized slots where a uop was not delivered to back-end due to lack of back-end resources.	
63:32	Reserved.	
Register Address: 3F2H, 1010	MSR_PEBS_DATA_CFG	
PEBS Data Configuration (R/W) Provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.		Thread
0	Memory Info. Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	GPRs. Setting this bit will capture the contents of the General Purpose registers in the PEBS record.	
2	XMMs. Setting this bit will capture the contents of the XMM registers in the PEBS record.	
3	LBRs. Setting this bit will capture LBR TO, FROM, and INFO in the PEBS record.	
23:4	Reserved.	
31:24	LBR Entries. Set the field to the desired number of entries - 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, software can use LBR_entries that is multiple of 3.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W)		Core
0	L1 Scrubbing Enable When set to 1, enable L1 scrubbing.	
31:1	Reserved.	
Register Address: 657H, 1623	MSR_FAST_UNCORE_MSRS_CTL	
Fast WRMSR/RDMSR Control MSR (R/W)		Thread
3:0	FAST_ACCESS_ENABLE: Bit 0: When set to '1', provides a hint for the hardware to enable fast access mode for the IA32_HWP_REQUEST MSR. This bit is sticky and is cleaned by the hardware only during reset time. This bit is valid only if FAST_UNCORE_MSRS_CAPABILITY[0] is set. Setting this bit will cause CPUID.06H:EAX[18] to be set.	
31:4	Reserved.	
Register Address: 65EH, 1630	MSR_FAST_UNCORE_MSRS_STATUS	
Indication of Uncore MSRs, Post Write Activates		Thread
0	Indicates whether the CPU is still in the middle of writing IA32_HWP_REQUEST MSR, even after the WRMSR instruction has retired. A value of 1 indicates the last write of IA32_HWP_REQUEST is still ongoing. A value of 0 indicates the last write of IA32_HWP_REQUEST is visible outside the logical processor. Software can use the status of this bit to avoid overwriting IA32_HWP_REQUEST.	
31:1	Reserved.	
Register Address: 65FH, 1631	MSR_FAST_UNCORE_MSRS_CAPABILITY	
Fast WRMSR/RDMSR Enumeration MSR (R/O)		Thread

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3:0	MSRS_CAPABILITY: Bit 0: If set to '1', hardware supports the fast access mode for the IA32_HWP_REQUEST MSR.	
31:4	Reserved.	
Register Address: 772H, 1906	IA32_HWP_REQUEST_PKG	
See Table 2-2.		Package
Register Address: 775H, 1909	IA32_PECI_HWP_REQUEST_INFO	
See Table 2-2.		Package
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Table 2-2.		Thread

2.17.4 MSRs Introduced in the 11th Generation Intel® Core™ Processors based on Tiger Lake Microarchitecture

Table 2-45 lists additional MSRs for 11th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_8CH or 06_8DH. The MSRs listed in Table 2-44 are also supported by these processors. For an MSR listed in Table 2-45 that also appears in the model-specific tables of prior generations, Table 2-45 supersedes prior generation tables.

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: A0H, 160	MSR_BIOS_MCU_ERRORCODE	
BIOS MCU ERRORCODE (R/O)		Package
15:0	Error Codes	
31:16	Reserved.	
Register Address: A7H, 167	MSR_BIOS_DEBUG	
BIOS DEBUG (R/O) This MSR indicates if WRMSR 79H failed to configure PRM memory and gives a hint to debug BIOS.		Thread
30:0	Reserved.	
31	MCU Partial Success When set to 1, WRMSR 79H skipped part of the functionality during BIOS.	
63:32	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/O) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Package
1:0	Reserved.	
2	FUSA_SUPPORTED	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3	RSM_IN_CPL0_ONLY When set to 1, the RSM instruction is only allowed in CPL0 (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.	
4	Reserved.	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL (MSR address 33H).	
31:6	Reserved.	
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTLSS3	
IA32_VMX_PROCBASED_CTLSS3 This MSR enumerates the allowed 1-settings of the third set of processor-based controls. Specifically, VM entry allows bit X of the tertiary processor-based VM-execution controls to be 1 if and only if bit X of the MSR is set to 1. If bit X of the MSR is cleared to 0, VM entry fails if control X and the “activate tertiary controls” primary processor-based VM-execution control are both 1.		Core
0	LOADIWKEY This control determines whether executions of LOADIWKEY cause VM exits.	
63:1	Reserved.	
Register Address: 601H, 1537	MSR_VR_CURRENT_CONFIG	
Power Limit 4 (PL4) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.		Package
12:0	PL4 Value PL4 value in 0.125 A increments. This field is locked by VR_CURRENT_CONFIG[LOCK]. When the LOCK bit is set to 1b, this field becomes Read Only.	
30:13	Reserved.	
31	Lock Indication (LOCK) This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1b, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.	
62:32	Not in use.	
63	Reserved.	
Register Address: 6A0H, 1696	IA32_U_CET	
Configure User Mode CET (R/W) See Table 2-2.		
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W) See Table 2-2.		
Register Address: 6A4H, 1700	IA32_PLO_SSP	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.		
Register Address: 6A5H, 1701	IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.		
Register Address: 6A6H, 1702	IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.		
Register Address: 6A7H, 1703	IA32_PL3_SSP	
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.		
Register Address: 6A8H, 1704	IA32_INTERRUPT_SSP_TABLE_ADDR	
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
See Table 2-2.		
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
See Table 2-2.		
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
See Table 2-2.		
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
See Table 2-2.		
Register Address: 990H, 2448	IA32_COPY_STATUS ¹	
See Table 2-2.		Thread
Register Address: 991H, 2449	IA32_IWKEYBACKUP_STATUS ¹	
See Table 2-2.		Platform
Register Address: C82H, 3202	IA32_L2_QOS_CFG	
IA32_CR_L2_QOS_CFG This MSR provides software an enumeration of the parameters that L2 QoS (Intel RDT) support in any particular implementation.		Core
0	CDP_ENABLE When set to 1, it will enable the code and data prioritization for the L2 CAT/Intel RDT feature. When set to 0, code and data prioritization is disabled for L2 CAT/Intel RDT. See Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” for further details on CDP.	
31:1	Reserved.	
Register Address: D10H–D17H, 3220–3351	IA32_L2_QOS_MASK [0-7]	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_CR_L2_QOS_MASK[0-7] Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Package
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	
Register Address: D91H, 3473	IA32_COPY_LOCAL_TO_PLATFORM ¹	
See Table 2-2.		Thread
Register Address: D92H, 3474	IA32_COPY_PLATFORM_TO_LOCAL ¹	
See Table 2-2.		Thread

NOTES:

1. Further details on Key Locker and usage of this MSR can be found here:

<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.

2.17.5 MSRs Introduced in the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Table 2-46 lists additional MSRs for 12th and 13th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_97H, 06_9AH, 06_BAH, 06_B7H, or 06_BFH. Table 2-47 lists the MSRs unique to the processor P-core. Table 2-48 lists the MSRs unique to the processor E-core.

The MSRs listed in Table 2-44¹ and Table 2-45 are also supported by these processors. For an MSR listed in Table 2-46, Table 2-47, or Table 2-48 that also appears in the model-specific tables of prior generations, Table 2-46, Table 2-47, and Table 2-48 supersede prior generation tables.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor will only allow one in-progress UC store at a time.	

1. MSRs at the following addresses are not supported in the 12th and 13th generation Intel Core processor E-core: 30CH, 329H, 541H, and 657H. The MSR at address 657H is not supported in the 12th and 13th generation Intel Core processor P-core.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 11.1.2.3, "Features to Disable Bus Locks."	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, "Features to Disable Bus Locks."	
30	Reserved.	
31	Reserved.	
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTL5	
Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.		Package
Register Address: C7H, 199	IA32_PMC6	
General Performance Counter 6 (R/W) See Table 2-2.		Core
Register Address: C8H, 200	IA32_PMC7	
General Performance Counter 7 (R/W) See Table 2-2.		Core
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/O) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Package
0	STLB_QOS_SUPPORTED When set to 1, the STLB QoS feature is supported and the STLB QoS MSRs (1A8FH - 1A97H) are accessible. When set to 0, access to these MSRs will #GP.	
1	Reserved.	
2	FUSA_SUPPORTED	
3	RSM_IN_CPLO_ONLY When set to 1, the RSM instruction is only allowed in CPLO (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.	
4	UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL.	
6	SNOOP_FILTER_QOS_SUPPORTED When set to 1, the Snoop Filter Qos Mask MSRs are supported. When set to 0, access to these MSRs will #GP.	

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
7	UC_STORE_THROTTLING_SUPPORTED When set 1, UC Store throttle capability exist through MSR_MEMORY_CTRL (33H) bit 27.	
31:8	Reserved.	
Register Address: E1H, 225	IA32_UMWAIT_CONTROL	
UMWAIT Control (R/W) See Table 2-2.		
Register Address: 10AH, 266	IA32_ARCH_CAPABILITIES	
Enumeration of Architectural Features (R/O) See Table 2-2.		
Register Address: 18CH, 396	IA32_PERFEVTSEL6	
See Table 2-20.		Core
Register Address: 18DH, 397	IA32_PERFEVTSEL7	
See Table 2-20.		Core
Register Address: 195H, 405	IA32_OVERCLOCKING_STATUS	
Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR. See Table 2-2.		Package
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
Primary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL2	
See Table 2-2.		

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4C7H, 1223	IA32_A_PMC6	
Full Width Writable IA32_PMC6 Alias (R/W) See Table 2-2.		
Register Address: 4C8H, 1224	IA32_A_PMC7	
Full Width Writable IA32_PMC7 Alias (R/W) See Table 2-2.		
Register Address: 650H, 1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	
Secondary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Module
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	
Register Address: 6E1H, 1761	IA32_PKRS	
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.		
Register Address: 776H, 1910	IA32_HWP_CTL	
See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Memory Encryption Capability MSR See Table 2-2.		
Register Address: 1200H—121FH, 4608—4639	IA32_LBR_x_INFO	
Last Branch Record Entry X Info Register (R/W) See Table 2-2.		
Register Address: 14CEH, 5326	IA32_LBR_CTL	
Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.		
Register Address: 14CFH, 5327	IA32_LBR_DEPTH	
Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.		
Register Address: 1500H—151FH, 5376—5407	IA32_LBR_x_FROM_IP	
Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.		
Register Address: 1600H—161FH, 5632—5663	IA32_LBR_x_TO_IP	
Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.		
Register Address: 17D2H, 6098	IA32_THREAD_FEEDBACK_CHAR	
Thread Feedback Characteristics (R/O) See Table 2-2.		
Register Address: 17D4H, 6100	IA32_HW_FEEDBACK_THREAD_CONFIG	
Hardware Feedback Thread Configuration (R/W) See Table 2-2.		
Register Address: 17DAH, 6106	IA32_HRESET_ENABLE	
History Reset Enable (R/W) See Table 2-2.		

The MSRs listed in Table 2-47 are unique to the 12th and 13th generation Intel Core processor P-core. These MSRs are not supported on the processor E-core.

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
Prefetch Disable Bits (R/W)		
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	Reserved.	
5	AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.	
63:6	Reserved.	
Register Address: 3F7H, 1015	MSR_PEBS_FRONTEND	
FrontEnd Precise Event Condition Select (R/W) See Table 2-39.		Thread
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W)		Thread
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W) See Table 2-44.		Core
Register Address: D10H–D17H, 3220–3351	IA32_L2_QOS_MASK[0-7]	
IA32_CR_L2_QOS_MASK[0-7] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Core
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	

The MSRs listed in Table 2-48 are unique to the 12th and 13th generation Intel Core processor E-core. These MSRs are not supported on the processor P-core.

Table 2-48. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D10H–D1FH, 3220–3359	IA32_L2_QOS_MASK [0-15]	
IA32_CR_L2_QOS_MASK [0-15] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Module
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBES_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C6H, 5313–5318	MSR_RELOAD_PMCx	
Reload value for IA32_PMCx (R/W)		Core
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBES_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	

Table 2-49 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_97H, 06_9AH, 06_BAH, 06_B7H, or 06_BFH.

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).	
63:4	Reserved.	
Register Address: 2000H, 8192	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 2001H, 8193	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2002H, 8194	MSR_UNC_CBO_0_PERFCTRO	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 2003H, 8195	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 2008H, 8200	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 2009H, 8201	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 200AH, 8202	MSR_UNC_CBO_1_PERFCTRO	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 200BH, 8203	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 2010H, 8208	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 2011H, 8209	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 2012H, 8210	MSR_UNC_CBO_2_PERFCTRO	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 2013H, 8211	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 2018H, 8216	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 2019H, 8217	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 201AH, 8218	MSR_UNC_CBO_3_PERFCTRO	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 201BH, 8219	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 2020H, 8224	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 2021H, 8225	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 2022H, 8226	MSR_UNC_CBO_4_PERFCTRO	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 2023H, 8227	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 2028H, 8232	MSR_UNC_CBO_5_PERFEVTSELO	

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 5, Counter 0 Event Select MSR		Package
Register Address: 2029H, 8233	MSR_UNC_CBO_5_PERFEVTSEL1	
Uncore C-Box 5, Counter 1 Event Select MSR		Package
Register Address: 202AH, 8234	MSR_UNC_CBO_5_PERFCTR0	
Uncore C-Box 5, Performance Counter 0		Package
Register Address: 202BH, 8235	MSR_UNC_CBO_5_PERFCTR1	
Uncore C-Box 5, Performance Counter 1		Package
Register Address: 2030H, 8240	MSR_UNC_CBO_6_PERFEVTSELO	
Uncore C-Box 6, Counter 0 Event Select MSR		Package
Register Address: 2031H, 8241	MSR_UNC_CBO_6_PERFEVTSEL1	
Uncore C-Box 6, Counter 1 Event Select MSR		Package
Register Address: 2032H, 8242	MSR_UNC_CBO_6_PERFCTR0	
Uncore C-Box 6, Performance Counter 0		Package
Register Address: 2033H, 8243	MSR_UNC_CBO_6_PERFCTR1	
Uncore C-Box 6, Performance Counter 1		Package
Register Address: 2038H, 8248	MSR_UNC_CBO_7_PERFEVTSELO	
Uncore C-Box 7, Counter 0 Event Select MSR		Package
Register Address: 2039H, 8249	MSR_UNC_CBO_7_PERFEVTSEL1	
Uncore C-Box 7, Counter 1 Event Select MSR		Package
Register Address: 203AH, 8250	MSR_UNC_CBO_7_PERFCTR0	
Uncore C-Box 7, Performance Counter 0		Package
Register Address: 203BH, 8251	MSR_UNC_CBO_7_PERFCTR1	
Uncore C-Box 7, Performance Counter 1		Package
Register Address: 2040H, 8256	MSR_UNC_CBO_8_PERFEVTSELO	
Uncore C-Box 8, Counter 0 Event Select MSR		Package
Register Address: 2041H, 8257	MSR_UNC_CBO_8_PERFEVTSEL1	
Uncore C-Box 8, Counter 1 Event Select MSR		Package
Register Address: 2042H, 8258	MSR_UNC_CBO_8_PERFCTR0	
Uncore C-Box 8, Performance Counter 0		Package
Register Address: 2043H, 8259	MSR_UNC_CBO_8_PERFCTR1	
Uncore C-Box 8, Performance Counter 1		Package
Register Address: 2048H, 8264	MSR_UNC_CBO_9_PERFEVTSELO	
Uncore C-Box 9, Counter 0 Event Select MSR		Package
Register Address: 2049H, 8265	MSR_UNC_CBO_9_PERFEVTSEL1	
Uncore C-Box 9, Counter 1 Event Select MSR		Package
Register Address: 204AH, 8266	MSR_UNC_CBO_9_PERFCTR0	
Uncore C-Box 9, Performance Counter 0		Package

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 204BH, 8267	MSR_UNC_CBO_9_PERFCTR1	
Uncore C-Box 9, Performance Counter 1		Package
Register Address: 2FD0H, 12240	MSR_UNC_ARB_0_PERFEVTSELO	
Uncore Arb Unit 0, Counter 0 Event Select MSR		Package
Register Address: 2FD1H, 12241	MSR_UNC_ARB_0_PERFEVTSEL1	
Uncore Arb Unit 0, Counter 1 Event Select MSR		Package
Register Address: 2FD2H, 12242	MSR_UNC_ARB_0_PERFCTRO	
Uncore Arb Unit 0, Performance Counter 0		Package
Register Address: 2FD3H, 12243	MSR_UNC_ARB_0_PERFCTR1	
Uncore Arb Unit 0, Performance Counter 1		Package
Register Address: 2FD4H, 12244	MSR_UNC_ARB_0_PERF_STATUS	
Uncore Arb Unit 0, Performance Status		Package
Register Address: 2FD5H, 12245	MSR_UNC_ARB_0_PERF_CTRL	
Uncore Arb Unit 0, Performance Control		Package
Register Address: 2FD8H, 12248	MSR_UNC_ARB_1_PERFEVTSELO	
Uncore Arb Unit 1, Counter 0 Event Select MSR		Package
Register Address: 2FD9H, 12249	MSR_UNC_ARB_1_PERFEVTSEL1	
Uncore Arb Unit 1, Counter 1 Event Select MSR		Package
Register Address: 2FDAH, 12250	MSR_UNC_ARB_1_PERFCTRO	
Uncore Arb Unit 1, Performance Counter 0		Package
Register Address: 2FDBH, 12251	MSR_UNC_ARB_1_PERFCTR1	
Uncore Arb Unit 1, Performance Counter 1		Package
Register Address: 2FDCH, 12252	MSR_UNC_ARB_1_PERF_STATUS	
Uncore Arb Unit 1, Performance Status		Package
Register Address: 2FDDH, 12253	MSR_UNC_ARB_1_PERF_CTRL	
Uncore Arb Unit 1, Performance Control		Package
Register Address: 2FDEH, 12254	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 2FDFH, 12255	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
43:0	Current count.	
63:44	Reserved.	

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2FF0H, 12272	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4 select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 2FF2H, 12274	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.6 MSRs Introduced in the Intel® Xeon® Scalable Processor Family

The Intel® Xeon® Scalable Processor Family (CUID Signature DisplayFamily_DisplayModel value of 06_55H) supports the MSRs listed in Table 2-50.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CUID Signature DisplayFamily_DisplayModel Value of 06_55H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
18	SGX Global Functions Enable (R/WL)	
20	LMCE_ENABLED (R/WL)	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:21	Reserved.	
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved.	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) See Table 2-26.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) See Table 2-26.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.	Package
30	Programmable TJ OFFSET (R/O) See Table 2-26.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) See Table 2-26.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/W0)	
16	Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).	
24:17	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count.	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and a host-space interface is available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface is available to SMM handler.	
63:60	Reserved.	
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WCO) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WCO) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	
7	Thermal Threshold #1 Log (R/WCO) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WCO) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WCO) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Current Limit Log (R/WC0) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WC0) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) See Table 2-26.	
27:24	TCC Activation Offset (R/W) See Table 2-26.	
63:28	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
This register defines the ratio limits. RATIO[0:7] must be populated in ascending order. RATIO[i+1] must be less than or equal to RATIO[i]. Entries with RATIO[i] will be ignored. If any of the rules above are broken, the configuration is silently rejected. If the programmed ratio is:		Package
- Above the fused ratio for that core count, it will be clipped to the fuse limits (assuming !OC). - Below the min supported ratio, it will be clipped.		
7:0	RATIO_0 Defines ratio limits.	
15:8	RATIO_1 Defines ratio limits.	
23:16	RATIO_2 Defines ratio limits.	
31:24	RATIO_3 Defines ratio limits.	
39:32	RATIO_4 Defines ratio limits.	
47:40	RATIO_5 Defines ratio limits.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55:48	RATIO_6 Defines ratio limits.	
63:56	RATIO_7 Defines ratio limits.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT_CORES	
This register defines the active core ranges for each frequency point. NUMCORE[0:7] must be populated in ascending order. NUMCORE[i+1] must be greater than NUMCORE[i]. Entries with NUMCORE[i] == 0 will be ignored. The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, the configuration is silently rejected.		Package
7:0	NUMCORE_0 Defines the active core ranges for each frequency point.	
15:8	NUMCORE_1 Defines the active core ranges for each frequency point.	
23:16	NUMCORE_2 Defines the active core ranges for each frequency point.	
31:24	NUMCORE_3 Defines the active core ranges for each frequency point.	
39:32	NUMCORE_4 Defines the active core ranges for each frequency point.	
47:40	NUMCORE_5 Defines the active core ranges for each frequency point.	
55:48	NUMCORE_6 Defines the active core ranges for each frequency point.	
63:56	NUMCORE_7 Defines the active core ranges for each frequency point.	
Register Address: 280H, 640	IA32_MCO_CTL2	
See Table 2-2.		Core
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Core
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 403H, 1027	IA32_MCO_MISC	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC0 reports MC errors from the IFU module.		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.		Package
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.		Package
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.		Package
Register Address: 413H, 1043	IA32_MC4_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
63:15	Reserved.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
7	Reserved.	
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.	
31:8	Reserved.	
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
31:10	Reserved.	
51:32	CLOS (R/W)	
63: 52	Reserved.	
Register Address: C90H, 3216	IA32_L3_QOS_MASK_0	
L3 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 0 enforcement.	
63:20	Reserved.	
Register Address: C91H, 3217	IA32_L3_QOS_MASK_1	
L3 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 1 enforcement.	
63:20	Reserved.	
Register Address: C92H, 3218	IA32_L3_QOS_MASK_2	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
L3 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2 >= 2.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 2 enforcement.	
63:20	Reserved.	
Register Address: C93H, 3219	IA32_L3_QOS_MASK_3	
L3 Class Of Service Mask - CLOS 3 (R/W). If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: C94H, 3220	IA32_L3_QOS_MASK_4	
L3 Class Of Service Mask - CLOS 4 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 4.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 4 enforcement.	
63:20	Reserved.	
Register Address: C95H, 3221	IA32_L3_QOS_MASK_5	
L3 Class Of Service Mask - CLOS 5 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 5.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 5 enforcement.	
63:20	Reserved.	
Register Address: C96H, 3222	IA32_L3_QOS_MASK_6	
L3 Class Of Service Mask - CLOS 6 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 6.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 6 enforcement.	
63:20	Reserved.	
Register Address: C97H, 3223	IA32_L3_QOS_MASK_7	
L3 Class Of Service Mask - CLOS 7 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 7.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 7 enforcement.	
63:20	Reserved.	
Register Address: C98H, 3224	IA32_L3_QOS_MASK_8	
L3 Class Of Service Mask - CLOS 8 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 8.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 8 enforcement.	
63:20	Reserved.	
Register Address: C99H, 3225	IA32_L3_QOS_MASK_9	
L3 Class Of Service Mask - CLOS 9 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 9.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 9 enforcement.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: C9AH, 3226	IA32_L3_QOS_MASK_10	
L3 Class Of Service Mask - CLOS 10 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 10.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 10 enforcement.	
63:20	Reserved.	
Register Address: C9BH, 3227	IA32_L3_QOS_MASK_11	
L3 Class Of Service Mask - CLOS 11 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 11.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 11 enforcement.	
63:20	Reserved.	
Register Address: C9CH, 3228	IA32_L3_QOS_MASK_12	
L3 Class Of Service Mask - CLOS 12 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 12.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 12 enforcement.	
63:20	Reserved.	
Register Address: C9DH, 3229	IA32_L3_QOS_MASK_13	
L3 Class Of Service Mask - CLOS 13 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 13.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 13 enforcement.	
63:20	Reserved.	
Register Address: C9EH, 3230	IA32_L3_QOS_MASK_14	
L3 Class Of Service Mask - CLOS 14 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 14.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 14 enforcement.	
63:20	Reserved.	
Register Address: C9FH, 3231	IA32_L3_QOS_MASK_15	
L3 Class Of Service Mask - CLOS 15 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 15.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 15 enforcement.	
63:20	Reserved.	

2.17.7 MSRs Specific to the 3rd Generation Intel® Xeon® Scalable Processor Family Based on Ice Lake Microarchitecture

The 3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_6AH or 06_6CH) support the MSRs listed in Table 2-51.

Table 2-51. MSRs Supported by the 3rd Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_6AH or 06_6CH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 612H, 1554	MSR_PACKAGE_ENERGY_TIME_STATUS	
Package energy consumed by the entire CPU (R/W)		Package
31:0	Total amount of energy consumed since last reset.	
63:32	Total time elapsed when the energy was last updated. This is a monotonic increment counter with auto wrap back to zero after overflow. Unit is 10ns.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
Allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.		Package
14:0	DRAM_PP_PWR_LIM: Power Limit[0] for DDR domain. Units = Watts, Format = 11.3, Resolution = 0.125W, Range = 0-2047.875W.	
15	PWR_LIM_CTRL_EN: Power Limit[0] enable bit for DDR domain.	
16	Reserved.	
23:17	CTRL_TIME_WIN: Power Limit[0] time window Y value, for DDR domain. Actual time_window for RAPL is: $(1/1024 \text{ seconds}) * (1 + (x/4)) * (2^y)$	
62:24	Reserved.	
63	PP_PWR_LIM_LOCK: When set, this entire register becomes read-only. This bit will typically be set by BIOS during boot.	
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM Power Parameters (R/W)		Package
14:0	Spec DRAM Power (DRAM_TDP): The Spec power allowed for DRAM. The TDP setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
15	Reserved.	

Table 2-51. MSRs Supported by the 3rd Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_6AH or 06_6CH (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30:16	Minimal DRAM Power (DRAM_MIN_PWR): The minimal power setting allowed for DRAM. Lower values will be clamped to this value. The minimum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
31	Reserved.	
46:32	Maximal Package Power (DRAM_MAX_PWR): The maximal power setting allowed for DRAM. Higher values will be clamped to this value. The maximum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
47	Reserved.	
54:48	Maximal Time Window (DRAM_MAX_WIN): The maximal time window allowed for the DRAM. Higher values will be clamped to this value. $x = \text{PKG\_MAX\_WIN}[54:53]$ $y = \text{PKG\_MAX\_WIN}[52:48]$ The timing interval window is a floating-point number given by $1.x * \text{power}(2,y)$. The unit of measurement is defined in MSR_DRAM_POWER_INFO_UNIT[TIME_UNIT].	
62:55	Reserved.	
63	LOCK: Lock bit to lock the register.	
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
See Table 2-2.		
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
See Table 2-2.		
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
See Table 2-2.		
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
See Table 2-2.		

2.17.8 MSRs Specific to the 4th and 5th Generation Intel® Xeon® Scalable Processor Families

The 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_8FH) and the 5th generation Intel® Xeon® Scalable Processor Family based on Emerald Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_CFH) both support the MSRs listed in Section 2.17, “MSRs In the 6th—13th Generation Intel® Core™ Proces-

sors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 P-core processors, Intel® Xeon® 6 E-core processors, and Intel® Series 2 Core™ Ultra Processors,” including Table 2-52. For an MSR listed in Table 2-52 that also appears in the model-specific tables of prior generations, Table 2-52 supersedes prior generation tables.

Certain bit field positions may be related to the maximum physical address width, the value of which is expressed as “MAXPHYADDR” in Table 2-52. See Section 2.1 for an explanation of this value.

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register (R/W)		Core
27:0	Reserved.	
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
31:30	Reserved.	
Register Address: A7H, 167	MSR_BIOS_DEBUG	
BIOS DEBUG (R/O) See Table 2-45.		Thread
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTLs	
Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.		Package
Register Address: BFH, 191	IA32_PBOPT_CTRL	
IA32_PBOPT_OPT_CTRL If IA32_ARCH_CAPABILITIES[32] = 1 This MSR provides an architectural enumeration function for model-specific behavior.		
0	PREDICTION_CONTROL_BARRIER (R/W) When set to 0 (default), original Indirect Branch Predictor Barrier Target Array Return (IBPB TA RET) mitigation is enabled. When set to 1, alternative IBPB mitigation is enabled.	
63:1	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	Reserved: returns zero.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Reserved: returns zero.	
2	INTEGRITY_CAPABILITIES When set to 1, the processor supports MSR_INTEGRITY_CAPABILITIES.	
3	RSM_IN_CPLD_ONLY Indicates that RSM will only be allowed in CPLD and will #GP for all non-CPLD privilege levels.	
4	UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL.	
6	Reserved: returns zero.	
7	UC_STORE_THROTTLING_SUPPORTED Indicates that the snoop filter quality of service MSRs are supported on this core. This is based on the existence of a non-inclusive cache and the L2/MLC QoS feature supported.	
63:8	Reserved: returns zero.	
Register Address: E1H, 225	IA32_UWAIT_CONTROL	
UWAIT Control (R/W) See Table 2-2.		
Register Address: EDH, 237	MSR_RAR_CONTROL	
RAR Control (R/W)		Thread
29:0	Reserved.	
30	IGNORE_IF Allow RAR servicing at the RLP regardless of the value of RFLAGS.IF.	
31	ENABLE RAR events are recognized. When RAR is not enabled, RARs are dropped.	
63:32	Reserved.	
Register Address: EEH, 238	MSR_RAR_ACTION_VECTOR_BASE	
Pointer to RAR Action Vector (R/W)		Thread
5:0	Reserved.	
MAXPHYADDR-1:6	VECTOR_PHYSICAL_ADDRESS Pointer to the physical address of the 64B aligned RAR action vector.	
63:MAXPHYADDR	Reserved.	
Register Address: EFH, 239	MSR_RAR_PAYLOAD_TABLE_BASE	
Pointer to Base of RAR Payload Table (R/W)		Thread
11:0	Reserved.	
MAXPHYADDR-1:12	TABLE_PHYSICAL_ADDRESS Pointer to the base physical address of the 4K aligned RAR payload table.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:MAXPHYADDR	Reserved.	
Register Address: F0H, 240	MSR_RAR_INFO	
Read Only RAR Information (R/O)		Thread
31:0	Supported payload type bitmap. A value of 1 in bit position [i] indicates that payload type [i] is supported.	
37:32	Table Max Index Maximum supported payload table index.	
63:38	Always zero.	
Register Address: 105H, 261	MSR_CORE_BIST	
Core BIST (R/W) Controls Array BIST activation and status checking as part of FUSA.		Core
31:0	BIST_ARRAY Bitmap indicating which arrays to run BIST on (WRITE). Bitmap indicating which arrays were not processed, i.e., completion mask (READ).	
39:32	BANK Array bank of the [least significant set bit] array indicated in EAX to start BIST(WRITE). Array bank interrupted or failed (READ).	
47:40	DWORD Array dword of the [least significant set bit] array indicated in EAX to start BIST (WRITE). Array dword interrupted or failed (READ).	
62:48	Reserved.	
63	CTRL_RESULT Indicates whether WRMSR should signal Machine-Check upon BIST-error (WRITE). BIST result PASS(0)/FAIL(1) of the (least significant set bit) array indicated in EAX (READ).	
Register Address: 10AH, 266	IA32_ARCH_CAPABILITIES	
Enumeration of Architectural Features (R/O) See Table 2-2.		
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
Prefetch Disable Bits (R/W)		
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	Reserved.	
5	AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.	
63:6	Reserved.	
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
Primary Maximum Turbo Ratio Limit (R/W) See Table 2-46.		Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT_CORES	
See Table 2-50.		Package
Register Address: 1C4H, 452	IA32_XFD	
Extended Feature Detect (R/W) See Table 2-2.		
Register Address: 1C5H, 453	IA32_XFD_ERR	
XFD Error Code (R/W) See Table 2-2.		
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	
COPY_SCAN_HASHES (W)		Die
63:0	SCAN_HASH_ADDR Contains the linear address of the SCAN Test HASH Binary loaded into memory.	
Register Address: 2C3H, 707	MSR_SCAN_HASHES_STATUS	
SCAN_HASHES_STATUS (R/O)		
15:0	CHUNK_SIZE Chunk size of the test in KB.	Die
23:16	NUM_CHUNKS Total number of chunks.	Die
31:24	Reserved: all zeros.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE The error-code refers to the LP that runs WRMSR(2C2H). 0x0: No error reported. 0x1: Attempt to copy scan-hashes when copy already in progress. 0x2: Secure Memory not set up correctly. 0x3: Scan-image header Image_info.ProgramID doesn't match RDMSR(2D9H)[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4: Reserved 0x5: Integrity check failed. 0x6: Re-install of scan test image attempted when current scan test image is in use by other LPs.	Thread
50:40	Reserved: set to all zeros.	
62:51	MAX_CORE_LIMIT Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	Die
63	Valid Valid bit is set when COPY_SCAN_HASHES has completed successfully.	Die
Register Address: 2C4H, 708	MSR_AUTHENTICATE_AND_COPY_CHUNK	
AUTHENTICATE_AND_COPY_CHUNK (W)		Die
7:0	CHUNK_INDEX Chunk Index, should be less than the total number of chunks defined by NUM_CHUNKS (MSR_SCAN_HASHES_STATUS[23:16]).	
63:8	CHUNK_ADDR Bits 63:8 of 256B aligned Linear address of scan chunk in memory.	
Register Address: 2C5H, 709	MSR_CHUNKS_AUTHENTICATION_STATUS	
CHUNKS_AUTHENTICATION_STATUS (R/O)		
7:0	VALID_CHUNKS Total number of Valid (authenticated) chunks.	Die
15:8	TOTAL_CHUNKS Total number of chunks.	Die
31:16	Reserved: all zeros.	
39:32	ERROR_CODE The error code refers to the LP that runs WRMSR(2C4H). 0x0: No error reported. 0x1: Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. 0x2: CHUNK authentication error. HASH of chunk did not match expected value.	Thread
63:40	Reserved: set to all zeros.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
ACTIVATE_SCAN (W)		Thread
7:0	CHUNK_START_INDEX Indicates chunk index to start from.	
15:8	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive).	
31:16	Reserved: all zeros.	
62:32	THREAD_WAIT_DELAY TSC based delay to allow threads to rendezvous.	
63	SIGNAL_MCE If 1, then on scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. If 0, then no logging/no signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
SCAN_STATUS (R/O)		
7:0	CHUNK_NUM SCAN Chunk that was reached.	Core
15:8	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive). Maps to same field in WRMSR(ACTIVATE_SCAN).	Core
31:16	Reserved: return all zeros.	
39:32	ERROR_CODE 0x0: No error. 0x1: SCAN operation did not start. Other thread did not join in time. 0x2: SCAN operation did not start. Interrupt occurred prior to threads rendezvous. 0x3: SCAN operation did not start. Power Management conditions are inadequate to run Intel In-field Scan. 0x4: SCAN operation did not start. Non-valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5: SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6: SCAN operation did not start. Core not capable of performing SCAN currently. 0x8: SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Intel In-field Scan concurrently. MAX_CORE_LIMIT exceeded. 0x9: Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed.	Thread
61:40	Reserved: return all zeros.	
62	SCAN_CONTROL_ERROR Scan-System-Controller malfunction.	Core

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63	SCAN_SIGNATURE_ERROR Core failed SCAN-SIGNATURE checking for this chunk.	Core
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	
SCAN_MODULE_ID (R/O)		Module
31:0	RevID of the currently installed scan test image. Maps to Revision field in external header (offset 4).	
63:32	Reserved: return all zeros.	
Register Address: 2C9H, 713	MSR_LAST_SAF_WP	
LAST_SAF_WP (R/O)		Core
31:0	LAST_WP Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. Available only if enumerated in MSR_INTEGRITY_CAPABILITIES[10:9].	
63:32	Reserved: return all zeros.	
Register Address: 2D9H, 729	MSR_INTEGRITY_CAPABILITIES	
INTEGRITY_CAPABILITIES (R/O)		Module
0	STARTUP_SCAN_BIST When set, supports Intel In-field Scan.	
3:1	Reserved: return all zeros.	
4	PERIODIC_SCAN_BIST When set, supports Intel In-field Scan.	
23:5	Reserved: return all zeros.	
31:24	ID of the scan programs supported for this part. WRMSR(2C2H) verifies this value against the corresponding value in the scan-image header, i.e., Image_info.	
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 413H, 1043	IA32_MC4_MISC	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTL3	
Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Table 2-2.		
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL2	
Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Table 2-2.		
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W) See Table 2-47.		Thread
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 61 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 64DH, 1613	MSR_PLATFORM_ENERGY_STATUS	
Platform Energy Status (R/O)		Package
31:0	TOTAL_ENERGY_CONSUMED Total energy consumption in J (32.0), in 10nsec units.	
63:32	TIME_STAMP Time stamp (U32.0).	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W-L)		Package
16:0	POWER_LIMIT_1 The average power limit value that the platform must not exceed over a time window as specified by the Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPL_POWER_UNIT.	
17	POWER_LIMIT_1_EN When set, the processor can apply control policies such that the platform average power does not exceed the Power_Limit_1 value over an exponential weighted moving average of the time window.	
18	CRITICAL_POWER_CLAMP_1 When set, the processor can go below the OS-requested P States to maintain the power below the specified Power_Limit_1 value.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
25:19	POWER_LIMIT_1_TIME This indicates the time window over which the Power_Limit_1 value should be maintained. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:26	Reserved.	
48:32	POWER_LIMIT_2 This is the Duration Power limit value that the platform must not exceed. The unit is specified in MSR_RAPL_POWER_UNIT.	
49	Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.	
50	Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.	
57:51	POWER_LIMIT_2_TIME This indicates the time window over which the Power_Limit_2 value should be maintained. This field has the same format as the POWER_LIMIT_1_TIME field.	
62:58	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 665H, 1637	MSR_PLATFORM_POWER_INFO	
Platform Power Information (R/W)		Package
16:0	MAX_PPL1 Maximum PP L1 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
31:17	MIN_PPL1 Minimum PP L1 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
48:32	MAX_PPL2 Maximum PP L2 value. The unit is specified in MSR_RAPL_POWER_UNIT.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:49	MAX_TW Maximum time window. The unit is specified in MSR_RAPL_POWER_UNIT.	
62:56	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 666H, 1638	MSR_PLATFORM_RAPL_SOCKET_PERF_STATUS	
Platform RAPL Socket Performance Status (R/O)		Package
31:0	Count of limited performance due to platform RAPL limit.	
Register Address: 6A0H, 1696	IA32_U_CET	
Configure User Mode CET (R/W) See Table 2-2.		
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W) See Table 2-2.		
Register Address: 6A4H, 1700	IA32_PL0_SSP	
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.		
Register Address: 6A5H, 1701	IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.		
Register Address: 6A6H, 1702	IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.		
Register Address: 6A7H, 1703	IA32_PL3_SSP	
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.		
Register Address: 6A8H, 1704	IA32_INTERRUPT_SSP_TABLE_ADDR	
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.		
Register Address: 6E1H, 1761	IA32_PKRS	
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.		
Register Address: 776H, 1910	IA32_HWP_CTL	
See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
Memory Encryption Capability MSR See Table 2-2.		

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 985H, 2437	IA32_UINTR_RR	
User Interrupt Request Register (R/W) See Table 2-2.		
Register Address: 986H, 2438	IA32_UINTR_HANDLER	
User Interrupt Handler Address (R/W) See Table 2-2.		
Register Address: 987H, 2439	IA32_UINTR_STACKADJUST	
User Interrupt Stack Adjustment (R/W) See Table 2-2.		
Register Address: 988H, 2440	IA32_UINTR_MISC	
User-Interrupt Target-Table Size and Notification Vector (R/W) See Table 2-2.		
Register Address: 989H, 2441	IA32_UINTR_PD	
User Interrupt PID Address (R/W) See Table 2-2.		
Register Address: 98AH, 2442	IA32_UINTR_TT	
User-Interrupt Target Table (R/W) See Table 2-2.		
Register Address: C70H, 3184	MSR_B1_PMON_EVNT_SELO	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C71H, 3185	MSR_B1_PMON_CTR0	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C72H, 3186	MSR_B1_PMON_EVNT_SEL1	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C73H, 3187	MSR_B1_PMON_CTR1	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C74H, 3188	MSR_B1_PMON_EVNT_SEL2	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C75H, 3189	MSR_B1_PMON_CTR2	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C76H, 3190	MSR_B1_PMON_EVNT_SEL3	
Uncore B-box 1vPerfMon event select MSR.		Package
Register Address: C77H, 3191	MSR_B1_PMON_CTR3	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C82H, 3122	MSR_W_PMON_BOX_OVF_CTRL	
Uncore W-box PerfMon local box overflow control MSR.		Package
Register Address: C8FH, 3215	IA32_PQR_ASSOC	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		
Register Address: C90H—C9EH, 3216—3230	IA32_L3_QOS_MASK_0 through IA32_L3_QOS_MASK_14	
See Table 2-50.		Package
Register Address: D10H—D17H, 3344—3351	IA32_L2_QOS_MASK_[0-7]	
IA32_CR_L2_QOS_MASK_[0-7] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. See Table 2-2.		Core
Register Address: D93H, 3475	IA32_PASID	
See Table 2-2.		
Register Address: 1200H—121FH, 4608—4639	IA32_LBR_x_INFO	
Last Branch Record Entry X Info Register (R/W) See Table 2-2.		
Register Address: 1406H, 5126	IA32_MCU_CONTROL	
See Table 2-2.		
Register Address: 14CEH, 5326	IA32_LBR_CTL	
Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.		
Register Address: 14CFH, 5327	IA32_LBR_DEPTH	
Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.		
Register Address: 1500H—151FH, 5376—5407	IA32_LBR_x_FROM_IP	
Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.		
Register Address: 1600H—161FH, 5632—5663	IA32_LBR_x_TO_IP	
Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.		

2.17.9 MSRs Introduced in the Intel® Core™ Ultra 7 Processor Supporting Performance Hybrid Architecture

Table 2-53 lists additional MSRs for the Intel Core Ultra 7 processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_AA. Table 2-54 lists the MSRs unique to the processor P-core. Table 2-55 lists the MSRs unique to the processor E-core.

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor will only allow one in-progress UC store at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
63:30	Reserved.	
Register Address: 7AH, 122	IA32_FEATURE_ACTIVATION	
Feature Activation (R/W) Implements Feature Activation command. WRMSR to this address activates all ‘activatable’ features on this thread. See Table 2-2.		
Register Address: 80H, 128	MSR_TRACE_HUB_STH ACPIBAR_BASE	
MSR_TRACE_HUB_STH ACPIBAR_BASE (R/W) This register is used by BIOS to program Trace Hub STH base address that will be used by AET messages.		Thread
0	LOCK Lock bit. If set, this MSR cannot be re-written anymore. The lock bit has to be set in order for the AET packets to be directed to Trace Hub MMIO.	
17:1	Reserved.	
45:18	ADDRESS AET target address in Trace Hub MMIO space.	
63:46	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration (R/W)		Core

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3:0	PKG_C_STATE_LIMIT Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings may be supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
7:4	MAX_CORE_C_STATE Possible values are: 0000—reserved; 0001—C1; 0010—C3, 0011—C6.	
9:8	Reserved.	
10	IO_MWAIT_REDIRECTION_ENABLE When set, will map IO_read instructions sent to IO registers PMG_IO_BASE_ADDR.PMB0+0/1/2 to MWAIT(C2,3,4) instructions; applies to deepc4 too.	
14:11	Reserved.	
15	CFG_LOCK When set, locks bits 15:0 of this register for further writes, until the next reset occurs.	
24:16	Reserved.	
25	C3_STATE_AUTO_DEMOTION_ENABLE When set, processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1_STATE_AUTO_DEMOTION_ENABLE When set, processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	ENABLE_C3_UNDEMOTION Enable Un-Demotion from Demoted C3.	
28	ENABLE_C1_UNDEMOTION Enable Un-Demotion from Demoted C1.	
29	ENABLE_PKGC_AUTODEMOTION Enable Package C-State Auto-Demotion. It enables use of the history of past package C-state depth and residence, as a factor in determining C-State depth.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30	ENABLE_PKGC_UNDEMOTION Enable Package C-State Un-Demotion. It enables considering cases where demotion was the incorrect decision in determining C-State depth.	
31	TIMED_MWAIT_ENABLE When set, enables Timed MWAIT feature. MWAIT would #GP on attempts to do setup MWAIT timer if this bit is not set.	
63:32	Reserved.	
Register Address: E4H, 228	MSR_IO_CAPTURE_BASE	
IO Capture Base (R/W) Power Management IO Redirection in C-state. See http://biosbits.org .		Core
15:0	LVL_2_BASE_ADDRESS Specifies the base address visible to software for IO redirection. If MSR_PKG_CST_CONFIG_CONTROL.IO_MWAIT_REDIRECTION_ENABLE, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	CST_RANGE Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL.IO_MWAIT_REDIRECTION_ENABLE: 000b—C3 is the max C-State to include. 001b—C6 is the max C-State to include. 010b—C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Feature Configuration (R/W)		Core
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
1	AESNI_DISABLE This bit disables Advanced Encryption Standard feature on this processor core. To disable AES, BIOS will write '11 to this MSR on every core.	
63:2	Reserved.	
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Feature Enable (R/W) Miscellaneous enables for thread specific features.		Thread
0	CPUID_GP_ON_CPL_GT_0 Causes CPUID to #GP if CPL greater than 0 and not in SMM.	
63:1	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target (R/W) Legacy register holding temperature related constants for Platform use.		Package
6:0	TCC Offset Time Window Describes the RATL averaging time window.	
7	TCC Offset Clamping Bit When enabled will allow RATL throttling below P1.	
15:8	Temperature Control Offset Fan Temperature Target Offset (a.k.a. T-Control) indicates the relative offset from the Thermal Monitor Trip Temperature at which fans should be engaged.	
23:16	TCC Activation Temperature The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
30:24	TCC Activation Offset Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only if MSR_PLATFORM_INFO[30] is set.	
31	LOCKED When set, this entire register becomes read-only.	
63:2	Reserved.	
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
PREFETCH Control (R/W) Prefetch disable bits.		Thread
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	DCU_NEXT_PAGE_PREFETCH_DISABLE If 1, disables Next Page prefetcher.	
5	AMP_PREFETCH_DISABLE If 1, disables L2 Adaptive Multipath Probability (AMP) prefetcher.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6	LLC_PAGE_PREFETCH_DISABLE If 1, disables the LLC Page prefetcher.	
7	AOP_PREFETCH_DISABLE	
8	STREAM_PREFETCH_CODE_FETCH_DISABLE	
63:9	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
OFFCORE_RSP_0 (R/W) Offcore Response Event Select Register		Thread
0	TRUE_DEMAND_CACHE_LOAD Demand Data Rd = DCU reads (includes partials) that is not tagged homeless.	
1	DEMAND_RFO Demand Instruction fetch = IFU Fetches. ItoM or RFO that is not tagged homeless.	
2	DEMAND_CODE_READ Demand Instruction fetch = IFU Fetches. CRd or CRd_UC.	
3	CORE_MODIFIED_WRITEBACK WBMtoI or WBMtoE.	
4	HW_PREFETCH_MLC_LOAD L2 prefetcher requests triggered by reads from MEC (except those triggered by I-side).	
5	HW_PREFETCH_MLC_RFO L2 prefetcher requests triggered by RFOs.	
6	HW_PREFETCH_MLC_CODE L2 prefetcher requests triggered by I-side requests.	
7	HW_PREFETCH_LLC_LOAD LLC prefetch requests triggered by DRd.	
8	HW_PREFETCH_LLC_RFO LLC prefetch requests triggered by RFO.	
9	HW_PREFETCH_LLC_CODE LLC prefetch requests triggered by CRd.	
10	L1_HWPREFETCH Covers Hardware PFRFO, PFNEAR, PFMED, PFFAR, PFHW, PFNTA, PFNPP, PFIPP including the homeless versions.	
11	ALL_STREAMING_STORE Write Combining. WCIL or WCILF.	
12	CORE_NON_MODIFIED_WB WBEFtoI or WBEFtoE.	
13	LLC_PREFETCH LLC prefetch of load/code/RFO.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
14	L1_SWPREFETCH Covers Software PFRFO, PFNEAR, PFMED, PFFAR, PFHW, PFNTA, PFNPP, PFIPP including the homeless versions.	
15	OTHER Includes CLFlush, CLFlushOPT, CLDemote, CLWB, Enqueue SetMonitor, PortIn, IntA, Lock, SplitLock, Unlock, SpCyc, ClrMonitor, PortOut, IntPriUp, IntLog, IntPhy, EOI, RdCurr, WbStol, LLCWBInv, LLCInv, NOP, PCOMMIT.	
16	ANY_RESP Match on any response.	
17	SUPPLIER_NONE No Supplier Details. DATA_PRE [6:3] = 0.	
18	LLC_HIT_M_STATE LLC/L3, M-state, DATA_PRE [6:3] = 2.	
19	LLC_HIT_E_STATE LLC/L3, E-state, DATA_PRE [6:3] = 4.	
20	LLC_HIT_S_STATE LLC/L3, S-state, DATA_PRE [6:3] = 6.	
21	LLC_HIT_F_STATE LLC/L3, F-state, DATA_PRE [6:3] = 8.	
22	FAR_MEM_LOCAL Far Memory, Local, DATA_PRE [6:3] = 1.	
23	FAR_MEM_REMOTE_0_HOP Far Memory, Remote 0-hop, DATA_PRE [6:3] = 3.	
24	FAR_MEM_REMOTE_1_HOP Far Memory, Remote 1-hop, DATA_PRE [6:3] = 5.	
25	FAR_MEM_REMOTE_2_PLUS_HOP Far Memory, Rem 2+ hop, DATA_PRE [6:3] = 7.	
26	NEAR_MEM_MISS_LOCAL_NODE LLC Miss Local Node. Near Memory, Local DATA_PRE [6:3] = E.	
27	NEAR_MEM_REMOTE_0_HOP Near Memory, Remote 0-hop, DATA_PRE [6:3] = B	
28	NEAR_MEM_REMOTE_1_HOP Near Memory, Remote 1-hop, DATA_PRE [6:3] = D.	
29	NEAR_MEM_REMOTE_2_PLUS_HOP Near Memory, Remote 2+ hop, DATA_PRE [6:3] = F.	
30	SPL_HIT Snoop Info: SPL-hit, DATA_PRE [2:0] = 6.	
31	SNOOP_NONE No details as to Snoop-related info. Snoop Info: None, DATA_PRE [2:0] = 0.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
32	NOT_NEEDED No snoop was needed to satisfy the request. Snoop Info: Not needed, DATA_PRE [2:0] = 1.	
33	MISS No snoop was needed to satisfy the request. Snoop Info: Miss, DATA_PRE [2:0] = 2.	
34	HIT_NO_FWD A snoop was needed and it Hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. Snoop Info: Hit No Fwd, DATA_PRE [2:0] = 3.	
35	HIT_EF_WITH_FWD A snoop was needed and data was Forwarded from a remote socket. Snoop Info: Hit EF w/Fwd, DATA_PRE [2:0] = 4.	
36	HITM A snoop was needed and it HitMed in local or remote cache. HitM denotes a cache-line was modified before snoop effect. Snoop Info: HitM, DATA_PRE [2:0] = 5.	
37	NON_DRAM Target was non-DRAM system address. Snoop Info: HitM, DATA_PRE [2:0] = 5.	
38	GO_ERR GO-ERR, RspData[3:0] = 0100.	
39	GO_NO_GO GO-NoGO, RspData[3:0] = 0111.	
40	INPKG_MEM_LOCAL In-package Memory, Local, DATA_PRE [6:3] = 9.	
41	INPKG_MEM_NONLOCAL In-package Memory, Non-Local, DATA_PRE [6:3] = C.	
43:42	Reserved.	
44	UC_LOAD PRd or UCRdF.	
45	UC_STORE WiL.	
46	PARTIAL_STREAMING_STORES WCiL.	
47	FULL_STREAMING_STORES WCiLF.	
48	L1_MODIFIED_WB EVICTIION EXTTYPE from MEC.	
49	L2_MODIFIED_WB WBMtol or WBMtoE.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
50	PSMI MemPushWr_NS (PSMI only).	
51	ITOM ItoM.	
63:52	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
OFFCORE_RSP_1 (R/W) Offcore Response Event Select Register. See MSR_OFFCORE_RSP_0 (at 1A6H).		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control (R/W) Various model-specific features enumeration. See http://biosbits.org .		Package
0	Reserved.	
1	ENABLE_HWP_VOTING_RIGHT When set (1), The CPU will take into account thread HWP requests for threads that have voting rights only (ignores thread requests if they do not have voting rights). When reset(0), The CPU will take into account all thread HWP requests, even for threads that don't have voting rights. Setting this bit will cause the HWP Base feature bit to be reported in CPUID as present; clearing will cause it to be reported as non-present.	
5:2	Reserved.	
6	ENABLE_HWP Setting this bit will cause the HWP Base feature bit to report as present in CPUID; clearing this bit will cause CPUID to report the feature as non-present.	
7	ENABLE_HWP_INTERRUPT Setting this bit will cause the HWP Interrupt feature CPUID.06H:EAX[8] bit to report as present; clearing will report as non-present.	
8	ENABLE_OUT_OF_BAND_AUTONOMOUS Setting this bit will cause the HWP Autonomous feature bit to report as present; clearing will report as non-present.	
11:9	Reserved.	
12	ENABLE_HWP_EPP Enable HWP EPP. Setting this bit (1) will cause the HWP CPUID.06H:EAX[10] Energy Performance Preference bit to report as present (1); clearing will report as non-present (0).	
13	LOCK Setting this bit will prevent the BIOS specific bits from changing until the next reset. i.e., only Bits [0,22] which are meant for OS use can be changed once the LOCK bit is set.	
63:14	Reserved.	
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Primary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 1F1H, 497	MSR_CRASHLOG_CONTROL	
Crash Log Control (R/W) Write data to a Crash Log configuration.		Thread
0	CDDIS CrashDump_Disable: If set, indicates that Crash Dump is disabled.	
1	EN_GPRS Collect GPRs on a crash dump. Only meaningful when CDDIS is zero.	
2	EN_GPRS_IN_SMM Collect GPRs in SMM on a crash dump. Only meaningful when CDDIS is zero. EN_GPRS will override this control,	
3	TRIPLE_FAULT_SHUTDOWN Collect a crash log on a triple fault shutdown. Only meaningful when CDDIS is zero.	
63:4	Reserved.	
Register Address: 1F5H, 501	MSR_PRMRR_PHYS_MASK	
Processor Reserved Memory Range Register - Physical Mask (R/W)		Core
9:0	Reserved.	
10	LOCK Once set, this bit prevents software from modifying the PRMRR.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
11	VALID This bit serves as the enable for the PRMRR; the PRMRR must be LOCKed before it can be enabled.	
19:12	Reserved.	
45:20	MASK PRMRR Address Mask.	
63:46	Reserved.	
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register (R/W) See http://biosbits.org .		Package
0	ENABLE_BIDIR_PROCHOT Used to enable or disable the response to PROCHOT# input. When set/enabled, the platform can force the CPU to throttle to a lower power condition such as Pn/Pm by asserting proshot#. When clear/disabled (default), the CPU ignores the status of the proshot input signal.	
1	C1E_ENABLE When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	
2	SAPM_IMC_C2_POLICY This bit determines if self-refresh activation is allowed when entering Package C2 State. If it is set to 0b, PCODE will keep the FORCE_SR_OFF bit asserted in Package C2 State and allow its negation according to the defined latency negotiations with the PCH and Display Engine in Package C3 and deeper states. Otherwise, self-refresh is allowed in Package C2 State.	
3	FAST_BRK_SNP_EN This bit controls the VID swing rate for the OTHER_SNP_WAKE events that are detected by the iMPH. This is the event that is detected by the iMPH when a non-DMI snoopable request is observed while UCLK domain is not functional. 0b: Use slow VID swing rate. 1b: Use fast VID swing rate.	
17:4	Reserved.	
18	PWR_PERF_PLTFRM_OVR Power performance platform override.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
19	EE_TURBO_DISABLE Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.	
20	RTH_DISABLE Setting this bit disables the Race to Halt optimization and avoids this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization.	
21	DIS_PROCHOT_OUT Prochot output disable.	
22	PROCHOT_RESPONSE Prochhot configurable response enable.	
23	VR_THERM_ALERT_DISABLE_LOCK When set to 1, locks PROCHOT related bits of this MSR. Once set, a reset is required to clear this bit.	
24	VR_THERM_ALERT_DISABLE When set to 1, disables the VR_THERMAL_ALERT signaling.	
25	DISABLE_RING_EE Disable Ring EE.	
26	DISABLE_SA_OPTIMIZATION Disable SA optimization.	
27	DISABLE_OOK Disable OOK.	
28	DISABLE_AUTONOMOUS Disable HWP autonomous mode.	
29	Reserved.	
30	CSTATE_PREWAKE_DISABLE C-state pre-wake disable.	
63:31	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
19:4	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
45:20	BASE PRMRR base address.	
63:46	Reserved.	
Register Address: 474H, 1140	IA32_MC29_CTL	
MC29_CTL. See Table 2-2.		Package
Register Address: 475H, 1141	IA32_MC29_STATUS	
MC29_STATUS. See Table 2-2.		Package
Register Address: 476H, 1142	IA32_MC29_ADDR	
MC29_ADDR. See Table 2-2.		Package
Register Address: 477H, 1143	IA32_MC29_MISC	
MC29_MISC. See Table 2-2.		Package
Register Address: 478H, 1144	IA32_MC30_CTL	
MC30_CTL. See Table 2-2.		Package
Register Address: 479H, 1145	IA32_MC30_STATUS	
MC30_STATUS. See Table 2-2.		Package
Register Address: 47AH, 1146	IA32_MC30_ADDR	
MC30_ADDR. See Table 2-2.		Package
Register Address: 47BH, 1147	IA32_MC30_MISC	
MC30_MISC. See Table 2-2.		Package
Register Address: 47CH, 1148	IA32_MC31_CTL	
MC31_CTL. See Table 2-2.		Package
Register Address: 47DH, 1149	IA32_MC31_STATUS	
MC31_STATUS. See Table 2-2.		Package
Register Address: 47EH, 1150	IA32_MC31_ADDR	
MC31_ADDR. See Table 2-2.		Package
Register Address: 47FH, 1151	IA32_MC31_MISC	
MC31_MISC. See Table 2-2.		Package
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (R/W) Reports SMM capability enhancement.		Package
0	LOCK When set, locks this register from further changes.	
1	SMM_CPU_SAVE_EN If 0, SMI/RSM will save/restore state in SMRAM If 1, SMI/RSM will save/restore state from SRAM.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	SMM_CODE_CHK_EN When clear (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set, any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 601H, 1537	MSR_VR_CURRENT_CONFIG	
Power Limit 4 (PL4) (R/W) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.		Package
15:0	CURRENT_LIMIT PL4 Value in 0.125 A increments. This field is locked by MSR_VR_CURRENT_CONFIG.LOCK. When the LOCK bit is set to 1, this field becomes Read Only.	
30:16	Reserved.	
31	LOCK This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.	
63:32	Reserved.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Min/Max Ratio Limits for Uncore LLC and Ring.		Package
6:0	MAX_CLR_RATIO Maximum allowed ratio for the Ring and Last Level Cache (LLC).	
7	Reserved.	
14:8	MIN_CLR_RATIO Minimum allowed ratio for the Ring and Last Level Cache (LLC).	
63:15	Reserved.	
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
MSR_PPO_POWER_LIMIT (R/W) PPO RAPL power unit control.		Package
14:0	IA_PP_PWR_LIM This is the power limitation on the IA cores power plane. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSR[PWR_UNIT].	
15	PWR_LIM_CTRL_EN This bit must be set in order to limit the power of the IA cores power plane. 0b: IA cores power plane power limitation is disabled. 1b: IA cores power plane power limitation is enabled.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
16	PP_CLAMP_LIM Power Plane Clamping limitation; allow going below P1. 0b: PBM is limited between P1 and P0. 1b: PBM can go below P1.	
23:17	CTRL_TIME_WIN x = CTRL_TIME_WIN[23:22] y = CTRL_TIME_WIN[21:17] The timing interval window is Floating Point number given by $1.x * \text{power}(2,y)$. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSR[TIME_UNIT]. The maximal time window is bounded by PACKAGE_POWER_SKU_MSR[PKG_MAX_WIN]. The minimum time window is 1 unit of measurement (as defined above).	
30:24	Reserved.	
31	PP_PWR_LIM_LOCK When set, all settings in this register are locked and are treated as Read Only.	
63:32	Reserved.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
Core Performance Limit Reasons Indicator of Frequency Clipping in Processor Cores. (Frequency refers to processor core frequency.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	RSR_LIMIT (R/O) Residency State Regulation Status. When set, frequency is reduced below the operating system request due to residency state regulation limit.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
7	VR_TDC (R/O) VR Therm Design Current Status. When set, frequency is reduced below the operating system request due to VR thermal design current limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced below the operating system request due to electrical or other constraints.	
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.	
12	MAX_TURBO_LIMIT (R/O) Max Turbo Limit Status. When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
13	TURBO_ATTEN (R/O) Turbo Transition Attenuation Status. When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	RSR_LIMIT_LOG (R/W) Residency State Regulation Log. When set, indicates that the Residency State Regulation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	RATL_LOG (R/W) Running average thermal limit Log, RW, When set by PCODE indicates that Running average thermal limit has cause IA frequency clipping. Software should write to this bit to clear the status in this bit.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR TDC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	MAX_TURBO_LIMIT_LOG (R/W) Max Turbo Limit Log. When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	TURBO_ATTEN_LOG (R/W) Turbo Transition Attenuation Log. When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 650H, 1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	
Secondary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W) Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor. The processor implements an exponential-weighted algorithm in the placement of the time windows.		Package
14:0	POWER_LIMIT_1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (a.k.a TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
15	POWER_LIMIT_1_EN When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit 1 over the time window specified by Power Limit 1 Time Window.	
16	CRITICAL_POWER_CLAMP_1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit 1 value.	
23:17	POWER_LIMIT_1_TIME Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, The unit is specified in MSR_RAPLPOWER_UNIT[Time Unit]	
31:24	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
46:32	POWER_LIMIT_2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit 1).	
47	POWER_LIMIT_2_EN When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit 2 over the Short Duration time window.	
48	CRITICAL_POWER_CLAMP_2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit 2 value.	
62:49	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 6BOH, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
MSR_GRAPHICS_PERF_LIMIT_REASONS Indicator of Frequency Clipping in the Processor Graphics. (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced due to running average thermal limit.	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR_TDC (R/O) VR Thermal Design Current Status. When set, frequency is reduced due to VR TDC limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
12	INEFFICIENT_OPERATION (R/O) Inefficient Operation Status. When set, processor graphics frequency is operating below target frequency.	
15:13	Reserved.	
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log. When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	RATL_LOG (R/W) Running Average Thermal Limit Log. When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	INEFFICIENT_OPERATION_LOG (R/W) Inefficient Operation Log. When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:29	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
MSR_RING_PERF_LIMIT_REASONS Indicator of Frequency Clipping in the Ring Interconnect. (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced due to running average thermal limit.	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR_TDC (R/O) VR Thermal Design Current Status. When set, frequency is reduced due to VR TDC limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
15:12	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log. When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	RATL_LOG (R/W) Running Average Thermal Limit Log. When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:28	Reserved.	
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR_CORE_MKTME_ACTIVATE (R/O) MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS].		Core

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	Reserved.	
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field and must be zero on a WRMSR.	
63:40	Reserved.	

The MSRs listed in Table 2-54 are unique to the Intel Core Ultra 7 processor P-core. These MSRs are not supported on the processor E-core.

Table 2-54. MSRs Supported by the Intel® Core™ Ultra 7 Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter 3 (R/W)		Thread
47:0	FIXED_COUNTER Top-down Microarchitecture Analysis unhalting number of available slots counter.	
63:48	Reserved.	
Register Address: 329H, 809	MSR_PERF_METRICS	
Performance Metrics (R/W) This register provides built-in support for Top-down Micro-architecture Analysis (TMA) metrics. It exposes the four TMA Level 1 metrics where the lower 32 bits are divided into four 8 bit fields, each of which is an integer percentage of the total TOPDOWN.SLOTS (as reported by fixed counter 3).		Thread
7:0	RETIRING Percent of utilized by uops that eventually retire (commit).	
15:8	BAD_SPECULATION Percent of Wasted due to incorrect speculation, covering Utilized by uops that do not retire, or Recovery Bubbles (unutilized slots).	
23:16	FRONTEND_BOUND Percent of Unutilized slots where Front-end did not deliver a uop while Back-end is ready.	
31:24	BACKEND_BOUND Percent of Unutilized slots where a uop was not delivered to Back-end due to lack of Back-end resources.	
39:32	MULTI_UOPS Frontend bound.	
47:40	BRANCH_MISPREDICTS Frontend bound.	

Table 2-54. MSRs Supported by the Intel® Core™ Ultra 7 Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:48	FRONTEND_LATENCY Frontend bound.	
63:56	MEMORY_BOUND Frontend bound.	
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W) See Table 2-47.		Thread
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W) See Table 2-44.		Core

The MSRs listed in Table 2-48 are unique to the Intel Core Ultra 7 processor E-core. These MSRs are not supported on the processor P-core.

Table 2-55. MSRs Supported by the Intel® Core™ Ultra 7 Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
SAF Control (W/O) Extension to SAF.		Package
0	INVALIDATE_CURRENT_STRIDE Invalidate all chunks in current stride.	
63:1	Reserved.	
Register Address: D18H–D1FH, 3352–3359	IA32_L2_MASK_[8-15]	
IA32_L2_MASK_[8-15] (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Module
15:0	WAY_MASK Capacity Bit Mask. Available ways vectors for class of service of IA core. ‘1 in bit indicates allocation to the way is allowed. ‘0 indicates allocation to the way is not allowed.	
31:16	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		Thread
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBES_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C8H, 5313 –5320	MSR_RELOAD_PMCx	
Reload value for IA32_PMCx (R/W)		Thread

Table 2-55. MSRs Supported by the Intel® Core™ Ultra 7 Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	
Register Address: 1A8EH, 6798	MSR_STLB_FILL_TRANSLATION	
STLB Fill Translation (W/O) STLB QoS MSR to fill translations into STLB.		Core
3:0	CLOS Class of service to use for the fill.	
9:4	Reserved.	
10	X Set to 1 when LA is to an executable page.	
11	RW Set to 1 when LA is to a writeable page.	
63:12	LA Logical address to use for fill.	

2.17.10 MSRs Introduced in the Intel® Xeon® 6 P-Core Processors

Table 2-56 lists additional MSRs for the Intel Xeon 6 P-core processors. Intel Xeon 6 P-core processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_ADH or 06_AEH.

For an MSR listed in Table 2-56 that also appears in the model-specific tables of prior generations, Table 2-56 supersedes prior generation tables.

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CONTROL	
MSR_MEMORY_CONTROL (R/W) Disables split locks, which are locked instructions that split a cache line.		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor allows one in-progress, post-retirement UC stores at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC load lock will trigger a fault. If clear to 0, UC load locks proceed normally.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will trigger an #AC fault. If clear to 0, split locks proceed normally	
63:30	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
SMI Counter (R/W)		Thread
31:0	SMI_COUNT Running count of SMI events since the last reset.	
63:32	Reserved.	
Register Address: 39H, 57	MSR_SOCKET_ID	
Socket ID (R/W) Reassigns the package-specific portions of the APIC ID. This MSR is used on scalable DP and high-end MP platforms to resolve legacy-mode APIC ID conflicts.		Package
10:0	PACKAGE_ID: Holds package ID. This reflects the upper bits of the APIC ID.	
63:11	Reserved.	
Register Address: 7AH, 122	IA32_FEATURE_ACTIVATION	
IA32_FEATURE_ACTIVATION (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread. See Table 2-2.		Thread
Register Address: 7BH, 123	IA32_MCU_ENUMERATION	
IA32_MCU_ENUMERATION (R/O) Enumeration of architectural features. See Table 2-2.		Package
Register Address: 7CH, 124	IA32_MCU_STATUS	
IA32_MCU_STATUS (R/O) Communicates results from the previous patch loads. See Table 2-2.		Package
Register Address: 82H, 130	IA32_FZM_RANGE_INDEX	
IA32_FZM_RANGE_INDEX (R/W) Index and Domain handle for a valid FZM region. Programmed by SW and used by other FRM MSRs FZM Range Index register to R/W Domain Index. See Table 2-2.		Thread
Register Address: 83H, 131	IA32_FZM_DOMAIN_CONFIG	
IA32_FZM_DOMAIN_CONFIG (R/O) Bit mask of valid regions within the domain identified by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 84H, 132	IA32_FZM_RANGE_STARTADDR	
IA32_FZM_RANGE_STARTADDR (R/O) Start address of the FZM range pointed to by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 85H, 133	IA32_FZM_RANGE_ENDADDR	
IA32_FZM_RANGE_ENDADDR (R/O) End address of the specified domain in FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 86H, 134	IA32_FZM_RANGE_WRITESTATUS	
IA32_FZM_RANGE_WRITESTATUS (R/O) Write status of the FZM range pointed to by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 87H, 135	IA32_MKTME_KEYID_PARTITIONING	
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX. See Table 2-2.		Package

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 90H, 144	IA32_SGXLEPUBKEYHASH4	
IA32_SGXLEPUBKEYHASH4 (R/W) See Table 2-2.		Thread
Register Address: 91H, 145	IA32_SGXLEPUBKEYHASH5	
IA32_SGXLEPUBKEYHASH5 (R/W) See Table 2-2.		Thread
Register Address: 98H, 152	MSR_SEAM_WBINVDP	
SEAM WBINVDP (R/W) Allows software to WBINVD sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 99H, 153	MSR_SEAM_WBNOINVDP	
SEAM WBNOINVDP (R/W) Allows software to WBNOINVDP sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 9AH, 154	MSR_SEAM_INTR_PENDING	
SEAM Interrupt Pending (R/O) Report out some event pending bits.		Thread
0	INTR Interrupt is pending.	
1	NMI NMI is pending.	
2	SMI SMI is pending.	
4:3	OTHER_EVENTS Other events pending.	
63:5	Reserved.	
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Control (R/W) The SMM Monitor Configuration involves SMM code specifying the MSEG location and enabling dual-monitor treatment by writing to the corresponding MSR. See Table 2-2.		Thread
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	STLB_QOS When set to 1, processor supports STLB QoS.	
1	Reserved.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	INTEGRITY_SUPPORTED When set to 1, processor supports Functional Safety. Specific FUSA capabilities are enumerated in MSR_FUSA_CAPABILITIES.	
3	RSM_IN_CPLO_ONLY Intel System Resources Defense: When set to 1, RSM will only be allowed in CPLO and will #GP for all non-CPLO privilege levels.	
4	UC_LOCK_DISABLE When set to 1, processor supports UC load lock disable.	
5	SPLIT_LOCK_DISABLE When set to 1, processor supports #AC on split locks.	
6	SNP_FILTER_QOS When set to 1, processor supports Snoop Filter Quality of Service MSRs.	
7	UC_STORE_THROTTLING When set to 1, processor supports UC store throttling through MSR_MEMORY_CTRL[UC_STORE_THROTTLE].	
63:8	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 105H, 261	MSR_ARRAY_BIST	
MSR_ARRAY_BIST (R/W) Triggered by writing and reading an MSR that can be written by Ring 0 software.		Core

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	ARRAY_LIST: Bit map which indicates which arrays to run MarchC- BIST ▪ Bit[0] MLC Data ▪ Bit[1] MLC Tag ▪ Bit[2] C6SRAM Data (NOP for WRMSR – used for reporting error only) ▪ Bit[3] PMA BIST (NOP for WRMSR – used for reporting error only) ▪ Bit[4] STLB Data ▪ Bit[5] IFU Data ▪ Bit[6] STLB Tag ▪ Bit[7] DCU Data ▪ Bit[8] DSB Data ▪ Bit[9] TMUL Data ▪ Bit[10] UROM pointer0 ▪ Bit[11] UROM pointer1-3 ▪ Bit[12] UROM pointer4-7 ▪ Bit[13] UROM unique0 ▪ Bit[14] UROM unique1/2 The WRMSR will run PBIST on all the arrays indicated in the bitmap, starting from the LSB. NOTE2: C6SRAM[Bit 2] and PMA[Bit 3] are only for reporting and do not execute BIST (done by EDX[15:0]uCode during Fusa-Reset).	
46:32	Reserved.	
62:47	Reserved.	
63	SIGNAL_MCE: Signal MCERR upon BIST failure.	
Register Address: 105H, 261	MSR_ARRAY_BIST_STATUS	
MSR_ARRAY_BIST_STATUS (R/O)		Core
31:0	ARRAY_COMPLETION_MASK Bitmap indicating which arrays from the ARRAY_BIST.ARRAY_LIST was not processed. 1 means not tested and 0 means tested.	
62:32	Reserved. Returns all 0s.	
63	PASS_FAIL: 0 means Pass on all arrays in the WRMSR(ARRAY_BIST.ARRAY_LIST) 1 means Fail on the LSB array in the RDMSR(ARRAY_BIST_STATUS.ARRAY_COMPLETION_MASK).	
Register Address: 122H, 290	IA32_TSX_CTRL	
IA32_TSX_CTRL (R/W)		
See Table 2-2.		Thread
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Feature Configuration (R/W)		Core

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
1	AESNI_DISABLE This bit disables Advanced Encryption Standard feature on this processor core. To disable AES, BIOS will write '11 to this MSR on every core.	
63:2	Reserved.	
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Miscellaneous enables for thread-specific features. (R/W)		Thread
0	CPUID_GP_ON_CPL_GT_0 Causes CPUID to #GP if CPL greater than 0 and not in SMM.	
63:1	Reserved.	
Register Address: 1E0H, 480	IA32_LER_INFO	
IA32_LER_INFO (R/W) Last Event Record Destination IP Register. See Table 2-2.		Thread
Register Address: 1F9H, 505	IA32_CPU_DCA_CAP	
IA32_CPU_DCA_CAP (R/O) See Table 2-2.		Thread
Register Address: 2A1H, 673	MSR_PRMRR_BASE_1	
MSR_PRMRR_BASE_1 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register.		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
19:4	Reserved.	
51:20	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A2H, 674	MSR_PRMRR_BASE_2	
MSR_PRMRR_BASE_2 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A3H, 675	MSR_PRMRR_BASE_3	
MSR_PRMRR_BASE_3 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A4H, 676	MSR_PRMRR_BASE_4	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_PRMRR_BASE_4 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A5H, 677	MSR_PRMRR_BASE_5	
MSR_PRMRR_BASE_5 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A6H, 678	MSR_PRMRR_BASE_6	
MSR_PRMRR_BASE_6 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A7H, 679	MSR_PRMRR_BASE_7	
MSR_PRMRR_BASE_7 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2B8H, 696	MSR_COPY_SBFT_HASHES	
MSR_COPY_SBFT_HASHES (W/O)		Module
63:0	SBFT_PROGRAM_SOURCE_ADDR EDX:EAX contains the linear address base of the SBFT Binary loaded into memory.	
Register Address: 2B9H, 697	MSR_SBFT_HASHES_STATUS	
MSR_COPY_SBFT_HASHES (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	
39:32	ERROR_CODE - EDX[7:0] The error code refers to the LP that runs WRMSR(2B8H). ▪ 0x0: Reserved. ▪ 0x1: Attempt to copy SBFT-hashes when copy already in progress. ▪ 0x2: Secure Memory not set up correctly. ▪ 0x3: Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. ▪ 0x4: Reserved. ▪ 0x5: Integrity check failed. ▪ 0x6: WRMSR(0x2B8) (ACTIVATE_SBAF) Reinstall of SBFT test image attempted when current SBFT test image is in use by other LPs. ▪ 0x7: Aborted due to #PF (Page Fault). ▪ 0x8: Unable to generate a Random Value.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, stride value must be >= 1.	
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	MAX_CORE_LIMIT EDX[30:19] - Maximum Number of Cores that can run SBFTAFSBAF simultaneously -1. 0 means 1 core at a time.	
63	Valid. EDX[31] - Valid bit is set when COPY_SBFT_HASHES completed successfully.	
Register Address: 2BAH, 698	MSR_AUTHENTICATE_AND_COPY_SBFT_CHUNK	
MSR_AUTHENTICATE_AND_COPY_SBFT_CHUNK (W/O)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2BBH, 699	MSR_SBFT_CHUNKS_AUTHENTICATION_STATUS	
MSR_SBFT_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	NUM_VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	
39:32	ERROR_CODE EDX[7:0] 0x0 - No error reported. 0x1 - Attempt to authenticate a CHUNK already marked as authentic or is currently being installed by another core. 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. 0x3 - Aborted due to #PF. 0x4 - Chunk Outside the current Stride. 0x5 - Interrupted.	
47:40	Reserved. EDX[15:8] - Set to all zeros.	
63:48	CURRENT_MAX_BUNDLE_INDX EDX[31:16] - Maximum Bundle Index in current stride.	
Register Address: 2BCH, 700	MSR_ACTIVATE_SBFT	
MSR_ACTIVATE_SBFT (W/O)		Core
13:0	SBFT_BUNDLE_INDEX EAX[13:0] - Indicates SBFT Bundle Index to start from.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
15:14	SBFT_PRGM_INDEX EAX[15:14] - Indicates what SBFT Program index to run.	
31:16	Reserved. Set to all zeros.	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC-based delay to allow threads to rendezvous.	
63	Reserved. EDX[31] - Must be set to 0. #GP fault otherwise.	
Register Address: 2BDH, 701	MSR_SBFT_STATUS	
MSR_SBFT_STATUS (R/O)		Core
13:0	SBFT_BUNDLE_INDEX EAX[13:0] - SBFT Bundle that was executed.	
15:14	SBFT_PGM_INDEX EAX[15:14] - Indicates what SBFT Program index that was last ran. Maps to same field in WRMSR(ACTIVATE_SBFT). On a test pass this field will be 2'b00.	
31:16	Reserved. EAX[31:16] - Return all zeros.	
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SBFT operation did not start. Other thread could not join. 0x2 - SBFT operation did not start. Interrupt occurred prior to SBFT coordination. 0x3 - Reserved. 0x4 - SBFT operation did not start. Non-valid SBFT BUNDLES in the SBFT_BUNDLE_INDEX. 0x5 - SBFT operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SBFT operation did not start. Core is not capable of performing SBFT currently. 0x7 - Reserved. 0x8 - SBFT operation did not start. Exceeded number of Logical Processors (LP) allowed to run SBFT-At-Field concurrently. 0x9 - SBFT operation did not start. Interrupt occurred or timer about to expire. 0xA - SBFT operation did not start. SBFT_PGM_INDEX is not valid. 0xB - SBFT operation aborted due to corrupted chunk. 0xC - SBFT operation did not start. TAP Data error. 0xD - SBFT operation did not start. SBFT program is not valid. All other error codes are reserved.	
60:40	Reserved. EDX[28:8] - Return all zeros.	
61	TEST_FAIL EDX[29:29] - Architectural Signature failed. Last thread executed HLT and completed SBFT and EBX != 0xACED.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:62	SBFT_STATUS EDX[31:30] - SBFT status (result of running SBAF). ▪ 00 - PASS. ▪ 10 - INTERRUPTED. ▪ 01 - FAILED SIGNATURE CHECK. ▪ 11 - FAILED.	
Register Address: 2BEH, 702	MSR_SBFT_MODULE_ID	
MSR_SBFT_MODULE_ID (R/O)		Module
31:0	SBFT-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2BFH, 703	MSR_SBFTAF_LAST_WP	
MSR_SBFTAF_LAST_WP (R/O)		Module
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SBFT) was executed. Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved.	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	
MSR_COPY_SCAN_HASHES (W/O)		Module
63:0	SCAN_HASH-ADDR EDX:EAX contains the linear address of the SCAN Test HASH Binary loaded into memory	
Register Address: 2C3H, 707	MSR_SCAN_HASHES_STATUS	
MSR_SCAN_HASHES_STATUS (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE EDX[7:0] - The error code refers to the LP that runs WRMSR(2C2H). 0x0 - Reserved. 0x1 - Attempt to copy scan-hashes when copy already in progress. 0x2 - Secure Memory not set up correctly. 0x3 - Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4 - Reserved. 0x5 - Integrity check failed. 0x6 - WRMSR(0x2C6) Re-install of scan test image attempted when current scan test image is in use by other LPs. 0x7 - Aborted due to #PF (Page Fault). 0x8 - Unable to generate a Random Value.	
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, the stride value must be >= 1.	
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	NAME EDX[30:19] - Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	
63	VALID EDX[31] - Valid bit is set when COPY_SCAN_HASHES completed.	
Register Address: 2C4H, 708	MSR_AUTHENTICATE_AND_COPY_CHUNK	
MSR_AUTHENTICATE_AND_COPY_CHUNK (R/O)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2C5H, 709	MSR_CHUNKS_AUTHENTICATION_STATUS	
MSR_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE EDX[7:0] ▪ 0x0 - No-error reported. ▪ 0x1 - Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. ▪ 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. ▪ 0x3 - Aborted due to #PF (Page Fault). ▪ 0x4 - Chunk Outside the current Stride.	
63:40	Reserved. EDX[31:8] - Set to all zeros.	
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
MSR_ACTIVATE_SCAN (W/O)		Core
15:0	CHUNK_START_INDEX EAX[15:0] - Indicates Chunk Index from which to start.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive).	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC based delay to allow threads to rendezvous.	
63	SIGNAL_MCE EDX[31] ▪ If 1: On scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. ▪ If 0: Don't no-logging/no-signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
MSR_SCAN_STATUS (R/O)		Core
15:0	CHUNK_NUM EAX[15:0] - SCAN Chunk that was reached.	
31:16	CHUNK_STOP_INDEX EAX[31:16] ▪ Indicates what chunk index to stop at (inclusive). ▪ Maps to same field in WRMSR(ACTIVATE_SCAN).	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SCAN operation did not start. Other thread could not join. 0x2 - SCAN operation did not start. Interrupt occurred prior to SCAN coordination. 0x3 - SCAN operation did not start. Power Management conditions are inadequate to run SAF. 0x4 - SCAN operation did not start. Non valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5 - SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SCAN operation did not start. Core not capable of performing SCAN currently. 0x7 - Debug Mode. Scan-At-Field results not to be trusted. 0x8 - SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Scan-At-Field concurrently. MAX_CORE_LIMIT exceeded. 0x9 - Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed. 0xB - Scan operation aborted due to corrupted chunk. 0xC - Scan operation did not start. All other error codes are reserved.	
61:40	Reserved. EDX[29:8] - Return all zeros.	
62	SCAN_CONTROL_ERROR EDX[30] - SCAN error in the Scan-At-Field controller. - Non ECC error.	
63	SCAN_SIGNATURE_ERROR EDX[31] - SCAN SIGNATURE error in the SCAN pattern fetched from main memory. - Non ECC error.	
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	
MSR_SCAN_MODULE_ID (R/O)		Module
31:0	SCAN-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2C9H, 713	MSR_LAST_SAF_WP	
MSR_LAST_SAF_WP (R/O)		Module

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. - Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved. EDX[7:0]	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2D9H, 729	MSR_INTEGRITY_CAPABILITIES	
MSR_INTEGRITY_CAPABILITIES (R/O) Enumerates features supported in Functional Safety.		Thread
0	STARTUP_SCAN_BIST When set to 1, processor supports Startup SCAN BIST.	
1	STARTUP_MEM_BIST When set to 1, processor supports Startup MEM BIST.	
2	PERIODIC_MEM_BIST When set to 1, processor supports Periodic MEM BIST.	
3	LOCKSTEP When set to 1, processor supports Lock Step Mode.	
4	PERIODIC_SCAN_BIST When set to 1, processor supports Periodic SCAN BIST.	
5	PLL_LOSS_DETECT When set to 1, processor supports PLL LOSS detection.	
6	PWR_LOSS_DETECT When set to 1, processor supports Power Loss detection.	
7	PERRINJ When set to 1, processor supports FUSA PERRINJ.	
8	SBFT_AT_FIELD When set to 1, processor supports SBFT-At-Field.	
10:9	SAF_GEN_REV 00 = REV1; 01 = REV2; 10 = REV3; 11 = REV4.	
14:11	Reserved.	
15	PRESERVE_MEMORY_NEEDED When set to 1, processor supports FUSARR_BASE/MASK MSRs.	
20:16	TID_BIT_SHIFT Number of bits to shift right on x2APICID to get a unique topology ID of all logical processors that share a scan test engine.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
21	ALL_LP_JOIN_NEEDED All logical processors that share scan test engine need to be tested together and must join using MSR_ACTIVATE_SCAN.	
23:22	Reserved.	
31:24	PATTERN_ID Processor scan pattern ID. ID of the startup and periodic scan programs supported for this part.	
63:32	Reserved.	
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 4DOH, 1232	IA32_MCG_EXT_CTL	
IA32_MCG_EXT_CTL (R/W) See Table 2-2.		Thread
Register Address: 4FOH, 1264	MSR_SAF_CTRL	
MSR_SAF_CTRL (W/O)		Core
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride. - Clears only the VALID_CHUNKS field on a RDMSR(CHUNKS_AUTHENTICATION_STATUS).	
63:1	Reserved.	
Register Address: 4F8H, 1272	MSR_SBFT_CTRL	
MSR_SBFT_CTRL (W/O)		Module
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride.	
63:1	Reserved. EDX[31:0],EAX[31:1]	
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W)		Thread
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W)		Core
0	SCRUB_DIS L1 scrubbing disable.	
63:1	Reserved.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 664H, 1636	MSR_MC6_RESIDENCY	
MSR_MC6_RESIDENCY (R/O) Time spent in the Module C6-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	Module	
63:0	RESIDENCY Time that this module is in module-specific C6 states since last reset.	
Register Address: 6E1H, 1761	IA32_PKRS	
IA32_PKRS (R/W) Specifies the PK permissions associated with each protection domain for supervisor pages. See Table 2-2.	Thread	
Register Address: 7A3H, 1955	IA32_MCU_EXT_SERVICE	
MCU Extended Service MSR (R/O) If IA32_ARCH_CAPABILITIES[22] = 1. See Table 2-2.	Module	
Register Address: 7A4H, 1956	IA32_MCU_ROLLBACK_MIN_ID	
Minimal MCU Revision ID for Rollback (R/O) See Table 2-2.	Module	
Register Address: 7B0H, 1968	IA32_ROLLBACK_SIGN_ID_0	
Rollback ID 0 (R/O) See Table 2-2.	Module	
Register Address: 7B1H, 1969	IA32_ROLLBACK_SIGN_ID_1	
Rollback ID 1 (R/O) See Table 2-2.	Module	
Register Address: 7B2H, 1970	IA32_ROLLBACK_SIGN_ID_2	
Rollback ID 2 (R/O) See Table 2-2.	Module	
Register Address: 7B3H, 1971	IA32_ROLLBACK_SIGN_ID_3	
Rollback ID 3 (R/O) See Table 2-2.	Module	
Register Address: 7B4H, 1972	IA32_ROLLBACK_SIGN_ID_4	
Rollback ID 4 (R/O) See Table 2-2.	Module	
Register Address: 7B5H, 1973	IA32_ROLLBACK_SIGN_ID_5	
Rollback ID 5 (R/O) See Table 2-2.	Module	
Register Address: 7B6H, 1974	IA32_ROLLBACK_SIGN_ID_6	
Rollback ID 6 (R/O) See Table 2-2.	Module	
Register Address: 7B7H, 1975	IA32_ROLLBACK_SIGN_ID_7	
Rollback ID 7 (R/O) See Table 2-2.	Module	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 7B8H, 1976	IA32_ROLLBACK_SIGN_ID_8	
Rollback ID 8 (R/O) See Table 2-2.	Module	
Register Address: 7B9H, 1977	IA32_ROLLBACK_SIGN_ID_9	
Rollback ID 9 (R/O) See Table 2-2.	Module	
Register Address: 7BAH, 1978	IA32_ROLLBACK_SIGN_ID_10	
Rollback ID 10 (R/O) See Table 2-2.	Module	
Register Address: 7BBH, 1979	IA32_ROLLBACK_SIGN_ID_11	
Rollback ID 11 (R/O) See Table 2-2.	Module	
Register Address: 7BCH, 1980	IA32_ROLLBACK_SIGN_ID_12	
Rollback ID 12 (R/O) See Table 2-2.	Module	
Register Address: 7BDH, 1981	IA32_ROLLBACK_SIGN_ID_13	
Rollback ID 13 (R/O) See Table 2-2.	Module	
Register Address: 7BEH, 1982	IA32_ROLLBACK_SIGN_ID_14	
Rollback ID 14 (R/O) See Table 2-2.	Module	
Register Address: 7BFH, 1983	IA32_ROLLBACK_SIGN_ID_15	
Rollback ID 15 (R/O) See Table 2-2.	Module	
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
IA32_TME_CAPABILITY (R/O) See Table 2-2.	Package	
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
IA32_TME_ACTIVATE (R/W) See Table 2-2.	Package	
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
Intel TME Exclude Mask (R/W) See Table 2-2.	Package	
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
Intel TME Exclude Base (R/W) See Table 2-2.	Package	
Register Address: 985H, 2437	IA32_UINTR_RR	
User Interrupt Request Register (R/W) See Table 2-2.	Thread	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 986H, 2438	IA32_UINTR_HANDLER	
User Interrupt Handler Address (R/W) See Table 2-2.		Thread
Register Address: 987H, 2439	IA32_UINTR_STACKADJUST	
User Interrupt Stack Adjustment (R/W) See Table 2-2.		Thread
Register Address: 988H, 2440	IA32_UINTR_NV	
User-Interrupt Size and Notification Vector (R/W) See Table 2-2.		Thread
Register Address: 989H, 2441	IA32_UINTR_PD	
User Interrupt PID Address (R/W) See Table 2-2.		Thread
Register Address: 98AH, 2442	IA32_UINTR_TT	
User-Interrupt Target Table (R/W) See Table 2-2.		Thread
Register Address: 990H, 2448	IA32_COPY_STATUS	
IA32_COPY_STATUS (R/O) See Table 2-2.		Thread
Register Address: 991H, 2449	IA32_IWKEYBACKUP_STATUS	
IA32_IWKEYBACKUP_STATUS (R/O) See Table 2-2.		Package
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS] (R/O)		Core
31:0	Reserved.	
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field. #GP on a non-zero write.	
63:40	Reserved.	
Register Address: C84H, 3204	MSR_MBA_CFG	
Memory Bandwidth Allocation (MBA) Configuration (R/W)		Package
1:0	Reserved.	
2	RAMBAE Resource Aware MBA Enable.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:3	Reserved.	
Register Address: CA0H, 3232	MSR_RMID_SNC_CONFIG	
RMID_SNC_CONFIG (R/W)		Package
0	RMID_LOCALIZED_DISTRIBUTION_MODE_ENABLE If set, Localized RMID distribution mode is enabled. If Clear, RMID Sharing mode is enabled.	
63:1	Reserved.	
Register Address: D50H, 3408	IA32_L2_QOS_EXT_BW_THRTL_0	
Memory Bandwidth Enforcement for COS0 (R/W) See Table 2-2.		Package
Register Address: D51H, 3409	IA32_L2_QOS_EXT_BW_THRTL_1	
Memory Bandwidth Enforcement for COS1 (R/W) See Table 2-2.		Package
Register Address: D52H, 3410	IA32_L2_QOS_EXT_BW_THRTL_2	
Memory Bandwidth Enforcement for COS2 (R/W) See Table 2-2.		Package
Register Address: D53H, 3411	IA32_L2_QOS_EXT_BW_THRTL_3	
Memory Bandwidth Enforcement for COS3 (R/W) See Table 2-2.		Package
Register Address: D54H, 3412	IA32_L2_QOS_EXT_BW_THRTL_4	
Memory Bandwidth Enforcement for COS4 (R/W) See Table 2-2.		Package
Register Address: D55H, 3413	IA32_L2_QOS_EXT_BW_THRTL_5	
Memory Bandwidth Enforcement for COS5 (R/W) See Table 2-2.		Package
Register Address: D56H, 3414	IA32_L2_QOS_EXT_BW_THRTL_6	
Memory Bandwidth Enforcement for COS6 (R/W) See Table 2-2.		Package
Register Address: D57H, 3415	IA32_L2_QOS_EXT_BW_THRTL_7	
Memory Bandwidth Enforcement for COS7 (R/W) See Table 2-2.		Package
Register Address: D58H, 3416	IA32_L2_QOS_EXT_BW_THRTL_8	
Memory Bandwidth Enforcement for COS8 (R/W) See Table 2-2.		Package
Register Address: D59H, 3417	IA32_L2_QOS_EXT_BW_THRTL_9	
Memory Bandwidth Enforcement for COS9 (R/W) See Table 2-2.		Package
Register Address: D5AH, 3418	IA32_L2_QOS_EXT_BW_THRTL_10	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Memory Bandwidth Enforcement for COS10 (R/W) See Table 2-2.		Package
Register Address: D5BH, 3419	IA32_L2_QOS_EXT_BW_THRTL_11	
Memory Bandwidth Enforcement for COS11 (R/W) See Table 2-2.		Package
Register Address: D5CH, 3420	IA32_L2_QOS_EXT_BW_THRTL_12	
Memory Bandwidth Enforcement for COS12 (R/W) See Table 2-2.		Package
Register Address: D5DH, 3421	IA32_L2_QOS_EXT_BW_THRTL_13	
Memory Bandwidth Enforcement for COS13 (R/W) See Table 2-2.		Package
Register Address: D5EH, 3422	IA32_L2_QOS_EXT_BW_THRTL_14	
Memory Bandwidth Enforcement for COS14 (R/W) See Table 2-2.		Package
Register Address: D91H, 3473	IA32_COPY_LOCAL_TO_PLATFORM	
See Table 2-2.		Thread
Register Address: D92H, 3474	IA32_COPY_PLATFORM_TO_LOCAL	
See Table 2-2.		Thread
Register Address: D93H, 3475	IA32_PASID	
See Table 2-2.		Thread
Register Address: 1400H, 5120	IA32_SEAMRR_BASE	
SEAM Memory Range Register for TDx - Base Address (R/W) See Table 2-2.		Core
Register Address: 1401H, 5121	IA32_SEAMRR_MASK	
SEAM Memory Range Register for TDX (R/W) See Table 2-2.		Core
Register Address: 1A8FH, 6799	MSR_STLB_QOS_INFO	
STLB_QOS_INFO (R/O) STLB QoS MASK configuration.		Core
5:0	NCLOS Number of CLOS supported for STLB resource using minus-1 notation.	
15:6	Reserved.	
19:16	4K_2M_CBM Length of capacity bitmask for 4K and 2M pages using minus-1 notation.	
28:20	Reserved.	
29	STLB_FILL_TRANSLATION_MSR_SUPPORTED MSR interface to fill STLB translations supported.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30	4K_2M_ALIAS Indicates that 4K/2M pages alias into the same structure.	
63:31	Reserved.	
Register Address: 1B01H, 6913	IA32_UARCH_MISC_CTL	
IA32_UARCH_MISC_CTL (R/W) See Table 2-2.		Thread

2.17.11 MSRs Introduced in the Intel® Xeon® 6 E-Core Processors

Table 2-57 lists additional MSRs for the Intel Xeon 6 E-core processors. Intel Xeon 6 E-core processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_AFH.

For an MSR listed in Table 2-57 that also appears in the model-specific tables of prior generations, Table 2-57 supersedes prior generation tables.

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2FH, 47	IA32_BARRIER	
BARRIER (R/O) The IA32_BARRIER MSR ensures ordered execution by acting like LFENCE, controlling the sequencing of subsequent MSR reads after prior MSR reads and instructions. See Table 2-2.		Core
Register Address: 33H, 51	MSR_MEMORY_CONTROL	
Memory Control (R/W) Disables split locks, which are locked instructions that split a cache line.		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor allows one in-progress, post-retirement UC stores at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC load lock will trigger a fault. If clear to 0, UC load locks proceed normally.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will trigger an #AC fault. If clear to 0, split locks proceed normally.	
63:30	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/W)		Thread
31:0	SMI_COUNT Running count of SMI events since the last reset.	
63:32	Reserved.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 39H, 57	MSR_SOCKET_ID	
Socket ID (R/W) Reassigns the package-specific portions of the APIC ID. This MSR is used on scalable DP and high-end MP platforms to resolve legacy-mode APIC ID conflicts.		Package
10:0	PACKAGE_ID Holds package ID. This reflects the upper bits of the APIC ID.	
63:11	Reserved.	
Register Address: 7BH, 123	IA32_MCU_ENUMERATION	
Enumeration of Architectural Features (R/O) See Table 2-2.		Package
Register Address: 7CH, 124	IA32_MCU_STATUS	
MCU Status (R/O) Communicates results from the previous patch loads. See Table 2-2.		Package
Register Address: 87H, 135	IA32_MKTME_KEYID_PARTITIONING	
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX. See Table 2-2.		Package
Register Address: 98H, 152	MSR_SEAM_WBINVDP	
SEAM WBINVDP (R/W) Allows software to WBINVD sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 99H, 153	MSR_SEAM_WBNOINVDP	
SEAM WBNOINVDP (R/W) Allows software to WBNOINVDP sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 9AH, 154	MSR_SEAM_INTR_PENDING	
SEAM Interrupt Pending (R/O) Report out some event pending bits.		Thread
0	INTR Interrupt is pending.	
1	NMI NMI is pending.	
2	SMI SMI is pending.	
4:3	OTHER_EVENTS Other events pending.	
63:5	Reserved.	
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
SMM Monitor Control (R/W) The SMM Monitor Configuration involves SMM code specifying the MSEG location and enabling dual-monitor treatment by writing to the corresponding MSR. See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information (R/O) Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
15:8	MAX_NON_TURBO_LIM_RATIO This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	
25:16	Reserved.	
26	DCU_16K_MODE_AVAIL 0b: Indicates that the part does not support the 16K DCU mode. 1b: Indicates that the part supports 16K DCU mode.	
27	Reserved.	
28	PRG_TURBO_RATIO_EN Programmable Turbo Ratios per number of Active Cores. 0 = Programming Not Allowed. 1 = Programming Allowed.	
34:29	Reserved.	
35	BIOS_GUARD_ENABLE Indicates whether the BIOS Guard feature is enabled in the CPU.	
36	PEG2DMIDIS_EN 0 = PEG2DMIDIS is disabled. 1 = PEG2DMIDIS is enabled.	
39:37	Reserved.	
47:40	MAX_EFFICIENCY_RATIO Maximum Efficiency Ratio. This is given in units of 100 MHz.	
58:48	Reserved.	
59	SMM_SUPOVR_STATE_LOCK_ENABLE When set, indicates that the CPU supports MSR SMM_SUPOVR_STATE_LOCK and the Hardware Shield feature.	
63:60	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	STLB_QOS When set to 1, processor supports STLB QoS.	
1	Reserved.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	INTEGRITY_SUPPORTED When set to 1, processor supports Functional Safety. Specific FUSA capabilities are enumerated in MSR_FUSA_CAPABILITIES.	
3	RSM_IN_CPLO_ONLY Intel System Resources Defense: When set to 1, RSM will only be allowed in CPLO and will #GP for all non-CPLO privilege levels.	
4	UC_LOCK_DISABLE When set to 1, processor supports UC load lock disable.	
5	SPLIT_LOCK_DISABLE When set to 1, processor supports #AC on split locks.	
6	SNP_FILTER_QOS When set to 1, processor supports Snoop Filter Quality of Service MSRs.	
7	UC_STORE_THROTTLING When set to 1, processor supports UC store throttling through MSR_MEMORY_CTRL[UC_STORE_THROTTLE].	
63:8	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Feature Configuration (R/W)		Core
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
1	AESNI_DISABLE This bit disables Advanced Encryption Standard feature on this processor core. To disable AES, BIOS will write '11 to this MSR on every core.	
63:2	Reserved.	
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Miscellaneous Enables for Thread-Specific Features (R/W)		Thread
0	CPUID_GP_ON_CPL_GT_0 Causes CPUID to #GP if CPL greater than 0 and not in SMM.	
63:1	Reserved.	
Register Address: 1BOH, 432	IA32_ENERGY_PERF_BIAS	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_ENERGY_PERF_BIAS (R/W) See Table 2-2.		Thread
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
IA32_PACKAGE_THERM_STATUS See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
IA32_PACKAGE_THERM_INTERRUPT (R/W) See Table 2-2.		Package
Register Address: 2A1H, 673	MSR_PRMRR_BASE_1	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A2H, 674	MSR_PRMRR_BASE_2	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A3H, 675	MSR_PRMRR_BASE_3	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_COPY_SCAN_HASHES (W/O)		Module
63:0	SCAN_HASH-ADDR EDX:EAX contains the linear address of the SCAN Test HASH Binary loaded into memory	
Register Address: 2C3H, 707		MSR_SCAN_HASHES_STATUS
MSR_SCAN_HASHES_STATUS (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	
39:32	ERROR_CODE EDX[7:0] - The error code refers to the LP that runs WRMSR(2C2H). 0x0 - Reserved. 0x1 - Attempt to copy scan-hashes when copy already in progress. 0x2 - Secure Memory not set up correctly. 0x3 - Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4 - Reserved. 0x5 - Integrity check failed. 0x6 - WRMSR(0x2C6) Re-install of scan test image attempted when current scan test image is in use by other LPs. 0x7 - Aborted due to #PF (Page Fault). 0x8 - Unable to generate a Random Value.	
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, the stride value must be >= 1	
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	NAME EDX[30:19] - Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	
63	VALID EDX[31] - Valid bit is set when COPY_SCAN_HASHES completed.	
Register Address: 2C4H, 708		MSR_AUTHENTICATE_AND_COPY_CHUNK
MSR_AUTHENTICATE_AND_COPY_CHUNK(R/O)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2C5H, 709		MSR_CHUNKS_AUTHENTICATION_STATUS

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	
39:32	ERROR_CODE EDX[7:0] 0x0 - No-error reported. 0x1 - Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. 0x3 - Aborted due to #PF (Page Fault). 0x4 - Chunk Outside the current Stride.	
63:40	Reserved. EDX[31:8] - Set to all zeros.	
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
MSR_ACTIVATE_SCAN (W/O)		Core
15:0	CHUNK_START_INDEX EAX[15:0] - Indicates Chunk Index from which to start.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive).	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC based delay to allow threads to rendezvous.	
63	SIGNAL_MCE EDX[31] - If 1: On scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. - If 0: Don't no-logging/no-signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
MSR_SCAN_STATUS (R/O)		Core
15:0	CHUNK_NUM EAX[15:0] - SCAN Chunk that was reached.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive). - Maps to same field in WRMSR(ACTIVATE_SCAN).	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SCAN operation did not start. Other thread could not join. 0x2 - SCAN operation did not start. Interrupt occurred prior to SCAN coordination. 0x3 - SCAN operation did not start. Power Management conditions are inadequate to run SAF. 0x4 - SCAN operation did not start. Non valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5 - SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SCAN operation did not start. Core not capable of performing SCAN currently. 0x7 - Debug Mode. Scan-At-Field results not to be trusted. 0x8 - SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Scan-At-Field concurrently. MAX_CORE_LIMIT exceeded. 0x9 - Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed. 0xB - Scan operation aborted due to corrupted chunk. 0xC - Scan operation did not start. All other error codes are reserved.	
61:40	Reserved. EDX[29:8] - Return all zeros.	
62	SCAN_CONTROL_ERROR EDX[30] - SCAN error in the Scan-At-Field controller. - Non ECC error.	
63	SCAN_SIGNATURE_ERROR EDX[31] - SCAN SIGNATURE error in the SCAN pattern fetched from main memory. - Non ECC error.	
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	
MSR_SCAN_MODULE_ID (R/O)		Module
31:0	SCAN-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2C9H, 713	MSR_LAST_SAF_WP	
MSR_LAST_SAF_WP (R/O)		Module

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. - Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved. EDX[7:0]	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2D6H, 726	MSR_TRIGGER_PERIODIC_MEM_BIST	
MSR_TRIGGER_PERIODIC_MEM_BIST (W/O)		Core
0	SIGNAL_MCE EAX[0] - If 1, then signal MCE on fail if machine check signaling enabled in the corresponding MCI_CTL. If 0 then don't signal machine checks.	
7:1	ARRAY_BANK EAX[7:1] - Reserved.	
15:8	TST_STEP_PARAM EAX[15:8] 0: Test All Arrays, or Test Arrays in STEPs of NUM_STEPS.	
31:16	Reserved. EAX[31:16]	
63:32	Reserved. EAX[31:0]	
Register Address: 2D7H, 727	MSR_PERIODIC_MEM_BIST_STATUS	
MSR_PERIODIC_MEM_BIST_STATUS (R/O)		Core
0	MEM_BIST_STATUS 0: PASS. 1: FAIL.	
63:1	Reserved.	
Register Address: 2D9H, 729	MSR_INTEGRITY_CAPABILITIES	
MSR_INTEGRITY_CAPABILITIES (R/O) Enumerates features supported in Functional Safety.		Thread
0	STARTUP_SCAN_BIST When set to 1, processor supports Startup SCAN BIST.	
1	STARTUP_MEM_BIST When set to 1, processor supports Startup MEM BIST.	
2	PERIODIC_MEM_BIST When set to 1, processor supports Periodic MEM BIST.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3	LOCKSTEP When set to 1, processor supports Lock Step Mode.	
4	PERIODIC_SCAN_BIST When set to 1, processor supports Periodic SCAN BIST.	
5	PLL_LOSS_DETECT When set to 1, processor supports PLL LOSS detection.	
6	PWR_LOSS_DETECT When set to 1, processor supports Power Loss detection.	
7	PERRINJ When set to 1, processor supports FUSA PERRINJ.	
8	SBFT_AT_FIELD When set to 1, processor supports SBFT-At-Field.	
10:9	SAF_GEN_REV 00 = REV1; 01 = REV2; 10 = REV3; 11 = REV4.	
14:11	Reserved.	
15	PRESERVE_MEMORY_NEEDED When set to 1, processor supports FUSARR_BASE/MASK MSRs.	
20:16	TID_BIT_SHIFT Number of bits to shift right on x2APICID to get a unique topology ID of all logical processors that share a scan test engine.	
21	ALL_LP_JOIN_NEEDED All logical processors that share scan test engine need to be tested together and must join using MSR_ACTIVATE_SCAN.	
23:22	Reserved.	
31:24	PATTERN_ID Processor scan pattern ID. ID of the startup and periodic scan programs supported for this part.	
63:32	Reserved.	
Register Address: 2DCH, 732	IA32_INTEGRITY_STATUS	
IA32_INTEGRITY_STATUS (R/O) Provides status information for integrity features. See Table 2-2.		Thread
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
MSR_PKG_C6_RESIDENCY (R/O)		Package
63:0	Package C6 Residency Counter	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
MSR_PKG_C7_RESIDENCY (R/O)		Package
63:0	Package C7 Residency Counter	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_CORE_C3_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
MSR_CORE_C6_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 3FEH, 1022	MSR_CORE_C7_RESIDENCY	
MSR_CORE_C7_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C7 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
MSR_SAF_CTRL (W/O)		Core
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride. - Clears only the VALID_CHUNKS field on a RDMSR(CHUNKS_AUTHENTICATION_STATUS).	
63:1	Reserved.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY	
MSR_MC6_RESIDENCY (R/O) Time spent in the Module C6-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).		Module
63:0	RESIDENCY Time that this module is in module-specific C6 states since last reset.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Thread
Register Address: 7A3H, 1955	IA32_MCU_EXT_SERVICE	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MCU Extended Service (R/W) See Table 2-2.		Module
Register Address: 7A4H, 1956	IA32_MCU_ROLLBACK_MIN_ID	
Minimal MCU Revision ID (R/O) See Table 2-2.		Module
Register Address: 7A5H, 1957	IA32_MCU_STAGING_MBOX_ADDR	
IA32_MCU_STAGING_MBOX_ADDR (R/O) See Table 2-2.		Package
Register Address: 7B0H, 1968	IA32_ROLLBACK_SIGN_ID_0	
Rollback ID 0 (R/O) See Table 2-2.		Module
Register Address: 7B1H, 1969	IA32_ROLLBACK_SIGN_ID_1	
Rollback ID 1 (R/O) See Table 2-2.		Module
Register Address: 7B2H, 1970	IA32_ROLLBACK_SIGN_ID_2	
Rollback ID 2 (R/O) See Table 2-2.		Module
Register Address: 7B3H, 1971	IA32_ROLLBACK_SIGN_ID_3	
Rollback ID 3 (R/O) See Table 2-2.		Module
Register Address: 7B4H, 1972	IA32_ROLLBACK_SIGN_ID_4	
Rollback ID 4 (R/O) See Table 2-2.		Module
Register Address: 7B5H, 1973	IA32_ROLLBACK_SIGN_ID_5	
Rollback ID 5 (R/O) See Table 2-2.		Module
Register Address: 7B6H, 1974	IA32_ROLLBACK_SIGN_ID_6	
Rollback ID 6 (R/O) See Table 2-2.		Module
Register Address: 7B7H, 1975	IA32_ROLLBACK_SIGN_ID_7	
Rollback ID 7 (R/O) See Table 2-2.		Module
Register Address: 7B8H, 1976	IA32_ROLLBACK_SIGN_ID_8	
Rollback ID 8 (R/O) See Table 2-2.		Module
Register Address: 7B9H, 1977	IA32_ROLLBACK_SIGN_ID_9	
Rollback ID 9 (R/O) See Table 2-2.		Module
Register Address: 7BAH, 1978	IA32_ROLLBACK_SIGN_ID_10	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Rollback ID 10 (R/O) See Table 2-2.		Module
Register Address: 7BBH, 1979	IA32_ROLLBACK_SIGN_ID_11	
Rollback ID 11 (R/O) See Table 2-2.		Module
Register Address: 7BCH, 1980	IA32_ROLLBACK_SIGN_ID_12	
Rollback ID 12 (R/O) See Table 2-2.		Module
Register Address: 7BDH, 1981	IA32_ROLLBACK_SIGN_ID_13	
Rollback ID 13 (R/O) See Table 2-2.		Module
Register Address: 7BEH, 1982	IA32_ROLLBACK_SIGN_ID_14	
Rollback ID 14 (R/O) See Table 2-2.		Module
Register Address: 7BFH, 1983	IA32_ROLLBACK_SIGN_ID_15	
Rollback ID 15 (R/O) See Table 2-2.		Module
Register Address: 988H, 2440	IA32_UINTR_NV	
User Interrupt Size and Notification Vector (R/W) See Table 2-2.		Thread
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR_CORE_MKTME_ACTIVATE (R/O) MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS].		Core
31:0	Reserved.	
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field. #GP on a non-zero write.	
63:40	Reserved.	
Register Address: C84H, 3204	MSR_MBA_CFG	
Memory Bandwidth Allocation (MBA) Configuration (R/W)		Package
1:0	Reserved.	
2	RAMBAE Resource Aware MBA Enable.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:3	Reserved.	
Register Address: CA0H, 3232	MSR_RMID_SNC_CONFIG	
MSR_RMID_SNC_CONFIG (R/W)		Package
0	RMID_LOCALIZED_DISTRIBUTION_MODE_ENABLE If set, Localized RMID distribution mode is enabled. If Clear, RMID Sharing mode is enabled.	
63:1	Reserved.	
Register Address: D50H, 3408	IA32_L2_QOS_EXT_BW_THRTL_0	
Memory Bandwidth Enforcement for COS0 (R/W) See Table 2-2.		Package
Register Address: D51H, 3409	IA32_L2_QOS_EXT_BW_THRTL_1	
Memory Bandwidth Enforcement for COS1 (R/W) See Table 2-2.		Package
Register Address: D52H, 3410	IA32_L2_QOS_EXT_BW_THRTL_2	
Memory Bandwidth Enforcement for COS2 (R/W) See Table 2-2.		Package
Register Address: D53H, 3411	IA32_L2_QOS_EXT_BW_THRTL_3	
Memory Bandwidth Enforcement for COS3 (R/W) See Table 2-2.		Package
Register Address: D54H, 3412	IA32_L2_QOS_EXT_BW_THRTL_4	
Memory Bandwidth Enforcement for COS4 (R/W) See Table 2-2.		Package
Register Address: D55H, 3413	IA32_L2_QOS_EXT_BW_THRTL_5	
Memory Bandwidth Enforcement for COS5 (R/W) See Table 2-2.		Package
Register Address: D56H, 3414	IA32_L2_QOS_EXT_BW_THRTL_6	
Memory Bandwidth Enforcement for COS6 (R/W) See Table 2-2.		Package
Register Address: D57H, 3415	IA32_L2_QOS_EXT_BW_THRTL_7	
Memory Bandwidth Enforcement for COS7 (R/W) See Table 2-2.		Package
Register Address: D58H, 3416	IA32_L2_QOS_EXT_BW_THRTL_8	
Memory Bandwidth Enforcement for COS8 (R/W) See Table 2-2.		Package
Register Address: D59H, 3417	IA32_L2_QOS_EXT_BW_THRTL_9	
Memory Bandwidth Enforcement for COS9 (R/W) See Table 2-2.		Package
Register Address: D5AH, 3418	IA32_L2_QOS_EXT_BW_THRTL_10	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Memory Bandwidth Enforcement for COS10 (R/W) See Table 2-2.		Package
Register Address: D5BH, 3419	IA32_L2_QOS_EXT_BW_THRTL_11	
Memory Bandwidth Enforcement for COS11 (R/W) See Table 2-2.		Package
Register Address: D5CH, 3420	IA32_L2_QOS_EXT_BW_THRTL_12	
Memory Bandwidth Enforcement for COS12 (R/W) See Table 2-2.		Package
Register Address: D5DH, 3421	IA32_L2_QOS_EXT_BW_THRTL_13	
Memory Bandwidth Enforcement for COS13 (R/W) See Table 2-2.		Package
Register Address: D5EH, 3422	IA32_L2_QOS_EXT_BW_THRTL_14	
Memory Bandwidth Enforcement for COS14 (R/W) See Table 2-2.		Package
Register Address: E00H, 3584	IA32_QOS_CORE_BW_THRTL_0	
CBA Levels Based on COS for Bandwidth Throttling (R/W) See Table 2-2.		Thread
Register Address: E01H, 3585	IA32_QOS_CORE_BW_THRTL_1	
CBA Levels Based on COS for Bandwidth Throttling (R/W) See Table 2-2.		Thread
Register Address: 1400H, 5120	IA32_SEAMRR_BASE	
SEAM Memory Range Register for TDX - Base Address (R/W) See Table 2-2.		Core
Register Address: 1401H, 5121	IA32_SEAMRR_MASK	
SEAM Memory Range Register for TDX (R/W) See Table 2-2.		Core
Register Address: 1A8FH, 6799	MSR_STLB_QOS_INFO	
STLB_QOS_INFO (R/O) STLB QoS MASK configuration.		Core
5:0	NCLOS Number of CLOS supported for STLB resource using minus-1 notation.	
15:6	Reserved.	
19:16	4K_2M_CBM Length of capacity bitmask for 4K and 2M pages using minus-1 notation.	
28:20	Reserved.	
29	STLB_FILL_TRANSLATION_MSR_SUPPORTED MSR interface to fill STLB translations supported.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30	4K_2M_ALIAS Indicates that 4K/2M pages alias into the same structure.	
63:31	Reserved.	

2.17.12 MSRs Introduced in the Intel® Series 2 Core™ Ultra Processor Supporting Performance Hybrid Architecture

Table 2-58 lists additional MSRs for the Intel Series 2 Core Ultra processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_BDH. Table 2-59 lists the MSRs unique to the processor P-core. Table 2-60 lists the MSRs unique to the processor E-core.

For an MSR listed in Table 2-58, Table 2-59, or Table 2-60 that also appears in the model-specific tables of prior generations, Table 2-58, Table 2-59, and Table 2-60 supersede prior generation tables.

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Thread
Register Address: 601H, 1537	MSR_PKG_POWER_LIMIT_4	
Package Power Limit 4 (R/W) Package-level maximum power limit (in Watts).		Package
15:0	POWER_LIMIT_4 PL4 Value in 0.125 W increments. This field is locked by PKG_POWER_LIMIT_4.LOCK. When the LOCK bit is set to 1, this field becomes Read Only. If the value is 0, PL4 limit is disabled.	
30:16	Reserved.	
31	LOCK This bit will lock the POWER_LIMIT_4 settings in this register and will also lock this setting. This means that once set to 1, the POWER_LIMIT_4 setting and this bit become Read Only until the next Warm Reset.	
63:32	Reserved.	
Register Address: 630H, 1584	MSR_PKG_C8_RESIDENCY	
MSR_PKG_C8_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
59:0	Package C8 Residency Counter Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 631H, 1585	MSR_PKG_C9_RESIDENCY	
MSR_PKG_C9_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C9 Residency Counter Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
MSR_PKG_C10_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C10 Residency Counter Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 651H, 1617	MSR_SECONDARY_TURBO_RATIO_LIMIT_CORES	
SECONDARY_TURBO_RATIO_LIMIT_CORES (R/W) This register defines the active core ranges for each frequency point. - NUMCORE[0:7] must be populated in ascending order. - NUMCORE[i+1] must be greater than NUMCORE[i]. - Entries with NUMCORE[i] == 0 will be ignored. - The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, we will silently reject the configuration.		Package
7:0	CORE_COUNT_0 Defines the active core ranges for each frequency point.	
15:8	CORE_COUNT_1 Defines the active core ranges for each frequency point.	
23:16	CORE_COUNT_2 Defines the active core ranges for each frequency point.	
31:24	CORE_COUNT_3 Defines the active core ranges for each frequency point.	
39:32	CORE_COUNT_4 Defines the active core ranges for each frequency point.	
47:40	CORE_COUNT_5 Defines the active core ranges for each frequency point.	
55:48	CORE_COUNT_6 Defines the active core ranges for each frequency point.	

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:56	CORE_COUNT_7 Defines the active core ranges for each frequency point.	
Register Address: 658H, 1624	MSR_WEIGHTED_CORE_CO	
Core-Count Weighted C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 659H, 1625	MSR_ANY_CORE_CO	
Any Core C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 65AH, 1626	MSR_ANY_GFXE_CO	
Any Graphics Engine C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.	
Register Address: 65BH, 1627	MSR_CORE_GFXE_OVERLAP_CO	
Core and Graphics Engine Overlapped C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.	
Register Address: C88H, 3208	IA32_RESOURCE_PRIORITY	
Thread scope Resource Priority Enable (R/W) See Table 2-2.		Thread
Register Address: C89H, 3209	IA32_RESOURCE_PRIORITY_PKG	
IA32_RESOURCE_PRIORITY_PKG (R/W) See Table 2-2.		Package
Register Address: 1900H, 6400	IA32_PMC_GPO_CTR	
Full Width Writable General Performance Counter 0 (R/W) See Table 2-2.		Thread
Register Address: 1901H, 6401	IA32_PMC_GPO_CFG_A	
IA32_PMC_GPO_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 0. See Table 2-2.		Thread

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1904H, 6404	IA32_PMC_GP1_CTR	
Full Width Writable General Performance Counter 1 (R/W) See Table 2-2.		Thread
Register Address: 1905H, 6405	IA32_PMC_GP1_CFG_A	
IA32_PMC_GP1_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 1. See Table 2-2.		Thread
Register Address: 1908H, 6408	IA32_PMC_GP2_CTR	
Full Width Writable General Performance Counter 2 (R/W) See Table 2-2.		Thread
Register Address: 1909H, 6409	IA32_PMC_GP2_CFG_A	
IA32_PMC_GP2_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 2. See Table 2-2.		Thread
Register Address: 190CH, 6412	IA32_PMC_GP3_CTR	
Full Width Writable General Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 190DH, 6413	IA32_PMC_GP3_CFG_A	
IA32_PMC_GP3_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 3. See Table 2-2.		Thread
Register Address: 1910H, 6416	IA32_PMC_GP4_CTR	
Full Width Writable General Performance Counter 4 (R/W) See Table 2-2.		Thread
Register Address: 1911H, 6417	IA32_PMC_GP4_CFG_A	
IA32_PMC_GP4_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 4. See Table 2-2.		Thread
Register Address: 1914H, 6420	IA32_PMC_GP5_CTR	
Full Width Writable General Performance Counter 5 (R/W) See Table 2-2.		Thread
Register Address: 1915H, 6421	IA32_PMC_GP5_CFG_A	
IA32_PMC_GP5_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 5. See Table 2-2.		Thread
Register Address: 1918H, 6424	IA32_PMC_GP6_CTR	
Full Width Writable General Performance Counter 6 (R/W) See Table 2-2.		Thread
Register Address: 1919H, 6425	IA32_PMC_GP6_CFG_A	

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_PMC_GP6_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 6. See Table 2-2.		Thread
Register Address: 191CH, 6428	IA32_PMC_GP7_CTR	
Full Width Writable General Performance Counter 7 (R/W) See Table 2-2.		Thread
Register Address: 191DH, 6429	IA32_PMC_GP7_CFG_A	
IA32_PMC_GP7_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 7. See Table 2-2.		Thread
Register Address: 1980H, 6528	IA32_PMC_FX0_CTR	
IA32_PMC_FX0_CTR (R/W) Fixed-Function Performance Counter 0 - Instructions Retired. See Table 2-2.		Thread
Register Address: 1984H, 6532	IA32_PMC_FX1_CTR	
IA32_PMC_FX1_CTR (R/W) Fixed-Function Performance Counter 1 - Unhalted core clock cycles. See Table 2-2.		Thread
Register Address: 1988H, 6536	IA32_PMC_FX2_CTR	
IA32_PMC_FX2_CTR (R/W) Fixed-Function Performance Counter 2 - Unhalted core reference cycles. See Table 2-2.		Thread

The MSRs listed in Table 2-59 are unique to the Intel Series 2 Core Ultra processor P-core. These MSRs are not supported on the processor E-core.

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C9H, 201	IA32_PMC8	
General Performance Counter 8 (R/W) See Table 2-2.		Thread
Register Address: CAH, 202	IA32_PMC9	
General Performance Counter 9 (R/W) See Table 2-2.		Thread
Register Address: 18EH, 398	IA32_PERFEVTSEL8	
Performance Event Select Register 8 (R/W) See Table 2-2.		Thread
Register Address: 18FH, 399	IA32_PERFEVTSEL9	
Performance Event Select Register 9 (R/W) See Table 2-2.		Thread
Register Address: 30CH, 780	IA32_FIXED_CTR3	

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Fixed-Function Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 329H, 809	MSR_PERF_METRICS	
MSR_PERF_METRICS (R/W) This register provides built-in support for Top-down Micro-architecture Analysis (TMA) metrics. It exposes the four TMA Level 1 metrics where the lower 32 bits are divided into four 8 bit fields, each of which is an integer percentage of the total TOPDOWN.SLOTS (as reported by fixed-function counter 3).		Thread
7:0	RETIRING Percent of utilized by uops that eventually retire (commit).	
15:8	BAD_SPECULATION Percent of Wasted due to incorrect speculation, covering Utilized by uops that do not retire, or Recovery Bubbles (unutilized slots).	
23:16	FRONTEND_BOUND Percent of Unutilized slots where Front-end did not deliver a uop while Back-end is ready.	
31:24	BACKEND_BOUND Percent of Unutilized slots where a uop was not delivered to Back-end due to lack of Back-end resources.	
39:32	MULTI_UOPS Frontend bound.	
47:40	BRANCH_MISPREDICTS Frontend bound.	
55:48	FRONTEND_LATENCY Frontend bound.	
63:56	MEMORY_BOUND Frontend bound.	
Register Address: 4C9H, 1225	IA32_A_PMC8	
Full Width Writable IA32_PMC8 Alias (R/W) See Table 2-2.		Thread
Register Address: 4CAH, 1226	IA32_A_PMC9	
Full Width Writable IA32_PMC9 Alias (R/W) See Table 2-2.		Thread
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Uarch Control (R/W)		Thread
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 540H, 1344	MSR_CORE_UARCH_CTL	
Core Uarch Control (R/W)		Core

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
0	SCRUB_DIS L1 scrubbing disable.	
63:1	Reserved.	
Register Address: 1920H, 6432	IA32_PMC_GP8_CTR	
Full Width Writable General Performance Counter 8 (R/W) See Table 2-2.		Thread
Register Address: 1921H, 6433	IA32_PMC_GP8_CFG_A	
IA32_PMC_GP8_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 8. See Table 2-2.		Thread
Register Address: 1924H, 6436	IA32_PMC_GP9_CTR	
Full Width Writable General Performance Counter 9 (R/W) See Table 2-2.		Thread
Register Address: 1925H, 6437	IA32_PMC_GP9_CFG_A	
IA32_PMC_GP9_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 9. See Table 2-2.		Thread
Register Address: 198CH, 6540	IA32_PMC_FX3_CTR	
IA32_PMC_FX3_CTR (R/W) See Table 2-2.		Thread

The MSRs listed in Table 2-60 are unique to the Intel Series 2 Core Ultra processor E-core. These MSRs are not supported on the processor P-core.

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2DCH, 732	IA32_INTEGRITY_STATUS	
Status Information for Integrity Features (R/O) See Table 2-2.		Thread
Register Address: 30DH, 781	IA32_FIXED_CTR4	
Fixed-Function Performance Counter 4 - Top-down Bad Speculation (R/W) See Table 2-2.		Thread
Register Address: 30EH, 782	IA32_FIXED_CTR5	
Fixed-Function Performance Counter 5 - Top-down Frontend Bound (R/W) See Table 2-2.		Thread
Register Address: 30FH, 783	IA32_FIXED_CTR6	
Fixed-Function Performance Counter 6 - Top-down Retiring (R/W) See Table 2-2.		Thread
Register Address: D18H, 3352	IA32_L2_MASK_8	

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
L2 CAT Mask for COS8 (R/W) See Table 2-2.		Module
Register Address: D19H, 3353	IA32_L2_MASK_9	
L2 CAT Mask for COS9 (R/W) See Table 2-2.		Module
Register Address: D1AH, 3354	IA32_L2_MASK_10	
L2 CAT Mask for COS10 (R/W) See Table 2-2.		Module
Register Address: D1BH, 3355	IA32_L2_MASK_11	
L2 CAT Mask for COS11 (R/W) See Table 2-2.		Module
Register Address: D1CH, 3356	IA32_L2_MASK_12	
L2 CAT Mask for COS12 (R/W) See Table 2-2.		Module
Register Address: D1DH, 3357	IA32_L2_MASK_13	
L2 CAT Mask for COS13 (R/W) See Table 2-2.		Module
Register Address: D1EH, 3358	IA32_L2_MASK_14	
L2 CAT Mask for COS14 (R/W) See Table 2-2.		Module
Register Address: D1FH, 3359	IA32_L2_MASK_15	
L2 CAT Mask for COS15 (R/W) See Table 2-2.		Module
Register Address: 1878H, 6264	MSR_WORK_CONSERVING_CLOS	
Work Conserving CLOS (R/W)		Module
0	WC_VALID WC Valid Bit that indicates WC MSR has been setup. This bit must be set for the WC algorithm to be enabled.	
7:1	Reserved.	
11:8	CLOS_START_PRI1 Starting CLOS range for priority 1.	
15:12	CLOS_END_PRI1 Ending CLOS range for priority 1.	
19:16	CLOS_START_PRI2 Starting CLOS range for priority 2.	
23:20	CLOS_END_PRI2 Ending CLOS range for priority 2.	
27:24	CLOS_START_PRI3 Starting CLOS range for priority 3.	

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:28	CLOS_END_PRI3 Ending CLOS range for priority 3.	
63:32	Reserved.	
Register Address: 1903H, 6403	IA32_PMC_GP0_CFG_C	
IA32_PMC_GP0_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1907H, 6407	IA32_PMC_GP1_CFG_C	
IA32_PMC_GP1_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 190AH, 6410	IA32_PMC_GP2_CFG_B	
IA32_PMC_GP2_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 190BH, 6411	IA32_PMC_GP2_CFG_C	
IA32_PMC_GP2_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 190EH, 6414	IA32_PMC_GP3_CFG_B	
IA32_PMC_GP3_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 190FH, 6415	IA32_PMC_GP3_CFG_C	
IA32_PMC_GP3_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1912H, 6418	IA32_PMC_GP4_CFG_B	
IA32_PMC_GP4_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 1913H, 6419	IA32_PMC_GP4_CFG_C	
IA32_PMC_GP4_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1916H, 6422	IA32_PMC_GP5_CFG_B	
IA32_PMC_GP5_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 1917H, 6423	IA32_PMC_GP5_CFG_C	
IA32_PMC_GP5_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 191AH, 6426	IA32_PMC_GP6_CFG_B	
IA32_PMC_GP6_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 191BH, 6427	IA32_PMC_GP6_CFG_C	
IA32_PMC_GP6_CFG_C (R/W) See Table 2-2.		Thread

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 191EH, 6430	IA32_PMC_GP7_CFG_B	
IA32_PMC_GP7_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 191FH, 6431	IA32_PMC_GP7_CFG_C	
IA32_PMC_GP7_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1982H, 6530	IA32_PMC_FX0_CFG_B	
Fixed-Function Counter Reload Configuration Register (R/W) See Table 2-2.		Thread
Register Address: 1983H, 6531	IA32_PMC_FX0_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 0 (R/W) See Table 2-2.		Thread
Register Address: 1986H, 6534	IA32_PMC_FX1_CFG_B	
Fixed-Function Counter Reload Configuration Register (R/W) See Table 2-2.		Thread
Register Address: 1987H, 6535	IA32_PMC_FX1_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 1 (R/W) See Table 2-2.		Thread
Register Address: 198BH, 6539	IA32_PMC_FX2_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 2 (R/W) See Table 2-2.		Thread
Register Address: 1990H, 6544	IA32_PMC_FX4_CTR	
Fixed-Function Performance Counter 4 - Top-down Bad Speculation (R/W) See Table 2-2.		Thread
Register Address: 1993H, 6547	IA32_PMC_FX4_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 4 (R/W) See Table 2-2.		Thread
Register Address: 1994H, 6548	IA32_PMC_FX5_CTR	
Fixed-Function Performance Counter 5 - Top-down Frontend Bound (R/W) See Table 2-2.		Thread
Register Address: 1997H, 6551	IA32_PMC_FX5_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 5 (R/W) See Table 2-2.		Thread
Register Address: 1998H, 6552	IA32_PMC_FX6_CTR	
Fixed-Function Performance Counter 5 - Top-down Bad Retiring (R/W) See Table 2-2.		Thread
Register Address: 199BH, 6555	IA32_PMC_FX6_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 6 (R/W) See Table 2-2.		Thread

2.18 MSRS IN THE INTEL® XEON PHI™ PROCESSOR 3200/5200/7200 SERIES AND THE INTEL® XEON PHI™ PROCESSOR 7215/7285/7295 SERIES

The Intel® Xeon Phi™ processor 3200, 5200, 7200 series, with a CPUID Signature DisplayFamily_DisplayModel value of 06_57H, supports the MSR interfaces listed in Table 2-61. These processors are based on the Knights Landing microarchitecture. The Intel® Xeon Phi™ processor 7215, 7285, 7295 series, with a CPUID Signature DisplayFamily_DisplayModel value of 06_85H, supports the MSR interfaces listed in Table 2-61 and Table 2-62. These processors are based on the Knights Mill microarchitecture. Some MSRs are shared between a pair of processor cores, and the scope is marked as module.

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination." See Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, "Time-Stamp Counter," and Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O)	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Thread
0	Lock. (R/WL)	
1	Reserved.	
2	Enable VMX outside SMX operation. (R/WL)	
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.		Thread
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W)		Package
2:0	Package C-State Limit (R/W) Specifies the lowest C-state for the package. This feature does not limit the processor core C-state. The power-on default value from bit[2:0] of this register reports the deepest package C-state the processor is capable to support when manufactured. It is recommended that BIOS always read the power-on default value reported from this bit field to determine the supported deepest C-state on the processor and leave it as default without changing it. 000b - C0/C1 (No package C-state support) 001b - C2 010b - C6 (non retention)* 011b - C6 (Retention)* 100b - Reserved 101b - Reserved 110b - Reserved 111b - No package C-state limit. All C-States supported by the processor are available. Note: C6 retention mode provides more power saving than C6 non-retention mode. Limiting the package to C6 non retention mode does prevent the MSR_PKG_C6_RESIDENCY counter (MSR 3F9h) from being incremented.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO registers at MSR_PMG_IO_CAPTURE_BASE[15:0] to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/O) When set, locks bits [15:0] of this register for further writes until the next reset occurs.	
25	Reserved.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Reserved.	
28	C1 State Auto Undemotion Enable (R/W) When set, enables Undemotion from Demoted C1.	
29	PKG C-State Auto Demotion Enable (R/W) When set, enables Package C state demotion.	
63:30	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Capture Base (R/W)		Tile

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
15:0	LVL_2 Base Address (R/W) Microcode will compare IO-read zone to this base address to determine if an MWAIT(C2/3/4) needs to be issued instead of the IO-read. Should be programmed to the chipset Plevel_2 IO address.	
22:16	C-State Range (R/W) The IO-port block size in which IO-redirection will be executed (0-127). Should be programmed based on the number of LVLx registers existing in the chipset.	
63:23	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, the AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 140H, 320	MISC_FEATURE_ENABLES	
MISC_FEATURE_ENABLES		Thread
0	Reserved.	
1	User Mode MONITOR and MWAIT (R/W) If set to 1, the MONITOR and MWAIT instructions do not cause invalid-opcode exceptions when executed with CPL > 0 or in virtual-8086 mode. If MWAIT is executed when CPL > 0 or in virtual-8086 mode, and if EAX indicates a C-state other than C0 or C1, the instruction operates as if EAX indicated the C-state C1.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread
Register Address: 17AH, 378	IA32_MCG_STATUS	
See Table 2-2.		Thread
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
31:0	Bank Support (SMM-RO) One bit per MCA bank. If the bit is set, that bank supports Enhanced MCA (Default all 0; does not support EMCA).	
55:32	Reserved.	
56	Targeted SMI (SMM-RO) Set if targeted SMI is supported.	
57	SMM_CPU_SVRSTR (SMM-RO) Set if SMM SRAM save/restore feature is supported.	
58	SMM_CODE_ACCESS_CHK (SMM-RO) Set if SMM code access check feature is supported.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 186H, 390	IA32_PERFVTSEL0	
Performance Monitoring Event Select Register (R/W) See Table 2-2.		Thread
7:0	Event Select.	
15:8	UMask.	
16	USR.	
17	OS.	
18	Edge.	
19	PC.	
20	INT.	
21	AnyThread.	
22	EN.	
23	INV.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:24	CMASK.	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Package
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Thread
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Module
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Module
0	Thermal Status (R/O)	
1	Thermal Status Log (R/WCO)	
2	PROTCHOT # or FORCEPR# Status (R/O)	
3	PROTCHOT # or FORCEPR# Log (R/WCO)	
4	Critical Temperature Status (R/O)	
5	Critical Temperature Status Log (R/WCO)	
6	Thermal Threshold #1 Status (R/O)	
7	Thermal Threshold #1 Log (R/WCO)	
8	Thermal Threshold #2 Status (R/O)	
9	Thermal Threshold #2 Log (R/WCO)	
10	Power Limitation Status (R/O)	
11	Power Limitation Log (R/WCO)	
15:12	Reserved.	
22:16	Digital Readout (R/O)	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O)	
31	Reading Valid (R/O)	
63:32	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		Thread
0	Fast-Strings Enable	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W)	
6:4	Reserved.	
7	Performance Monitoring Available (R)	
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O)	
12	Processor Event Based Sampling Unavailable (R/O)	
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W)	
18	ENABLE MONITOR FSM (R/W)	
21:19	Reserved.	
22	Limit CPUID Maxval (R/W)	
23	xTPR Message Disable (R/W)	
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	
37:35	Reserved.	
38	Turbo Mode Disable (R/W)	
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R)	
29:24	Target Offset (R/W)	
63:30	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher.	Core
1	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher.	Core
63:2	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
Offcore Response Event Select Register (R/W)		Shared

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Shared
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode for Groups of Cores (R/W)		Package
0	Reserved.	
7:1	Maximum Number of Cores in Group 0 Number active processor cores which operates under the maximum ratio limit for group 0.	Package
15:8	Maximum Ratio Limit for Group 0 Maximum turbo ratio limit when the number of active cores are not more than the group 0 maximum core count.	Package
20:16	Number of Incremental Cores Added to Group 1 Group 1, which includes the specified number of additional cores plus the cores in group 0, operates under the group 1 turbo max ratio limit = "group 0 Max ratio limit" - "group ratio delta for group 1".	Package
23:21	Group Ratio Delta for Group 1 An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.	Package
28:24	Number of Incremental Cores Added to Group 2 Group 2, which includes the specified number of additional cores plus all the cores in group 1, operates under the group 2 turbo max ratio limit = "group 1 Max ratio limit" - "group ratio delta for group 2".	Package
31:29	Group Ratio Delta for Group 2 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.	Package
36:32	Number of Incremental Cores Added to Group 3 Group 3, which includes the specified number of additional cores plus all the cores in group 2, operates under the group 3 turbo max ratio limit = "group 2 Max ratio limit" - "group ratio delta for group 3".	Package
39:37	Group Ratio Delta for Group 3 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.	Package
44:40	Number of Incremental Cores Added to Group 4 Group 4, which includes the specified number of additional cores plus all the cores in group 3, operates under the group 4 turbo max ratio limit = "group 3 Max ratio limit" - "group ratio delta for group 4".	Package
47:45	Group Ratio Delta for Group 4 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.	Package
52:48	Number of Incremental Cores Added to Group 5 Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".	Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:53	Group Ratio Delta for Group 5 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.	Package
60:56	Number of Incremental Cores Added to Group 6 Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".	Package
63:61	Group Ratio Delta for Group 6 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.	Package
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Thread
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
See Table 2-2.		Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W)		Thread
0	LBR Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	BTF Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	
5:2	Reserved.	
6	TR Setting this bit to 1 enables branch trace messages to be sent.	
7	BTS Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	
8	BTINT When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	
9	BTS_OFF_OS When set, BTS or BTM is skipped if CPL = 0.	
10	BTS_OFF_USR When set, BTS or BTM is skipped if CPL > 0.	
11	FREEZE_LBRS_ON_PMI When set, the LBR stack is frozen on a PMI request.	
12	FREEZE_PERFMON_ON_PMI When set, each ENABLE bit of the global counter control MSR are frozen (address 3BFH) on a PMI request.	
13	Reserved.	
14	FREEZE_WHILE_SMM When set, freezes PerfMon and trace messages while in SMM.	
31:15	Reserved.	
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record from Linear IP (R)		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record to Linear IP (R)		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Core
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Core
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Core
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Core
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Core
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Core
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Core
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Core
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Core
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Core
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Core
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Core
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Core
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Core
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Core
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Core
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Core
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Core
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Core
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Core
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Core

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Core
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Core
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Core
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Core
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Core
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Core
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Core
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Core
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Package
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2.		Thread
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2.		Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2.		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2.		Thread
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C3 Residency Counter (R/O)	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
63:0	Package C6 Residency Counter (R/O)	Package
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
63:0	Package C7 Residency Counter (R/O)	Package
Register Address: 3FCH, 1020	MSR_MCO_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Module
63:0	Module C0 Residency Counter (R/O)	
Register Address: 3FDH, 1021	MSR_MC6_RESIDENCY	
63:0	Module C6 Residency Counter (R/O)	Module
Register Address: 3FFH, 1023	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Core
63:0	CORE C6 Residency Counter (R/O)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS."		Core
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS."		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Thread
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Thread
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2.		Thread
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C2 Residency Counter (R/O)	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
PPO RAPL Power Limit Control (R/W) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O) See Table 2-25.		Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
ConfigTDP Level 1 ratio and power level (R/O) See Table 2-25.		Package
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 ratio and power level (R/O) See Table 2-25.		Package
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W) See Table 2-25.		Package
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W) See Table 2-25.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0)	
1	Thermal Status (R0)	
5:2	Reserved.	
6	VR Therm Alert Status (R0)	
7	Reserved.	
8	Electrical Design Point Status (R0)	
63:9	Reserved.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Core
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		Thread
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		Thread
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		Thread
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		Thread
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		Thread
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		Thread
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service Register Bits [31:0] (R/O)		Thread
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service Register Bits [63:32] (R/O)		Thread
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service Register Bits [95:64] (R/O)		Thread
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits [127:96] (R/O)		Thread
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits [159:128] (R/O)		Thread
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits [191:160] (R/O)		Thread
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits [223:192] (R/O)		Thread
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits [255:224] (R/O)		Thread
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits [31:0] (R/O)		Thread
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits [63:32] (R/O)		Thread
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits [95:64] (R/O)		Thread
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits [127:96] (R/O)		Thread
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits [159:128] (R/O)		Thread
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits [191:160] (R/O)		Thread
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits [223:192] (R/O)		Thread
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits [255:224] (R/O)		Thread
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request Register Bits [31:0] (R/O)		Thread
Register Address: 821H, 2081	IA32_X2APIC_IRR1	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Interrupt Request Register Bits [63:32] (R/O)		Thread
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits [95:64] (R/O)		Thread
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits [127:96] (R/O)		Thread
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits [159:128] (R/O)		Thread
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits [191:160] (R/O)		Thread
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits [223:192] (R/O)		Thread
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits [255:224] (R/O)		Thread
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status Register (R/W)		Thread
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		Thread
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		Thread
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		Thread
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		Thread
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Register (R/W)		Thread
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		Thread
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		Thread
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		Thread
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		Thread
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		Thread
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2		Thread

Table 2-62 lists model-specific registers that are supported by the Intel® Xeon Phi™ processor 7215, 7285, 7295 series based on the Knights Mill microarchitecture.

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Configuration (R/W) This MSR is readable only if VMX is enabled, and writeable only if VMX is enabled and in SMM mode, and is used to configure the VMX MSEG base address. See Table 2-2.		Core
Register Address: 480H, 1152	IA32_VMX_BASIC	

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2.		Core
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2.		Core
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)		Core
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-exit Controls (R/O) See Table 2-2.		Core
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-entry Controls (R/O) See Table 2-2.		Core
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2.		Core
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2.		Core
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2.		Core
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2.		Core
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2.		Core
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2.		Core
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Table 2-2.		Core
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.		Core
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLS	

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTL5	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTL5	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.		Core
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTL5	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.		Core
Register Address: 491H, 1169	IA32_VMX_FMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Core

2.19 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 2-63 lists MSRs (architectural and model-specific) that are defined across processor generations based on Intel NetBurst microarchitecture. The processor can be identified by its CPUID signatures of DisplayFamily encoding of 0FH, see Table 2-1.

- MSRs with an "IA32_" prefix are designated as "architectural." This means that the functions of these MSRs and their addresses remain the same for succeeding families of IA-32 processors.
- MSRs with an "MSR_" prefix are model specific with respect to address functionalities. The column "Model Availability" lists the model encoding value(s) within the Pentium 4 and Intel Xeon processor family at the specified register address. The model encoding value of a processor can be queried using CPUID. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 0H, 0	IA32_P5_MC_ADDR		
See Section 2.23, "MSRs in Pentium Processors."		0, 1, 2, 3, 4, 6	Shared
Register Address: 1H, 1	IA32_P5_MC_TYPE		
See Section 2.23, "MSRs in Pentium Processors."		0, 1, 2, 3, 4, 6	Shared
Register Address: 6H, 6	IA32_MONITOR_FILTER_LINE_SIZE		
See Section 11.10.5, "Monitor/Mwait Address Range Determination."		3, 4, 6	Shared
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER		
Time Stamp Counter See Table 2-2.		0, 1, 2, 3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.			
Register Address: 17H, 23	IA32_PLATFORM_ID		
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine “slot” information for the processor and the proper microcode update to load.		0, 1, 2, 3, 4, 6	Shared
Register Address: 1BH, 27	IA32_APIC_BASE		
APIC Location and Status (R/W) See Table 2-2. See Section 13.4.4, “Local APIC Status and Location.”		0, 1, 2, 3, 4, 6	Unique
Register Address: 2AH, 42	MSR_EBC_HARD_POWERON		
Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.		0, 1, 2, 3, 4, 6	Shared
0	Output Tri-state Enabled (R) Indicates whether tri-state output is enabled (1) or disabled (0) as set by the strapping of SMI#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
1	Execute BIST (R) Indicates whether the execution of the BIST is enabled (1) or disabled (0) as set by the strapping of INIT#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
2	In Order Queue Depth (R) Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
3	MCERR# Observation Disabled (R) Indicates whether MCERR# observation is enabled (0) or disabled (1) as determined by the strapping of A9#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
4	BINIT# Observation Enabled (R) Indicates whether BINIT# observation is enabled (0) or disabled (1) as determined by the strapping of A10#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
6:5	APIC Cluster ID (R) Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
7	Bus Park Disable (R) Indicates whether bus park is enabled (0) or disabled (1) as set by the strapping of A15#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
11:8	Reserved.		
13:12	Agent ID (R) Contains the logical agent ID value as set by the strapping of BR[3:0]. The logical ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.		
63:14	Reserved.		
Register Address: 2BH, 43	MSR_EBC_SOFT_POWERON		
Processor Soft Power-On Configuration (R/W) Enables and disables processor features.		0, 1, 2, 3, 4, 6	Shared
0	RCNT/SCNT On Request Encoding Enable (R/W) Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).		
1	Data Error Checking Disable (R/W) Set to disable system data bus parity checking; clear to enable parity checking.		
2	Response Error Checking Disable (R/W) Set to disable (default); clear to enable.		
3	Address/Request Error Checking Disable (R/W) Set to disable (default); clear to enable.		
4	Initiator MCERR# Disable (R/W) Set to disable MCERR# driving for initiator bus requests (default); clear to enable.		
5	Internal MCERR# Disable (R/W) Set to disable MCERR# driving for initiator internal errors (default); clear to enable.		
6	BINIT# Driver Disable (R/W) Set to disable BINIT# driver (default); clear to enable driver.		
63:7	Reserved.		
Register Address: 2CH, 44	MSR_EBC_FREQUENCY_ID		
Processor Frequency Configuration The bit field layout of this MSR varies according to the MODEL value in the CUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2. (R) The field Indicates the current processor frequency configuration.		2,3, 4, 6	Shared
15:0	Reserved.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
18:16	Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6)		
	133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.		
	266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.		
23:19	Reserved.		
31:24	Core Clock Frequency to System Bus Frequency Ratio (R) The processor core clock frequency to system bus frequency ratio observed at the deassertion of the reset pin.		
63:32	Reserved.		
Register Address: 2CH, 44	MSR_EBC_FREQUENCY_ID		
Processor Frequency Configuration (R) The bit field layout of this MSR varies according to the MODEL value of the CPUID version information. This bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding less than 2. Indicates current processor frequency configuration.		0, 1	Shared
20:0	Reserved.		
23:21	Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz All others values reserved.		
63:24	Reserved.		
Register Address: 3AH, 58	IA32_FEATURE_CONTROL		
Control Features in IA-32 Processor (R/W) See Table 2-2. (If CPUID.01H:ECX[5])		3, 4, 6	Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
BIOS Update Trigger Register (W) See Table 2-2.		0, 1, 2, 3, 4, 6	Shared
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID		
BIOS Update Signature ID (R/W) See Table 2-2.		0, 1, 2, 3, 4, 6	Unique
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL		
SMM Monitor Configuration (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: FEH, 254	IA32_MTRRCAP		
MTRR Information See Section 14.11.1, "MTRR Feature Identification."		0, 1, 2, 3, 4, 6	Unique
Register Address: 174H, 372	IA32_SYSENTER_CS		
CS Register Target for CPL 0 Code (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP		
Stack Pointer for CPL 0 Stack (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP		
CPL 0 Code Entry Point (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 179H, 377	IA32_MCG_CAP		
Machine Check Capabilities (R) See Table 2-2 and Section 18.3.1.1, "IA32_MCG_CAP MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 17AH, 378	IA32_MCG_STATUS		
Machine Check Status (R) See Table 2-2 and Section 18.3.1.2, "IA32_MCG_STATUS MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 17BH, 379	IA32_MCG_CTL		
Machine Check Feature Enable (R/W) See Table 2-2 and Section 18.3.1.3, "IA32_MCG_CTL MSR."			
Register Address: 180H, 384	MSR_MCG_RAX		
Machine Check EAX/RAX Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 181H, 385	MSR_MCG_RBX		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Machine Check EBX/RBX Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 182H, 386	MSR_MCG_RCX		
Machine Check ECX/RX Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 183H, 387	MSR_MCG_RDX		
Machine Check EDX/RDX Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 184H, 388	MSR_MCG_RSI		
Machine Check ESI/RSI Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 185H, 389	MSR_MCG_RDI		
Machine Check EDI/RDI Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 186H, 390	MSR_MCG_RBP		
Machine Check EBP/RBP Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 187H, 391	MSR_MCG_RSP		
Machine Check ESP/RSP Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 188H, 392	MSR_MCG_RFLAGS		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Machine Check EFLAGS/RFLAG Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 189H, 393	MSR_MCG_RIP		
Machine Check EIP/RIP Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 18AH, 394	MSR_MCG_MISC		
Machine Check Miscellaneous See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
0	DS When set, the bit indicates that a page assist or page fault occurred during DS normal operation. The processors response is to shut down. The bit is used as an aid for debugging DS handling code. It is the responsibility of the user (BIOS or operating system) to clear this bit for normal operation.		
63:1	Reserved.		
Register Address: 18BH–18FH, 395–399	MSR_MCG_RESERVED1–MSR_MCG_RESERVED5		
Reserved.			
Register Address: 190H, 400	MSR_MCG_R8		
Machine Check R8 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 191H, 401	MSR_MCG_R9		
Machine Check R9D/R9 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 192H, 402	MSR_MCG_R10		
Machine Check R10 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 193H, 403	MSR_MCG_R11		
Machine Check R11 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 194H, 404	MSR_MCG_R12		
Machine Check R12 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 195H, 405	MSR_MCG_R13		
Machine Check R13 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 196H, 406	MSR_MCG_R14		
Machine Check R14 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 197H, 407	MSR_MCG_R15		
Machine Check R15 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 198H, 408	IA32_PERF_STATUS		
See Table 2-2. See Section 17.1, "Enhanced Intel Speedstep® Technology."		3, 4, 6	Unique
Register Address: 199H, 409	IA32_PERF_CTL		
See Table 2-2. See Section 17.1, "Enhanced Intel Speedstep® Technology."		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 19AH, 410	IA32_CLOCK_MODULATION		
Thermal Monitor Control (R/W) See Table 2-2 and Section 17.8.3, “Software Controlled Clock Modulation.”		0, 1, 2, 3, 4, 6	Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT		
Thermal Interrupt Control (R/W) See Section 17.8.2, “Thermal Monitor,” and Table 2-2.		0, 1, 2, 3, 4, 6	Unique
Register Address: 19CH, 412	IA32_THERM_STATUS		
Thermal Monitor Status (R/W) See Section 17.8.2, “Thermal Monitor,” and Table 2-2.		0, 1, 2, 3, 4, 6	Shared
Register Address: 19DH, 413	MSR_THERM2_CTL		
Thermal Monitor 2 Control			
For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.		3	Shared
For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.		4, 6	Shared
Register Address: 1A0H, 416	IA32_MISC_ENABLE		
Enable Miscellaneous Processor Features (R/W)		0, 1, 2, 3, 4, 6	Shared
0	Fast-Strings Enable. See Table 2-2.		
1	Reserved.		
2	x87 FPU Fopcode Compatibility Mode Enable		
3	Thermal Monitor 1 Enable See Section 17.8.2, “Thermal Monitor,” and Table 2-2.		
4	Split-Lock Disable When set, the bit causes an #AC exception to be issued instead of a split-lock cycle. Operating systems that set this bit must align system structures to avoid split-lock scenarios. When the bit is clear (default), normal split-locks are issued to the bus. This debug feature is specific to the Pentium 4 processor.		
5	Reserved.		
6	Third-Level Cache Disable (R/W) When set, the third-level cache is disabled; when clear (default) the third-level cache is enabled. This flag is reserved for processors that do not have a third-level cache. Note that the bit controls only the third-level cache; and only if overall caching is enabled through the CD flag of control register CR0, the page-level cache controls, and/or the MTRRs. See Section 14.5.4, “Disabling and Enabling the L3 Cache.”		
7	Performance Monitoring Available (R) See Table 2-2.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
8	Suppress Lock Enable When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.		
9	Prefetch Queue Disable When set, disables the prefetch queue. When clear (default), enables the prefetch queue.		
10	FERR# Interrupt Reporting Enable (R/W) When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled. When this flag is set and the processor is in the stop-clock state (STPCLK# is asserted), asserting the FERR# pin signals to the processor that an interrupt (such as, INIT#, BINIT#, INTR, NMI, SMI#, or RESET#) is pending and that the processor should return to normal operation to handle the interrupt. This flag does not affect the normal operation of the FERR# pin (to indicate an unmasked floating-point error) when the STPCLK# pin is not asserted.		
11	Branch Trace Storage Unavailable (BTS_UNAVAILABLE) (R) See Table 2-2. When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.		
12	PEBS_UNAVAILABLE: Processor Event Based Sampling Unavailable (R) See Table 2-2. When set, the processor does not support processor event-based sampling (PEBS); when clear, PEBS is supported.		
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.	3	
17:14	Reserved.		
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	3, 4, 6	

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
19	Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line of the 128-byte sector containing currently required data. When set to 0, the processor fetches both cache lines in the sector. Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.		
21:20	Reserved.		
22	Limit CPUID MAXVAL (R/W) See Table 2-2. Setting this can cause unexpected behavior to software that depends on the availability of CPUID leaves greater than 3.	3, 4, 6	
23	xTPR Message Disable (R/W) See Table 2-2.		Shared
24	L1 Data Cache Context Mode (R/W) When set, the L1 data cache is placed in shared mode; when clear (default), the cache is placed in adaptive mode. This bit is only enabled for IA-32 processors that support Intel Hyper-Threading Technology. See Section 14.5.6, "L1 Data Cache Context Mode." When L1 is running in adaptive mode and CR3s are identical, data in L1 is shared across logical processors. Otherwise, L1 is not shared and cache use is competitive. If the Context ID feature flag (ECX[10]) is set to 0 after executing CPUID with EAX = 1, the ability to switch modes is not supported. BIOS must not alter the contents of IA32_MISC_ENABLE[24].		
33:25	Reserved.		
34	XD Bit Disable (R/W) See Table 2-3.		Unique
63:35	Reserved.		
Register Address: 1A1H, 417	MSR_PLATFORM_BRV		
Platform Feature Requirements (R)		3, 4, 6	Shared
17:0	Reserved.		
18	PLATFORM Requirements When set to 1, indicates the processor has specific platform requirements. The details of the platform requirements are listed in the respective data sheets of the processor.		
63:19	Reserved.		
Register Address: 1D7H, 471	MSR_LER_FROM_LIP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 20.13.3, "Last Exception Records."		0, 1, 2, 3, 4, 6	Unique
31:0	From Linear IP Linear address of the last branch instruction.		
63:32	Reserved.		
Register Address: 1D7H, 471	MSR_LER_FROM_LIP		
63:0	From Linear IP Linear address of the last branch instruction (If IA-32e mode is active).		Unique
Register Address: 1D8H, 472	MSR_LER_TO_LIP		
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 20.13.3, "Last Exception Records."		0, 1, 2, 3, 4, 6	Unique
31:0	From Linear IP Linear address of the target of the last branch instruction.		
63:32	Reserved.		
Register Address: 1D8H, 472	MSR_LER_TO_LIP		
63:0	From Linear IP Linear address of the target of the last branch instruction (If IA-32e mode is active).		Unique
Register Address: 1D9H, 473	MSR_DEBUGCTLA		
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 20.13.1, "MSR_DEBUGCTLA MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 1DAH, 474	MSR_LASTBRANCH_TOS		
Last Branch Record Stack TOS (R/O) Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 20.13.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture," and addresses 1DBH-1DEH and 680H-68FH.		0, 1, 2, 3, 4, 6	Unique
Register Address: 1DBH, 475	MSR_LASTBRANCH_0		
Last Branch Record 0 (R/O) One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH. See Section 20.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."		0, 1, 2	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 1DCH, 476	MSR_LASTBRANCH_1		
Last Branch Record 1 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 1DDH, 477	MSR_LASTBRANCH_2		
Last Branch Record 2 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 1DEH, 478	MSR_LASTBRANCH_3		
Last Branch Record 3 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0		
Variable Range Base MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 277H, 631	IA32_PAT		
Page Attribute Table See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE		
Default Memory Types (R/W) See Table 2-2 and Section 14.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		0, 1, 2, 3, 4, 6	Shared
Register Address: 300H, 768	MSR_BPU_COUNTER0		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 301H, 769	MSR_BPU_COUNTER1		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 302H, 770	MSR_BPU_COUNTER2		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 303H, 771	MSR_BPU_COUNTER3		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 304H, 772	MSR_MS_COUNTER0		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 305H, 773	MSR_MS_COUNTER1		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 306H, 774	MSR_MS_COUNTER2		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 307H, 775	MSR_MS_COUNTER3		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 308H, 776	MSR_FLAME_COUNTER0		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 309H, 777	MSR_FLAME_COUNTER1		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30AH, 778	MSR_FLAME_COUNTER2		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30BH, 779	MSR_FLAME_COUNTER3		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30CH, 780	MSR_IQ_COUNTER0		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30DH, 781	MSR_IQ_COUNTER1		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30EH, 782	MSR_IQ_COUNTER2		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30FH, 783	MSR_IQ_COUNTER3		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 310H, 784	MSR_IQ_COUNTER4		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 311H, 785	MSR_IQ_COUNTER5		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 360H, 864	MSR_BPU_CCCR0		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 361H, 865	MSR_BPU_CCCR1		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 362H, 866	MSR_BPU_CCCR2		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 363H, 867	MSR_BPU_CCCR3		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 364H, 868	MSR_MS_CCCR0		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 365H, 869	MSR_MS_CCCR1		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 366H, 870	MSR_MS_CCCR2		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 367H, 871	MSR_MS_CCCR3		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 368H, 872	MSR_FLAME_CCCR0		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 369H, 873	MSR_FLAME_CCCR1		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36AH, 874	MSR_FLAME_CCCR2		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36BH, 875	MSR_FLAME_CCCR3		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36CH, 876	MSR_IQ_CCCR0		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36DH, 877	MSR_IQ_CCCR1		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36EH, 878	MSR_IQ_CCCR2		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36FH, 879	MSR_IQ_CCCR3		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 370H, 880	MSR_IQ_CCCR4		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 371H, 881	MSR_IQ_CCCR5		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A0H, 928	MSR_BSU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A1H, 929	MSR_BSU_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A2H, 930	MSR_FSB_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A3H, 931	MSR_FSB_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A4H, 932	MSR_FIRM_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A5H, 933	MSR_FIRM_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A6H, 934	MSR_FLAME_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A7H, 935	MSR_FLAME_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A8H, 936	MSR_DAC_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A9H, 937	MSR_DAC_ESCR1		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AAH, 938	MSR_MOB_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ABH, 939	MSR_MOB_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ACH, 940	MSR_PMH_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ADH, 941	MSR_PMH_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AEH, 942	MSR_SAAT_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AFH, 943	MSR_SAAT_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BOH, 944	MSR_U2L_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B1H, 945	MSR_U2L_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B2H, 946	MSR_BPU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B3H, 947	MSR_BPU_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B4H, 948	MSR_IS_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B5H, 949	MSR_IS_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B6H, 950	MSR_ITLB_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B7H, 951	MSR_ITLB_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B8H, 952	MSR_CRU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B9H, 953	MSR_CRU_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BAH, 954	MSR_IQ_ESCR0		
See Section 22.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.		0, 1, 2	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 3BBH, 955	MSR_IQ_ESCR1		
See Section 22.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.		0, 1, 2	Shared
Register Address: 3BCH, 956	MSR_RAT_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BDH, 957	MSR_RAT_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BEH, 958	MSR_SSU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3COH, 960	MSR_MS_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C1H, 961	MSR_MS_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C2H, 962	MSR_TBPU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C3H, 963	MSR_TBPU_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C4H, 964	MSR_TC_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C5H, 965	MSR_TC_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C8H, 968	MSR_IX_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C9H, 969	MSR_IX_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CAH, 970	MSR_ALF_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CBH, 971	MSR_ALF_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CCH, 972	MSR_CRU_ESCR2		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CDH, 973	MSR_CRU_ESCR3		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3E0H, 992	MSR_CRU_ESCR4		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3E1H, 993	MSR_CRU_ESCR5		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3F0H, 1008	MSR_TC_PRECISE_EVENT		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)		
Processor Event Based Sampling (PEBS) (R/W) Controls the enabling of processor event sampling and replay tagging.		0, 1, 2, 3, 4, 6	Shared
12:0	See https://perfmon-events.intel.com/ .		
23:13	Reserved.		
24	UOP Tag Enables replay tagging when set.		
25	ENABLE_PEBS_MY_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 22.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is called ENABLE_PEBS in IA-32 processors that do not support Intel Hyper-Threading Technology.		
26	ENABLE_PEBS_OTH_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 22.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is reserved for IA-32 processors that do not support Intel Hyper-Threading Technology.		
63:27	Reserved.		
Register Address: 3F2H, 1010	MSR_PEBS_MATRIX_VERT		
See https://perfmon-events.intel.com/ .		0, 1, 2, 3, 4, 6	Shared
Register Address: 400H, 1024	IA32_MCO_CTL		
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 401H, 1025	IA32_MCO_STATUS		
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 402H, 1026	IA32_MCO_ADDR		
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 403H, 1027	IA32_MCO_MISC		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 18.3.2.4, "IA32_MCi_MISC MSRs." The IA32_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.		0, 1, 2, 3, 4, 6	Shared
Register Address: 404H, 1028	IA32_MC1_CTL		
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 405H, 1029	IA32_MC1_STATUS		
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 406H, 1030	IA32_MC1_ADDR		
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 407H, 1031	IA32_MC1_MISC		
See Section 18.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC1_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			Shared
Register Address: 408H, 1032	IA32_MC2_CTL		
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 409H, 1033	IA32_MC2_STATUS		
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40AH, 1034	IA32_MC2_ADDR		
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 40BH, 1035	IA32_MC2_MISC		
See Section 18.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 40CH, 1036	IA32_MC3_CTL		
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40DH, 1037	IA32_MC3_STATUS		
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40EH, 1038	IA32_MC3_ADDR		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 40FH, 1039	IA32_MC3_MISC		
See Section 18.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC3_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 410H, 1040	IA32_MC4_CTL		
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 411H, 1041	IA32_MC4_STATUS		
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 412H, 1042	IA32_MC4_ADDR		
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 413H, 1043	IA32_MC4_MISC		
See Section 18.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 480H, 1152	IA32_VMX_BASIC		
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		3, 4, 6	Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL		
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		3, 4, 6	Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL		
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL		
Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 485H, 1157	IA32_VMX_MISC		
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data," and Table 2-2.		3, 4, 6	Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0		
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.		3, 4, 6	Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1		
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.		3, 4, 6	Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0		
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and Table 2-2.		3, 4, 6	Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1		
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and Table 2-2.		3, 4, 6	Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM		
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration," and Table 2-2.		3, 4, 6	Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2		
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 600H, 1536	IA32_DS_AREA		
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		0, 1, 2, 3, 4, 6	Unique
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP		
Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (680H-68FH). This part of the stack contains pointers to the source instruction for one of the last 16 branches, exceptions, or interrupts taken by the processor. The MSRs at 680H-68FH, 6C0H-6CfH are not available in processor releases before family 0FH, model 03H. These MSRs replace MSRs previously located at 1DBH-1DEH, which performed the same function for early releases. See Section 20.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."		3, 4, 6	Unique
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP		
Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP		
Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP		
Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP		
Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP		
Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP		
Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP		
Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP		
Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP		
Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP		
Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP		
Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP		
Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP		
Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP		
Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP		
Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 20.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."		3, 4, 6	Unique
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP		
Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP		
Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP		
Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP		
Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP		
Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP		
Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP		
Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP		
Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP		
Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP		
Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP		
Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP		
Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP		
Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP		
Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: C000_0080H	IA32_EFER		
Extended Feature Enables See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0081H	IA32_STAR		
System Call Target Address (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0082H	IA32_LSTAR		
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0084H	IA32_FMASK		
System Call Flag Mask (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0100H	IA32_FS_BASE		
Map of BASE Address of FS (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0101H	IA32_GS_BASE		
Map of BASE Address of GS (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE		
Swap Target of BASE Address of GS (R/W) See Table 2-2.		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
NOTES 1. For HT-enabled processors, there may be more than one logical processors per physical unit. If an MSR is Shared, this means that one MSR is shared between logical processors. If an MSR is unique, this means that each logical processor has its own MSR.			

2.19.1 MSRs Unique to Intel® Xeon® Processor MP with L3 Cache

The MSRs listed in Table 2-64 apply to Intel® Xeon® Processor MP with up to 8MB level three cache. These processors can be detected by enumerating the deterministic cache parameter leaf, CPUID.04H, to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 3 or 4 (see CPUID instruction for more details).

Table 2-64. MSRs Unique to 64-bit Intel® Xeon® Processor MP with Up to an 8 MB L3 Cache

Register Address: Hex	Register Name		
Register Information		Model Availability	Shared/ Unique
Register Address: 107CCH	MSR_IFSB_BUSQ0		
IFSB BUSQ Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107CDH	MSR_IFSB_BUSQ1		
IFSB BUSQ Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107CEH	MSR_IFSB_SNPQ0		
IFSB SNPQ Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107CFH	MSR_IFSB_SNPQ1		
IFSB SNPQ Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107D0H	MSR_EFSB_DRDY0		
EFSB DRDY Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107D1H	MSR_EFSB_DRDY1		
EFSB DRDY Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107D2H	MSR_IFSB_CTL6		
IFSB Latency Event Control Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107D3H	MSR_IFSB_CNTR7		
IFSB Latency Event Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared

The MSRs listed in Table 2-65 apply to Intel® Xeon® Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter, CPUID.04H, to detect the presence of the third level cache, and

with CPUID reporting family encoding 0FH, model encoding 6 (See CPUID instruction for more details.). The performance monitoring MSRs listed in Table 2-65 are shared between logical processors in the same core, but are replicated for each core.

Table 2-65. MSRs Unique to Intel® Xeon® Processor 7100 Series

Register Address: Hex	Register Name		
Register Information		Model Availability	Shared/Unique
Register Address: 107CCH	MSR_EMON_L3_CTR_CTL0		
GBUSQ Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107CDH	MSR_EMON_L3_CTR_CTL1		
GBUSQ Event Control and Counter Register (R/W)		6	Shared
Register Address: 107CEH	MSR_EMON_L3_CTR_CTL2		
GSNPQ Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107CFH	MSR_EMON_L3_CTR_CTL3		
GSNPQ Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D0H	MSR_EMON_L3_CTR_CTL4		
FSB Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107D1H	MSR_EMON_L3_CTR_CTL5		
FSB Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D2H	MSR_EMON_L3_CTR_CTL6		
FSB Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D3H	MSR_EMON_L3_CTR_CTL7		
FSB Event Control and Counter Register (R/W)		6	Shared

2.20 MSRS IN INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Model-specific registers (MSRs) for Intel Core Solo, Intel Core Duo processors, and Dual-core Intel Xeon processor LV are listed in Table 2-66. The column "Shared/Unique" applies to Intel Core Duo processor. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/Unique
Register Address: 0H, 0	P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.		Unique
Register Address: 1H, 1	P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.		Unique

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, "Time-Stamp Counter," and Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.		Shared
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
3	MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
4	Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
6: 5	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
11	Reserved.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
13	Reserved	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes	
15	Reserved.	
17:16	APIC Cluster ID (R/O)	
18	System Bus Frequency (R/O) 0 = 100 MHz. 1 = Reserved.	
19	Reserved.	
21:20	Symmetric Arbitration ID (R/O)	
26:22	Clock Frequency Ratio (R/O)	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in IA-32 Processor (R/W) See Table 2-2.		Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0	
Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the 'to' address. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1	
Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2	
Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3	
Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 44H, 68	MSR_LASTBRANCH_4	
Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 45H, 69	MSR_LASTBRANCH_5	
Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 46H, 70	MSR_LASTBRANCH_6	
Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 47H, 71	MSR_LASTBRANCH_7	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Unique
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the scalable bus clock speed.		Shared
2:0	101B: 100 MHz (FSB 400) 001B: 133 MHz (FSB 533) 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Unique
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Shared
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
0	RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.	
1	EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved	
Register Address: 186H, 390	IA32_PERFVTSELO	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Shared
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2 and Section 17.8.2, "Thermal Monitor."		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2 and Section 17.8.2, "Thermal Monitor".		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Unique
15:0	Reserved.	
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.	
63:16	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
2:0	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
9:8	Reserved.	
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
12	Reserved.	
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2. Setting this bit may cause behavior in software that depends on the availability of CPUID leaves greater than 2.	Shared
33:23	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Shared
63:35	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	MTRRphysBase0	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Memory Type Range Registers		Unique
Register Address: 201H, 513	MTRRphysMask0	
Memory Type Range Registers		Unique
Register Address: 202H, 514	MTRRphysBase1	
Memory Type Range Registers		Unique
Register Address: 203H, 515	MTRRphysMask1	
Memory Type Range Registers		Unique
Register Address: 204H, 516	MTRRphysBase2	
Memory Type Range Registers		Unique
Register Address: 205H, 517	MTRRphysMask2	
Memory Type Range Registers		Unique
Register Address: 206H, 518	MTRRphysBase3	
Memory Type Range Registers		Unique
Register Address: 207H, 519	MTRRphysMask3	
Memory Type Range Registers		Unique
Register Address: 208H, 520	MTRRphysBase4	
Memory Type Range Registers		Unique
Register Address: 209H, 521	MTRRphysMask4	
Memory Type Range Registers		Unique
Register Address: 20AH, 522	MTRRphysBase5	
Memory Type Range Registers		Unique
Register Address: 20BH, 523	MTRRphysMask5	
Memory Type Range Registers		Unique
Register Address: 20CH, 524	MTRRphysBase6	
Memory Type Range Registers		Unique
Register Address: 20DH, 525	MTRRphysMask6	
Memory Type Range Registers		Unique
Register Address: 20EH, 526	MTRRphysBase7	
Memory Type Range Registers		Unique
Register Address: 20FH, 527	MTRRphysMask7	
Memory Type Range Registers		Unique
Register Address: 250H, 592	MTRRfix64K_00000	
Memory Type Range Registers		Unique
Register Address: 258H, 600	MTRRfix16K_80000	
Memory Type Range Registers		Unique
Register Address: 259H, 601	MTRRfix16K_A0000	
Memory Type Range Registers		Unique

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 268H, 616	MTRRfix4K_C0000	
Memory Type Range Registers		Unique
Register Address: 269H, 617	MTRRfix4K_C8000	
Memory Type Range Registers		Unique
Register Address: 26AH, 618	MTRRfix4K_D0000	
Memory Type Range Registers		Unique
Register Address: 26BH, 619	MTRRfix4K_D8000	
Memory Type Range Registers		Unique
Register Address: 26CH, 620	MTRRfix4K_E0000	
Memory Type Range Registers		Unique
Register Address: 26DH, 621	MTRRfix4K_E8000	
Memory Type Range Registers		Unique
Register Address: 26EH, 622	MTRRfix4K_F0000	
Memory Type Range Registers		Unique
Register Address: 26FH, 623	MTRRfix4K_F8000	
Memory Type Range Registers		Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2 and Section 14.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		Unique
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 408H, 1032	IA32_MC2_CTL	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 40CH, 1036	MSR_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 40DH, 1037	MSR_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 40EH, 1038	MSR_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 410H, 1040	IA32_MC3_CTL	
IA32_MC3_CTL	See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 411H, 1041	IA32_MC3_STATUS	
IA32_MC3_STATUS	See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 412H, 1042	MSR_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 413H, 1043	MSR_MC3_MISC	
Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.		Unique
Register Address: 414H, 1044	MSR_MC5_CTL	
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).		Unique
Register Address: 415H, 1045	MSR_MC5_STATUS	
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.		Unique
Register Address: 416H, 1046	MSR_MC5_ADDR	
Machine Check Error Reporting Register - contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCi_STATUS register is set.		Unique
Register Address: 417H, 1047	MSR_MC5_MISC	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISCV flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information." (If CPUID.01H:ECX[5])		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data." (If CPUID.01H:ECX[5])		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0." (If CPUID.01H:ECX[5])		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0." (If CPUID.01H:ECX[5])		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4." (If CPUID.01H:ECX[5])		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4." (If CPUID.01H:ECX[5])		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration." (If CPUID.01H:ECX[5])		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5] and IA32_VMX_PROCBASED_CTL[bit 63])		Unique
Register Address: 600H, 1536	IA32_DS_AREA	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		Unique
31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.	
63:32	Reserved.	
Register Address: C000_0080H	IA32_EFER	
See Table 2-2.		Unique
10:0	Reserved.	
11	Execute Disable Bit Enable	
63:12	Reserved.	

2.21 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section 2.22 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

Table 2-67. MSRs in Pentium M Processors

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 0H, 0	P5_MC_ADDR
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER
See Section 20.17, "Time-Stamp Counter," and see Table 2-2.	
Register Address: 17H, 23	IA32_PLATFORM_ID
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	
Register Address: 2AH, 42	MSR_EBL_CR_POWERON
Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.	
0	Reserved.
1	Data Error Checking Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
2	Response Error Checking Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
3	MCERR# Drive Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
4	Address Parity Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
6:5	Reserved.
7	BINIT# Driver Enable (R) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
11	Reserved.
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
13	Reserved.
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes. Always 0 on the Pentium M processor.
15	Reserved.
17:16	APIC Cluster ID (R/O) Always 00B on the Pentium M processor.
18	System Bus Frequency (R/O) 0 = 100 MHz. 1 = Reserved. Always 0 on the Pentium M processor.
19	Reserved.
21:20	Symmetric Arbitration ID (R/O) Always 00B on the Pentium M processor.
26:22	Clock Frequency Ratio (R/O)
Register Address: 40H, 64	MSR_LASTBRANCH_0

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 41H, 65	MSR_LASTBRANCH_1
Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 42H, 66	MSR_LASTBRANCH_2
Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 43H, 67	MSR_LASTBRANCH_3
Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 44H, 68	MSR_LASTBRANCH_4
Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 45H, 69	MSR_LASTBRANCH_5
Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 46H, 70	MSR_LASTBRANCH_6
Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 47H, 71	MSR_LASTBRANCH_7
Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 119H, 281	MSR_BBL_CR_CTL
Control Register Used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.	
63:0	Reserved.
Register Address: 11EH, 281	MSR_BBL_CR_CTL3
Control Register 3 Used to configure the L2 Cache.	
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
4:1	Reserved.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
5	ECC Check Enable (R/O) This bit enables ECC checking on the cache data bus. ECC is always generated on write cycles. 0 = Disabled (default). 1 = Enabled. For the Pentium M processor, ECC checking on the cache data bus is always enabled.
7:6	Reserved.
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
22:9	Reserved.
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.
63:24	Reserved.
Register Address: 179H, 377	IA32_MCG_CAP
Read-only register that provides information about the machine-check architecture of the processor.	
7:0	Count (R/O) Indicates the number of hardware unit error reporting banks available in the processor.
8	IA32_MCG_CTL Present (R/O) 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
63:9	Reserved.
Register Address: 17AH, 378	IA32_MCG_STATUS
Global Machine Check Status	
0	RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
1	EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
2	MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
63:3	Reserved.
Register Address: 198H, 408	IA32_PERF_STATUS

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
See Table 2-2.	
Register Address: 199H, 409	IA32_PERF_CTL
See Table 2-2.	
Register Address: 19AH, 410	IA32_CLOCK_MODULATION
Clock Modulation (R/W). See Table 2-2 and Section 17.8.3, "Software Controlled Clock Modulation."	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT
Thermal Interrupt Control (R/W) See Table 2-2 and Section 17.8.2, "Thermal Monitor."	
Register Address: 19CH, 412	IA32_THERM_STATUS
Thermal Monitor Status (R/W) See Table 2-2 and Section 17.8.2, "Thermal Monitor."	
Register Address: 19DH, 413	MSR_THERM2_CTL
Thermal Monitor 2 Control	
15:0	Reserved.
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
63:16	Reserved.
Register Address: 1A0H, 416	IA32_MISC_ENABLE
Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.	
2:0	Reserved.
3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
6:4	Reserved.
7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
9:8	Reserved.
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
	Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported
12	Processor Event Based Sampling Unavailable (R/O) 1 = Processor does not support processor event based sampling (PEBS); 0 = PEBS is supported. The Pentium M processor does not support PEBS.
15:13	Reserved.
16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled. On the Pentium M processor, this bit may be configured to be read-only.
22:17	Reserved.
23	xTPR Message Disable (R/W) When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority. The default is processor specific.
63:24	Reserved.
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: - MSR_LASTBRANCH_0_FROM_IP (at 40H). - Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 1D9H, 473	MSR_DEBUGCTLB
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 1DDH, 477	MSR_LER_TO_LIP
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)," and Section 20.16.2, "Last Branch and Last Exception MSRs."	
Register Address: 1DEH, 478	MSR_LER_FROM_LIP

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)," and Section 20.16.2, "Last Branch and Last Exception MSRs."	
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE
Default Memory Types (R/W) Sets the memory type for the regions of physical memory that are not mapped by the MTRRs. See Section 14.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	
Register Address: 400H, 1024	IA32_MCO_CTL
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 401H, 1025	IA32_MCO_STATUS
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 402H, 1026	IA32_MCO_ADDR
See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 404H, 1028	IA32_MC1_CTL
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 405H, 1029	IA32_MC1_STATUS
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 406H, 1030	IA32_MC1_ADDR
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 408H, 1032	IA32_MC2_CTL
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 409H, 1033	IA32_MC2_STATUS
See Chapter 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 40AH, 1034	IA32_MC2_ADDR
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 40CH, 1036	MSR_MC4_CTL
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 40DH, 1037	MSR_MC4_STATUS
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 40EH, 1038	MSR_MC4_ADDR

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 410H, 1040	MSR_MC3_CTL
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."	
Register Address: 411H, 1041	MSR_MC3_STATUS
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."	
Register Address: 412H, 1042	MSR_MC3_ADDR
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 600H, 1536	IA32_DS_AREA
DS Save Area (R/W) See Table 2-2. Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 22.6.3.4, "Debug Store (DS) Mechanism."	
31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.
63:32	Reserved.

2.22 MSRS IN THE P6 FAMILY PROCESSORS

The following MSRs are defined for the P6 family processors. The MSRs in this table that are shaded are available only in the Pentium II and Pentium III processors. Beginning with the Pentium 4 processor, some of the MSRs in this list have been designated as "architectural" and have had their names changed. See Table 2-2 for a list of the architectural MSRs.

Table 2-68. MSRs in the P6 Family Processors

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 0H, 0	P5_MC_ADDR
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 10H, 16	TSC
See Section 20.17, "Time-Stamp Counter."	
Register Address: 17H, 23	IA32_PLATFORM_ID
Platform ID (R) The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	
49:0	Reserved.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
52:50	Platform Id (R) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
56:53	L2 Cache Latency Read.
59:57	Reserved.
60	Clock Frequency Ratio Read.
63:61	Reserved.
Register Address: 1BH, 27	APIC_BASE
Section 13.4.4, "Local APIC Status and Location."	
7:0	Reserved.
8	Boot Strap Processor Indicator Bit 1 = BSP
10:9	Reserved.
11	APIC Global Enable Bit - Permanent till reset 1 = Enabled. 0 = Disabled.
31:12	APIC Base Address.
63:32	Reserved.
Register Address: 2AH, 42	EBL_CR_POWERON
Processor Hard Power-On Configuration (R/W) Enables and disables processor features, and (R) indicates current processor configuration.	
0	Reserved ¹
1	Data Error Checking Enable (R/W) 1 = Enabled. 0 = Disabled.
2	Response Error Checking Enable FRCERR Observation Enable (R/W) 1 = Enabled. 0 = Disabled.
3	AERR# Drive Enable (R/W) 1 = Enabled. 0 = Disabled.
4	BERR# Enable for Initiator Bus Requests (R/W) 1 = Enabled. 0 = Disabled.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
5	Reserved.
6	BERR# Driver Enable for Initiator Internal Errors (R/W) 1 = Enabled. 0 = Disabled.
7	BINIT# Driver Enable (R/W) 1 = Enabled. 0 = Disabled.
8	Output Tri-state Enabled (R) 1 = Enabled. 0 = Disabled.
9	Execute BIST (R) 1 = Enabled. 0 = Disabled.
10	AERR# Observation Enabled (R) 1 = Enabled. 0 = Disabled.
11	Reserved.
12	BINIT# Observation Enabled (R) 1 = Enabled. 0 = Disabled.
13	In Order Queue Depth (R) 1 = 1. 0 = 8.
14	1-MByte Power on Reset Vector (R) 1 = 1MByte. 0 = 4GBytes.
15	FRC Mode Enable (R) 1 = Enabled. 0 = Disabled.
17:16	APIC Cluster ID (R)
19:18	System Bus Frequency (R) 00 = 66MHz. 10 = 100Mhz. 01 = 133MHz. 11 = Reserved.
21:20	Symmetric Arbitration ID (R)
25:22	Clock Frequency Ratio (R)
26	Low Power Mode Enable (R/W)
27	Clock Frequency Ratio
63:28	Reserved. ¹
Register Address: 33H, 51	MSR_TEST_CTRL

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Test Control Register	
29:0	Reserved.
30	Streaming Buffer Disable
31	Disable LOCK# Assertion for split locked access.
Register Address: 79H, 121	BIOS_UPDT_TRIG
BIOS Update Trigger Register.	
Register Address: 88H, 136	BBL_CR_D0[63:0]
Chunk 0 data register D[63:0]: used to write to and read from the L2.	
Register Address: 89H, 137	BBL_CR_D1
Chunk 1 data register D[63:0]: used to write to and read from the L2.	
Register Address: 8AH, 138	BBL_CR_D2
Chunk 2 data register D[63:0]: used to write to and read from the L2.	
Register Address: 8BH, 139	BIOS_SIGN/BBL_CR_D3
BIOS Update Signature Register or Chunk 3 data register D[63:0]. Used to write to and read from the L2 depending on the usage model.	
Register Address: C1H, 193	PerfCtr0 (PERFCTR0)
Performance Counter Register See Table 2-2.	
Register Address: C2H, 194	PerfCtr1 (PERFCTR1)
Performance Counter Register See Table 2-2.	
Register Address: FEH, 254	MTRRcap
Memory Type Range Registers	
Register Address: 116H, 278	BBL_CR_ADDR
Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses.	
2:0	Reserved; set to 0.
31:3	Address bits [35:3].
63:32	Reserved.
Register Address: 118H, 280	BBL_CR_DECC
Data ECC register D[7:0]: used to write ECC and read ECC to/from L2.	
Register Address: 119H, 281	BBL_CR_CTL
Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.	

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
4:0	L2 Command: 01100 = Data Read w/ LRU update (RLU). 01110 = Tag Read w/ Data Read (TRR). 01111 = Tag Inquire (TI). 00010 = L2 Control Register Read (CR). 00011 = L2 Control Register Write (CW). 010 + MESI encode = Tag Write w/ Data Read (TWR). 111 + MESI encode = Tag Write w/ Data Write (TWW). 100 + MESI encode = Tag Write (TW).
6:5	
7	State to L2
9:8	Reserved.
11:10	Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11 Way to L2
13:12	Modified - 11, Exclusive - 10, Shared - 01, Invalid - 00 Way from L2
15:14	State from L2.
16	Reserved.
17	L2 Hit.
18	Reserved.
20:19	User supplied ECC.
21	Processor number: ² Disable = 1. Enable = 0. Reserved.
63:22	Reserved.
Register Address: 11AH, 282	BBL_CR_TRIG
Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.	
Register Address: 11BH, 283	BBL_CR_BUSY
Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY.	
Register Address: 11EH, 286	BBL_CR_CTL3
Control register 3: used to configure the L2 Cache.	
0	L2 Configured (read/write).
4:1	L2 Cache Latency (read/write).
5	ECC Check Enable (read/write).
6	Address Parity Check Enable (read/write).
7	CRTN Parity Check Enable (read/write).
8	L2 Enabled (read/write).

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
10:9	L2 Associativity (read only): 00 = Direct Mapped. 01 = 2 Way. 10 = 4 Way. 11 = Reserved.
12:11	Number of L2 banks (read only).
17:13	Cache size per bank (read/write): 00001 = 256 KBytes. 00010 = 512 KBytes. 00100 = 1 MByte. 01000 = 2 MBytes. 10000 = 4 MBytes.
18	Cache State error checking enable (read/write).
19	Reserved.
22:20	L2 Physical Address Range support: 111 = 64 GBytes. 110 = 32 GBytes. 101 = 16 GBytes. 100 = 8 GBytes. 011 = 4 GBytes. 010 = 2 GBytes. 001 = 1 GByte. 000 = 512 MBytes.
23	L2 Hardware Disable (read only).
24	Reserved.
25	Cache bus fraction (read only).
63:26	Reserved.
Register Address: 174H, 372	SYSENTER_CS_MSR
CS register target for CPL 0 code	
Register Address: 175H, 373	SYSENTER_ESP_MSR
Stack pointer for CPL 0 stack	
Register Address: 176H, 374	SYSENTER_EIP_MSR
CPL 0 code entry point	
Register Address: 179H, 377	MCG_CAP
Machine Check Global Control Register	
Register Address: 17AH, 378	MCG_STATUS
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.	
Register Address: 17BH, 379	MCG_CTL
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).	
Register Address: 186H, 390	PerfEvtSel0 (EVNTSEL0)

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Performance Event Select Register 0 (R/W)	
7:0	Event Select Refer to Performance Counter section for a list of event encodings.
15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
17	OS Controls the counting of events at Privilege level of 0.
18	E Occurrence/Duration Mode Select: 1 = Occurrence. 0 = Duration.
19	PC Enabled the signaling of performance counter overflow via BPO pin.
20	INT Enables the signaling of counter overflow via input to APIC: 1 = Enable. 0 = Disable.
22	ENABLE Enables the counting of performance events in both counters: 1 = Enable. 0 = Disable.
23	INV Inverts the result of the CMASK condition: 1 = Inverted. 0 = Non-Inverted.
31:24	CMASK (Counter Mask)
Register Address: 187H, 391	PerfEvtSel1 (EVNTSEL1)
Performance Event Select for Counter 1 (R/W)	
7:0	Event Select Refer to Performance Counter section for a list of event encodings.
15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
17	OS Controls the counting of events at Privilege level of 0.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
18	E Occurrence/Duration Mode Select: 1 = Occurrence. 0 = Duration.
19	PC Enabled the signaling of performance counter overflow via BP0 pin.
20	INT Enables the signaling of counter overflow via input to APIC. 1 = Enable. 0 = Disable.
23	INV Inverts the result of the CMASK condition. 1 = Inverted. 0 = Non-Inverted.
31:24	CMASK (Counter Mask)
Register Address: 1D9H, 473	DEBUGCTLMR
Enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.	
0	Enable/Disable Last Branch Records
1	Branch Trap Flag
2	Performance Monitoring/Break Point Pins
3	Performance Monitoring/Break Point Pins
4	Performance Monitoring/Break Point Pins
5	Performance Monitoring/Break Point Pins
6	Enable/Disable Execution Trace Messages
31:7	Reserved.
Register Address: 1DBH, 475	LASTBRANCHFROMIP
32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.	
Register Address: 1DCH, 476	LASTBRANCHTOIP
32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.	
Register Address: 1DDH, 477	LASTINTFROMIP
Last INT from IP	
Register Address: 1DEH, 478	LASTINTTOIP
Last INT to IP	
Register Address: 200H, 512	MTRRphysBase0
Memory Type Range Registers	
Register Address: 201H, 513	MTRRphysMask0
Memory Type Range Registers	

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 202H, 514	MTRRphysBase1
Memory Type Range Registers	
Register Address: 203H, 515	MTRRphysMask1
Memory Type Range Registers	
Register Address: 204H, 516	MTRRphysBase2
Memory Type Range Registers	
Register Address: 205H, 517	MTRRphysMask2
Memory Type Range Registers	
Register Address: 206H, 518	MTRRphysBase3
Memory Type Range Registers	
Register Address: 207H, 519	MTRRphysMask3
Memory Type Range Registers	
Register Address: 208H, 520	MTRRphysBase4
Memory Type Range Registers	
Register Address: 209H, 521	MTRRphysMask4
Memory Type Range Registers	
Register Address: 20AH, 522	MTRRphysBase5
Memory Type Range Registers	
Register Address: 20BH, 523	MTRRphysMask5
Memory Type Range Registers	
Register Address: 20CH, 524	MTRRphysBase6
Memory Type Range Registers	
Register Address: 20DH, 525	MTRRphysMask6
Memory Type Range Registers	
Register Address: 20EH, 526	MTRRphysBase7
Memory Type Range Registers	
Register Address: 20FH, 527	MTRRphysMask7
Memory Type Range Registers	
Register Address: 250H, 592	MTRRfix64K_00000
Memory Type Range Registers	
Register Address: 258H, 600	MTRRfix16K_80000
Memory Type Range Registers	
Register Address: 259H, 601	MTRRfix16K_A0000
Memory Type Range Registers	
Register Address: 268H, 616	MTRRfix4K_C0000
Memory Type Range Registers	
Register Address: 269H, 617	MTRRfix4K_C8000

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Memory Type Range Registers	
Register Address: 26AH, 618	MTRRfix4K_D0000
Memory Type Range Registers	
Register Address: 26BH, 619	MTRRfix4K_D8000
Memory Type Range Registers	
Register Address: 26CH, 620	MTRRfix4K_E0000
Memory Type Range Registers	
Register Address: 26DH, 621	MTRRfix4K_E8000
Memory Type Range Registers	
Register Address: 26EH, 622	MTRRfix4K_F0000
Memory Type Range Registers	
Register Address: 26FH, 623	MTRRfix4K_F8000
Memory Type Range Registers	
Register Address: 2FFH, 767	MTRRdefType
Memory Type Range Registers	
2:0	Default memory type
10	Fixed MTRR enable
11	MTRR Enable
Register Address: 400H, 1024	MCO_CTL
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).	
Register Address: 401H, 1025	MCO_STATUS
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.	
15:0	MC_STATUS_MCACOD
31:16	MC_STATUS_MSCOD
57	MC_STATUS_DAM
58	MC_STATUS_ADDRV
59	MC_STATUS_MISCV
60	MC_STATUS_EN. (Note: For MCO_STATUS only, this bit is hardcoded to 1.)
61	MC_STATUS_UC
62	MC_STATUS_O
63	MC_STATUS_V
Register Address: 402H, 1026	MCO_ADDR
Register Address: 403H, 1027	MCO_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 404H, 1028	MC1_CTL

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 405H, 1029	MC1_STATUS
Bit definitions same as MCO_STATUS.	
Register Address: 406H, 1030	MC1_ADDR
Register Address: 407H, 1031	MC1_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 408H, 1032	MC2_CTL
Register Address: 409H, 1033	MC2_STATUS
Bit definitions same as MCO_STATUS.	
Register Address: 40AH, 1034	MC2_ADDR
Register Address: 40BH, 1035	MC2_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 40CH, 1036	MC4_CTL
Register Address: 40DH, 1037	MC4_STATUS
Bit definitions same as MCO_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.	
Register Address: 40EH, 1038	MC4_ADDR
Defined in MCA architecture but not implemented in P6 Family processors.	
Register Address: 40FH, 1039	MC4_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 410H, 1040	MC3_CTL
Register Address: 411H, 1041	MC3_STATUS
Bit definitions same as MCO_STATUS.	
Register Address: 412H, 1042	MC3_ADDR
Register Address: 413H, 1043	MC3_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
NOTES 1. Bit 0 of this register has been redefined several times, and is no longer used in P6 family processors. 2. The processor number feature may be disabled by setting bit 21 of the BBL_CR_CTL MSR (model-specific register address 119h) to "1". Once set, bit 21 of the BBL_CR_CTL may not be cleared. This bit is write-once. The processor number feature will be disabled until the processor is reset. 3. The Pentium III processor will prevent FSB frequency overclocking with a new shutdown mechanism. If the FSB frequency selected is greater than the internal FSB frequency the processor will shutdown. If the FSB selected is less than the internal FSB frequency the BIOS may choose to use bit 11 to implement its own shutdown policy.	

2.23 MSRS IN PENTIUM PROCESSORS

The following MSRs are defined for the Pentium processors. The P5_MC_ADDR, P5_MC_TYPE, and TSC MSRs (named IA32_P5_MC_ADDR, IA32_P5_MC_TYPE, and IA32_TIME_STAMP_COUNTER in the Pentium 4 processor) are architectural; that is, code that accesses these registers will run on Pentium 4 and P6 family processors without generating exceptions (see Section 2.1, "Architectural MSRs"). The CESR, CTR0, and CTR1 MSRs are unique to Pentium processors; code that accesses these registers will generate exceptions on Pentium 4 and P6 family processors.

Table 2-69. MSRs in the Pentium Processor

Register Address: Hex, Decimal	Register Name
Register Information	
Register Address: 0H, 0	P5_MC_ADDR
See Section 18.11.2, "Pentium Processor Machine-Check Exception Handling."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 18.11.2, "Pentium Processor Machine-Check Exception Handling."	
Register Address: 10H, 16	TSC
See Section 20.17, "Time-Stamp Counter."	
Register Address: 11H, 17	CESR
See Section 22.6.9.1, "Control and Event Select Register (CESR)."	
Register Address: 12H, 18	CTR0
Section 22.6.9.3, "Events Counted."	
Register Address: 13H, 19	CTR1
Section 22.6.9.3, "Events Counted."	