

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

Documentation Changes

August 2012

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.



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Intel, the Intel logo, Pentium, Xeon, Intel NetBurst, Intel Core, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo, Intel Core 2 Extreme, Intel Pentium D, Itanium, Intel SpeedStep, MMX, Intel Atom, and VTune are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copyright © 1997-2012 Intel Corporation. All rights reserved.



Contents

Revision History	4
Preface	7
Summary Tables of Changes	8
Documentation Changes	9



Revision History

Revision	Description	Date
-001	<ul style="list-style-type: none">Initial release	November 2002
-002	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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	<ul style="list-style-type: none">Removed Documentation changes 1-17.Added Documentation changes 1-24.	June 2003
-005	<ul style="list-style-type: none">Removed Documentation Changes 1-24.Added Documentation Changes 1-15.	September 2003
-006	<ul style="list-style-type: none">Added Documentation Changes 16- 34.	November 2003
-007	<ul style="list-style-type: none">Updated Documentation changes 14, 16, 17, and 28.Added Documentation Changes 35-45.	January 2004
-008	<ul style="list-style-type: none">Removed Documentation Changes 1-45.Added Documentation Changes 1-5.	March 2004
-009	<ul style="list-style-type: none">Added Documentation Changes 7-27.	May 2004
-010	<ul style="list-style-type: none">Removed Documentation Changes 1-27.Added Documentation Changes 1.	August 2004
-011	<ul style="list-style-type: none">Added Documentation Changes 2-28.	November 2004
-012	<ul style="list-style-type: none">Removed Documentation Changes 1-28.Added Documentation Changes 1-16.	March 2005
-013	<ul style="list-style-type: none">Updated title.There are no Documentation Changes for this revision of the document.	July 2005
-014	<ul style="list-style-type: none">Added Documentation Changes 1-21.	September 2005
-015	<ul style="list-style-type: none">Removed Documentation Changes 1-21.Added Documentation Changes 1-20.	March 9, 2006
-016	<ul style="list-style-type: none">Added Documentation changes 21-23.	March 27, 2006
-017	<ul style="list-style-type: none">Removed Documentation Changes 1-23.Added Documentation Changes 1-36.	September 2006
-018	<ul style="list-style-type: none">Added Documentation Changes 37-42.	October 2006
-019	<ul style="list-style-type: none">Removed Documentation Changes 1-42.Added Documentation Changes 1-19.	March 2007
-020	<ul style="list-style-type: none">Added Documentation Changes 20-27.	May 2007
-021	<ul style="list-style-type: none">Removed Documentation Changes 1-27.Added Documentation Changes 1-6	November 2007
-022	<ul style="list-style-type: none">Removed Documentation Changes 1-6Added Documentation Changes 1-6	August 2008
-023	<ul style="list-style-type: none">Removed Documentation Changes 1-6Added Documentation Changes 1-21	March 2009



Revision	Description	Date
-024	<ul style="list-style-type: none"> Removed Documentation Changes 1-21 Added Documentation Changes 1-16 	June 2009
-025	<ul style="list-style-type: none"> Removed Documentation Changes 1-16 Added Documentation Changes 1-18 	September 2009
-026	<ul style="list-style-type: none"> Removed Documentation Changes 1-18 Added Documentation Changes 1-15 	December 2009
-027	<ul style="list-style-type: none"> Removed Documentation Changes 1-15 Added Documentation Changes 1-24 	March 2010
-028	<ul style="list-style-type: none"> Removed Documentation Changes 1-24 Added Documentation Changes 1-29 	June 2010
-029	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	September 2010
-030	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	January 2011
-031	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	April 2011
-032	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-14 	May 2011
-033	<ul style="list-style-type: none"> Removed Documentation Changes 1-14 Added Documentation Changes 1-38 	October 2011
-034	<ul style="list-style-type: none"> Removed Documentation Changes 1-38 Added Documentation Changes 1-16 	December 2011
-035	<ul style="list-style-type: none"> Removed Documentation Changes 1-16 Added Documentation Changes 1-18 	March 2012
-036	<ul style="list-style-type: none"> Removed Documentation Changes 1-18 Added Documentation Changes 1-17 	May 2012
-037	<ul style="list-style-type: none"> Removed Documentation Changes 1-17 Added Documentation Changes 1-28 	August 2012

§



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-M</i>	253666
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z</i>	253667
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference</i>	326018
<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

Nomenclature

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

Summary Tables of Changes

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

Codes Used in Summary Tables

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

Documentation Changes(Sheet 1 of 2)

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

Documentation Changes(Sheet 2 of 2)

No.	DOCUMENTATION CHANGES
25	Updates to Chapter 34, Volume 3C
26	Updates to Chapter 35, Volume 3C
27	Updates to Appendix B, Volume 3C
28	Updates to Appendix C, Volume 3C

Documentation Changes

1. Updates to Chapter 1, Volume 1

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

...

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® Core™ i7 processor

- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Xeon® processor E5 family
- Intel® Xeon® processor E3-1200 family
- 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 v2 product family
- 3rd generation Intel® Core™ processors
- Next generation Intel® Core™ processors

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

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

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

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

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

The Intel® Atom™ processor family is based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and the Intel® Core™ i5 processor are based on the Intel® microarchitecture code name Nehalem and support Intel 64 architecture.

Processors based on Intel® microarchitecture code name Westmere 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, 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Intel® microarchitecture code name Sandy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Intel® microarchitecture code name Ivy Bridge and support Intel 64 architecture.

The Next Generation Intel® Core™ processors are based on the Intel® microarchitecture code name Haswell and support 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 processors, 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.

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

1.2 OVERVIEW OF VOLUME 1: BASIC ARCHITECTURE

A description of this manual's content follows:

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

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

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

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

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

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

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

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

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

Chapter 10 — Programming with Streaming SIMD Extensions (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 Streaming SIMD Extensions 2 (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 SSE3, SSSE3 and SSE4. Provides an overview of the SSE3 instruction set, Supplemental SSE3, SSE4, and guidelines for writing code that accesses these extensions.

Chapter 13 — Programming with AVX. Provides an overview of the Intel® AVX instruction set and gives guidelines for writing code that accesses the AVX extensions.

Chapter 14 — Input/Output. Describes the processor's I/O mechanism, including I/O port addressing, I/O instructions, and I/O protection mechanisms.

Chapter 15 — 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 x87 FPU Exception Handlers. Describes how to design and write MS-DOS* compatible exception handling facilities for FPU exceptions (includes software and hardware requirements and assembly-language code examples). This appendix also describes general techniques for writing robust FPU exception handlers.

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

...

2. Updates to Chapter 6, Volume 1

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

...

6.3.7 Branch Functions in 64-Bit Mode

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

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

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

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

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

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

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

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

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

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

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

...

3. Updates to Chapter 12, Volume 1

Change bars show changes to Chapter 12 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1: Basic Architecture*.

...

12.5 OVERVIEW OF SSSE3 INSTRUCTIONS

SSSE3 provides 32 instructions to accelerate a variety of multimedia and signal processing applications employing SIMD integer data. See:

- Section 12.6, “SSSE3 Instructions,” provides an introduction to individual SSSE3 instructions.
- *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A & 2B*, provide detailed information on individual instructions.
- Chapter 13, “System Programming for Instruction Set Extensions and Processor Extended States,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*, gives guidelines for integrating SSE/SSE2/SSE3/SSSE3 extensions into an operating-system environment.

...

12.10.4 Packed Blending Instructions

SSE4.1 adds 6 instructions used for blending (BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, PBLENDVB, PBLENDW).

Blending conditionally copies a data element in a source operand to the same element in the destination. SSE4.1 instructions improve blending operations for most field sizes. A single new SSE4.1 instruction can generally replace a sequence of 2 to 4 operations using previous architectures.

The variable blend instructions (BLENDVPS, BLENDVPD, PBLENDW) introduce the use of control bits stored in an implicit XMM register (XMM0). The most significant bit in each field (the sign bit, for 2’s complement integer or floating-point) is used as a selector. See Table 12-3.

...

4. Updates to Chapter 1, Volume 2A

Change bars show changes to Chapter 1 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L*.

...

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

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

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme QX6000 series
- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® 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® Core™i7 processor
- Intel® Core™i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Xeon® processor E5 family

- Intel® Xeon® processor E3-1200 family
- 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 v2 product family
- 3rd generation Intel® Core™ processors
- Next generation Intel® Core™ processors

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

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

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

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

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

The Intel® Atom™ processor family is based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and the Intel® Core™ i5 processor are based on the Intel® microarchitecture code name Nehalem and support Intel 64 architecture.

Processors based on Intel® microarchitecture code name Westmere 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, 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Intel® microarchitecture code name Sandy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Intel® microarchitecture code name Ivy Bridge and support Intel 64 architecture.

The Next Generation Intel® Core™ processors are based on the Intel® microarchitecture code name Haswell and support 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.

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.

...

5. Updates to Chapter 2, Volume 2A

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L*.

...

Table 2-14 Instructions in each Exception Class

Exception Class	Instruction
Type 1	(V)MOVAPD, (V)MOVAPS, (V)MOVDQA, (V)MOVNTDQ, (V)MOVNTDQA, (V)MOVNTPD, (V)MOVNTPS
Type 2	(V)ADDPD, (V)ADDPs, (V)ADDSUBPD, (V)ADDSUBPS, (V)CMPPD, (V)CMPPS, (V)CVTDQ2PS, (V)CVTPD2DQ, (V)CVTPD2PS, (V)CVTTPD2DQ, (V)CVTTPS2DQ, (V)DIVPD, (V)DIVPS, (V)DPPD*, (V)DPPS*, (V)HADDPD, (V)HADDPs, (V)HSUBPD, (V)HSUBPS, (V)MAXPD, (V)MAXPS, (V)MINPD, (V)MINPS, (V)MULPD, (V)MULPS, (V)ROUNDPD, (V)ROUNDPS, (V)SQRTPD, (V)SQRTPS, (V)SUBPD, (V)SUBPS
Type 3	(V)ADDSD, (V)ADDSS, (V)CMPSD, (V)CMPSS, (V)COMISD, (V)COMISS, (V)CVTTPS2PD, (V)CVTSD2SI, (V)CVTSD2SS, (V)CVTTSI2SD, (V)CVTTSI2SS, (V)CVTSS2SD, (V)CVTSS2SI, (V)CVTSS2SI, (V)CVTSS2SI, (V)DIVSD, (V)DIVSS, (V)MAXSD, (V)MAXSS, (V)MINS, (V)MINSS, (V)MULSD, (V)MULSS, (V)ROUNDSD, (V)ROUNDSS, (V)SQRTSD, (V)SQRTSS, (V)SUBSD, (V)SUBSS, (V)UCOMISD, (V)UCOMISS
Type 4	(V)AESDEC, (V)AESDECLAST, (V)AESENC, (V)AESENCLAST, (V)AESIMC, (V)AESKEYGENASSIST, (V)ANDPD, (V)ANDPS, (V)ANDNPD, (V)ANDNPS, (V)BLENDPD, (V)BLENDPS, (V)BLENDVPD, (V)BLENDVPS, (V)LDDQU, (V)MASK-MOVDQU, (V)PTEST, (V)PTESTPS, (V)PTESTPD, (V)MOVSDQU*, (V)MOVSHDUP, (V)MOVSLDUP, (V)MOVUPD*, (V)MOVUPS*, (V)MPSADBW, (V)ORPD, (V)ORPS, (V)PABSB, (V)PABSW, (V)PABSD, (V)PACKSSWB, (V)PACKSSDW, (V)PACKUSWB, (V)PACKUSDW, (V)PADDB, (V)PADDW, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PAD-DUSB, (V)PADDUSW, (V)PALIGNR, (V)PAND, (V)PANDN, (V)PAVGB, (V)PAVGW, (V)PBLENDVB, (V)PBLENDW, (V)PCMP(E/I)STRI/M***, (V)PCMPEQB, (V)PCMPEQW, (V)PCMPEQD, (V)PCMPEQQ, (V)PCMPGTB, (V)PCMPGTW, (V)PCMPGTD, (V)PCMPGTQ, (V)PCLMULQDQ, (V)PHADDW, (V)PHADD, (V)PHADDSD, (V)PHMINPOSUW, (V)PHSUBD, (V)PHSUBW, (V)PHSUBSW, (V)PMADDWD, (V)PMADDUSW, (V)PMAxSB, (V)PMAxSw, (V)PMAxSD, (V)PMAxUB, (V)PMAxUW, (V)PMAxUD, (V)PMINSB, (V)PMINSW, (V)PMINSD, (V)PMINUB, (V)PMINUW, (V)PMINUD, (V)PMULHUW, (V)PMULHRW, (V)PMULHW, (V)PMULLW, (V)PMULLD, (V)PMULUDQ, (V)PMULDQ, (V)POR, (V)PSADBW, (V)PSHUFb, (V)PSHUFd, (V)PSHUFHW, (V)PSHUFLW, (V)PSIGNB, (V)PSIGNW, (V)PSIGND, (V)PSLLW, (V)PSLLD, (V)PSLLQ, (V)PSRAW, (V)PSRAD, (V)PSRLW, (V)PSRLD, (V)PSRLQ, (V)PSUBB, (V)PSUBW, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PUNPCKHBW, (V)PUNPCKHWD, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKLBW, (V)PUNPCKLWD, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PXOR, (V)RCPPS, (V)RSQRTPS, (V)SHUFPD, (V)SHUFPS, (V)UNPCKHPD, (V)UNPCKHPS, (V)UNPCKLPD, (V)UNPCKLPS, (V)XORPD, (V)XORPS
Type 5	(V)CVTDQ2PD, (V)EXTRACTPS, (V)INSERTPS, (V)MOVD, (V)MOVQ, (V)MOVDDUP, (V)MOVLPD, (V)MOVLPS, (V)MOVHPD, (V)MOVHPS, (V)MOVSD, (V)MOVSS, (V)PEXTRB, (V)PEXTRD, (V)PEXTRW, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ, (V)RCPSS, (V)RSQRTSS, (V)PMOVSX/ZX, (V)VDMXCSR*, (V)VTMXCSR
Type 6	VEXTRACTF128, VPERMILPD, VPERMILPS, VPERM2F128, VBROADCASTSS, VBROADCASTSD, VBROADCASTF128, VINSERTF128, VMASKMOVPS**, VMASKMOVPD**
Type 7	(V)MOVLHPS, (V)MOVHLPS, (V)MOVMSKPD, (V)MOVMSKPS, (V)PMOVMskb, (V)PSLLDQ, (V)PSRLDQ, (V)PSLLW, (V)PSLLD, (V)PSLLQ, (V)PSRAW, (V)PSRAD, (V)PSRLW, (V)PSRLD, (V)PSRLQ
Type 8	VZEROALL, VZERoupper

(*) - Additional exception restrictions are present - see the Instruction description for details

(**) - Instruction behavior on alignment check reporting with mask bits of less than all 1s are the same as with mask bits of all 1s, i.e. no alignment checks are performed.

(***) - PCMPSTRI, PCMPSTRM, PCMPSTRI, and PCMPSTRM instructions do not cause #GP if the memory operand is not aligned to 16-Byte boundary.

...

2.4.4 Exceptions Type 4 (>=16 Byte mem arg no alignment, no floating-point exceptions)

Table 2-20 Type 4 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	VEX prefix: If XCRO[2:1] != '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CRO.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CRO.TS[bit 3]=1.
Stack, SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)	X	X	X	X	Legacy SSE: Memory operand is not 16-byte aligned. ¹
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.

NOTES:

1. PCMPSTRI, PCMPSTRM, PCMPSTRI, and PCMPSTRM instructions do not cause #GP if the memory operand is not aligned to 16-Byte boundary.

...

6. Updates to Chapter 3, Volume 2A

Change bars show changes to Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L*.

...

ADC—Add with Carry

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
14 <i>ib</i>	ADC AL, <i>imm8</i>	I	Valid	Valid	Add with carry <i>imm8</i> to AL.
15 <i>iw</i>	ADC AX, <i>imm16</i>	I	Valid	Valid	Add with carry <i>imm16</i> to AX.
15 <i>id</i>	ADC EAX, <i>imm32</i>	I	Valid	Valid	Add with carry <i>imm32</i> to EAX.
REX.W + 15 <i>id</i>	ADC RAX, <i>imm32</i>	I	Valid	N.E.	Add with carry <i>imm32</i> sign extended to 64-bits to RAX.
80 /2 <i>ib</i>	ADC <i>r/m8</i> , <i>imm8</i>	MI	Valid	Valid	Add with carry <i>imm8</i> to <i>r/m8</i> .
REX + 80 /2 <i>ib</i>	ADC <i>r/m8</i> [*] , <i>imm8</i>	MI	Valid	N.E.	Add with carry <i>imm8</i> to <i>r/m8</i> .
81 /2 <i>iw</i>	ADC <i>r/m16</i> , <i>imm16</i>	MI	Valid	Valid	Add with carry <i>imm16</i> to <i>r/m16</i> .
81 /2 <i>id</i>	ADC <i>r/m32</i> , <i>imm32</i>	MI	Valid	Valid	Add with CF <i>imm32</i> to <i>r/m32</i> .
REX.W + 81 /2 <i>id</i>	ADC <i>r/m64</i> , <i>imm32</i>	MI	Valid	N.E.	Add with CF <i>imm32</i> sign extended to 64-bits to <i>r/m64</i> .
83 /2 <i>ib</i>	ADC <i>r/m16</i> , <i>imm8</i>	MI	Valid	Valid	Add with CF sign-extended <i>imm8</i> to <i>r/m16</i> .
83 /2 <i>ib</i>	ADC <i>r/m32</i> , <i>imm8</i>	MI	Valid	Valid	Add with CF sign-extended <i>imm8</i> into <i>r/m32</i> .
REX.W + 83 /2 <i>ib</i>	ADC <i>r/m64</i> , <i>imm8</i>	MI	Valid	N.E.	Add with CF sign-extended <i>imm8</i> into <i>r/m64</i> .
10 /r	ADC <i>r/m8</i> , <i>r8</i>	MR	Valid	Valid	Add with carry byte register to <i>r/m8</i> .
REX + 10 /r	ADC <i>r/m8</i> [*] , <i>r8</i> [*]	MR	Valid	N.E.	Add with carry byte register to <i>r/m64</i> .
11 /r	ADC <i>r/m16</i> , <i>r16</i>	MR	Valid	Valid	Add with carry <i>r16</i> to <i>r/m16</i> .
11 /r	ADC <i>r/m32</i> , <i>r32</i>	MR	Valid	Valid	Add with CF <i>r32</i> to <i>r/m32</i> .
REX.W + 11 /r	ADC <i>r/m64</i> , <i>r64</i>	MR	Valid	N.E.	Add with CF <i>r64</i> to <i>r/m64</i> .
12 /r	ADC <i>r8</i> , <i>r/m8</i>	RM	Valid	Valid	Add with carry <i>r/m8</i> to byte register.
REX + 12 /r	ADC <i>r8</i> [*] , <i>r/m8</i> [*]	RM	Valid	N.E.	Add with carry <i>r/m64</i> to byte register.
13 /r	ADC <i>r16</i> , <i>r/m16</i>	RM	Valid	Valid	Add with carry <i>r/m16</i> to <i>r16</i> .
13 /r	ADC <i>r32</i> , <i>r/m32</i>	RM	Valid	Valid	Add with CF <i>r/m32</i> to <i>r32</i> .
REX.W + 13 /r	ADC <i>r64</i> , <i>r/m64</i>	RM	Valid	N.E.	Add with CF <i>r/m64</i> to <i>r64</i> .

NOTES:

*In 64-bit mode, *r/m8* can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
MR	ModRM:r/m (r, w)	ModRM:reg (r)	NA	NA
MI	ModRM:r/m (r, w)	imm8	NA	NA
I	AL/AX/EAX/RAX	imm8	NA	NA

Description

Adds the destination operand (first operand), the source operand (second operand), and the carry (CF) flag and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) The state of the CF flag represents a carry from a previous addition. When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADC instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a carry in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

The ADC instruction is usually executed as part of a multibyte or multiword addition in which an ADD instruction is followed by an ADC instruction.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

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. See the summary chart at the beginning of this section for encoding data and limits.

Operation

DEST ← DEST + SRC + CF;

Intel C/C++ Compiler Intrinsic Equivalent

ADC: extern unsigned char _addcarry_u8(unsigned char c_in, unsigned char src1, unsigned char src2, unsigned char *sum_out);

ADC: extern unsigned char _addcarry_u16(unsigned char c_in, unsigned short src1, unsigned short src2, unsigned short *sum_out);

ADC: extern unsigned char _addcarry_u32(unsigned char c_in, unsigned int src1, unsigned char int, unsigned int *sum_out);

ADC: extern unsigned char _addcarry_u64(unsigned char c_in, unsigned __int64 src1, unsigned __int64 src2, unsigned __int64 *sum_out);

Flags Affected

The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

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 is used to access memory and it 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 but the destination is not a memory operand.

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 but the destination is not a memory operand.

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 but the destination is not a memory operand.

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 but the destination is not a memory operand.

...

CLFLUSH—Flush Cache Line

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
OF AE /7	CLFLUSH <i>m8</i>	M	Valid	Valid	Flushes cache line containing <i>m8</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (w)	NA	NA	NA

Description

Invalidates the cache line that contains the linear address specified with the source operand from all levels of the processor cache hierarchy (data and instruction). The invalidation is broadcast throughout the cache coherence domain. If, at any level of the cache hierarchy, the line is inconsistent with memory (dirty) it is written to memory before invalidation. The source operand is a byte memory location.

The availability of CLFLUSH is indicated by the presence of the CPUID feature flag CLFSH (bit 19 of the EDX register, see “CPUID—CPU Identification” in this chapter). The aligned cache line size affected is also indicated with the CPUID instruction (bits 8 through 15 of the EBX register when the initial value in the EAX register is 1).

The memory attribute of the page containing the affected line has no effect on the behavior of this instruction. It should be noted that processors are free to speculatively fetch and cache data from system memory regions assigned a memory-type allowing for speculative reads (such as, the WB, WC, and WT memory types). PREFETCH h instructions can be used to provide the processor with hints for this speculative behavior. Because this speculative fetching can occur at any time and is not tied to instruction execution, the CLFLUSH instruction is not ordered with respect to PREFETCH h instructions or any of the speculative fetching mechanisms (that is, data can be speculatively loaded into a cache line just before, during, or after the execution of a CLFLUSH instruction that references the cache line).

CLFLUSH is only ordered by the MFENCE instruction. It is not guaranteed to be ordered by any other fencing or serializing instructions or by another CLFLUSH instruction. For example, software can use an MFENCE instruction to ensure that previous stores are included in the write-back.

The CLFLUSH instruction can be used at all privilege levels and is subject to all permission checking and faults associated with a byte load (and in addition, a CLFLUSH instruction is allowed to flush a linear address in an execute-only segment). Like a load, the CLFLUSH instruction sets the A bit but not the D bit in the page tables.

The CLFLUSH instruction was introduced with the SSE2 extensions; however, because it has its own CPUID feature flag, it can be implemented in IA-32 processors that do not include the SSE2 extensions. Also, detecting the presence of the SSE2 extensions with the CPUID instruction does not guarantee that the CLFLUSH instruction is implemented in the processor.

CLFLUSH operation is the same in non-64-bit modes and 64-bit mode.

Operation

Flush_Cache_Line(SRC);

Intel C/C++ Compiler Intrinsic Equivalents

CLFLUSH: `void _mm_clflush(void const *p)`

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#UD	If CPUID.01H:EDX.CLFSH[bit 19] = 0. If the LOCK prefix is used. If instruction prefix is 66H, F2H or F3H.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#UD	If CPUID.01H:EDX.CLFSH[bit 19] = 0. If the LOCK prefix is used. If instruction prefix is 66H, F2H or F3H.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.
-----------------	-------------------

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) For a page fault.
- #UD If CPUID.01H: EDX.CLFSH[bit 19] = 0.
If the LOCK prefix is used.
If instruction prefix is 66H, F2H or F3H.

...

Table 3-17 Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
<i>Basic CPUID Information</i>		
0H	EAX EBX ECX EDX	Maximum Input Value for Basic CPUID Information (see Table 3-18) "Genu" "ntel" "inel"
01H	EAX EBX ECX EDX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-5) Bits 07-00: Brand Index Bits 15-08: CLFLUSH line size (Value * 8 = cache line size in bytes) Bits 23-16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31-24: Initial APIC ID Feature Information (see Figure and Table 3-20) Feature Information (see Figure 3-7 and Table 3-21) NOTES: * The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. This field is only valid if CPUID.1.EDX.HTT[bit 28]= 1.
02H	EAX EBX ECX EDX	Cache and TLB Information (see Table 3-22) Cache and TLB Information Cache and TLB Information Cache and TLB Information
03H	EAX EBX ECX EDX	Reserved. Reserved. Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.) Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	<p>NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature. See AP-485, <i>Intel Processor Identification and the CPUID Instruction</i> (Order Number 241618) for more information on PSN.</p>
CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLE.BOOT_NT4[bit 22] = 0 (default).	
<i>Deterministic Cache Parameters Leaf</i>	
04H	<p>NOTES: Leaf 04H output depends on the initial value in ECX.* See also: "INPUT EAX = 4: Returns Deterministic Cache Parameters for each level on page 3-166.</p> <p>EAX Bits 04-00: Cache Type Field 0 = Null - No more caches 1 = Data Cache 2 = Instruction Cache 3 = Unified Cache 4-31 = Reserved</p> <p>Bits 07-05: Cache Level (starts at 1) Bit 08: Self Initializing cache level (does not need SW initialization) Bit 09: Fully Associative cache</p> <p>Bits 13-10: Reserved Bits 25-14: Maximum number of addressable IDs for logical processors sharing this cache**, *** Bits 31-26: Maximum number of addressable IDs for processor cores in the physical package**, ****, *****</p> <p>EBX Bits 11-00: L = System Coherency Line Size** Bits 21-12: P = Physical Line partitions** Bits 31-22: W = Ways of associativity**</p> <p>ECX Bits 31-00: S = Number of Sets**</p> <p>EDX Bit 0: Write-Back Invalidate/Invalidate 0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache. Bit 1: Cache Inclusiveness 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 2: Complex Cache Indexing 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31-03: Reserved = 0</p>

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	<p>NOTES:</p> <p>* If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0. Invalid sub-leaves of EAX = 04H: ECX = n, n > 3.</p> <p>** Add one to the return value to get the result.</p> <p>***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</p> <p>**** 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.</p> <p>***** The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.</p>	
<i>MONITOR/MWAIT Leaf</i>		
05H	EAX	Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	EBX	Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	ECX	Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled Bits 31 - 02: Reserved
	EDX	Bits 03 - 00: Number of C0* sub C-states supported using MWAIT Bits 07 - 04: Number of C1* sub C-states supported using MWAIT Bits 11 - 08: Number of C2* sub C-states supported using MWAIT Bits 15 - 12: Number of C3* sub C-states supported using MWAIT Bits 19 - 16: Number of C4* sub C-states supported using MWAIT Bits 31 - 20: Reserved = 0 NOTE: * The definition of C0 through C4 states for MWAIT extension are processor-specific C-states, not ACPI C-states.
<i>Thermal and Power Management Leaf</i>		
06H	EAX	Bit 00: Digital temperature sensor is supported if set Bit 01: Intel Turbo Boost Technology Available (see description of IA32_MISC_ENABLE[38]). Bit 02: ARAT. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved Bit 04: PLN. Power limit notification controls are supported if set. Bit 05: ECMD. Clock modulation duty cycle extension is supported if set. Bit 06: PTM. Package thermal management is supported if set. Bits 31 - 07: Reserved Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor
	EBX	Bits 31 - 04: Reserved

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of expected processor performance at frequency specified in CPUID Brand String Bits 02 - 01: Reserved = 0 Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H) Bits 31 - 04: Reserved = 0
	EDX	Reserved = 0
<i>Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)</i>		
07H	Sub-leaf 0 (Input ECX = 0). * EAX Bits 31:00: Reports the maximum input value for supported leaf 7 sub-leaves. EBX Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 06: Reserved Bit 07: SMEP. Supports Supervisor Mode Execution Protection if 1. Bit 08: Reserved Bit 09: Supports Enhanced REP MOVSB/STOSB if 1. Bit 10: INVPCID. If 1, supports INVPCID instruction for system software that manages process-context identifiers. Bit 31:11: Reserved ECX Reserved EDX Reserved NOTE: * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Invalid sub-leaves of EAX = 07H: ECX = n, n > 0.	
<i>Direct Cache Access Information Leaf</i>		
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H) Reserved
	EBX	Reserved
	ECX	Reserved
	EDX	
<i>Architectural Performance Monitoring Leaf</i>		
0AH	EAX	Bits 07 - 00: Version ID of architectural performance monitoring Bits 15- 08: Number of general-purpose performance monitoring counter per logical processor Bits 23 - 16: Bit width of general-purpose, performance monitoring counter Bits 31 - 24: Length of EBX bit vector to enumerate architectural performance monitoring events

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	<p>EBX Bit 00: Core cycle event not available if 1 Bit 01: Instruction retired event not available if 1 Bit 02: Reference cycles event not available if 1 Bit 03: Last-level cache reference event not available if 1 Bit 04: Last-level cache misses event not available if 1 Bit 05: Branch instruction retired event not available if 1 Bit 06: Branch mispredict retired event not available if 1 Bits 31- 07: Reserved = 0</p> <p>ECX Reserved = 0</p> <p>EDX Bits 04 - 00: Number of fixed-function performance counters (if Version ID > 1) Bits 12- 05: Bit width of fixed-function performance counters (if Version ID > 1) Reserved = 0</p>
<i>Extended Topology Enumeration Leaf</i>	
OBH	<p>NOTES: Most of Leaf 0BH output depends on the initial value in ECX. EDX output do not vary with initial value in ECX. ECX[7:0] output always reflect initial value in ECX. If ECX contains an invalid sub-leaf index, EAX/EBX/EDX return 0; ECX returns same ECX input. Invalid sub-leaves of EAX = 0BH: ECX = n, n > 1. Leaf 0BH exists if EBX[15:0] is not zero.</p> <p>EAX Bits 04-00: Number of bits to shift right on x2APIC ID to get a unique topology ID of the next level type*. All logical processors with the same next level ID share current level. Bits 31-05: Reserved.</p> <p>EBX Bits 15 - 00: Number of logical processors at this level type. The number reflects configuration as shipped by Intel**. Bits 31- 16: Reserved.</p> <p>ECX Bits 07 - 00: Level number. Same value in ECX input Bits 15 - 08: Level type***. Bits 31 - 16:: Reserved.</p> <p>EDX Bits 31- 00: x2APIC ID the current logical processor.</p> <p>NOTES: * Software should use this field (EAX[4:0]) to enumerate processor topology of the system.</p> <p>** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.</p> <p>*** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0 : invalid 1 : SMT 2 : Core 3-255 : Reserved</p>

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
<i>Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)</i>		
0DH		<p>NOTES: Leaf 0DH main leaf (ECX = 0).</p> <p>EAX Bits 31-00: Reports the valid bit fields of the lower 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved. Bit 00: legacy x87 Bit 01: 128-bit SSE Bit 02: 256-bit AVX Bits 31- 03: Reserved</p> <p>EBX Bits 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.</p> <p>ECX Bit 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor, i.e all the valid bit fields in XCR0.</p> <p>EDX Bit 31-00: Reports the valid bit fields of the upper 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved.</p>
<i>Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)</i>		
0DH	EAX	Bits 31-01: Reserved Bit 00: XSAVEOPT is available;
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
<i>Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)</i>		
0DH		<p>NOTES: Leaf 0DH output depends on the initial value in ECX. Each valid sub-leaf index maps to a valid bit in the XCR0 register starting at bit position 2 * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Invalid sub-leaves of EAX = 0DH: ECX = n, n > 2.</p> <p>EAX Bits 31-0: The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index, <i>n</i>. This field reports 0 if the sub-leaf index, <i>n</i>, is invalid*.</p> <p>EBX Bits 31-0: The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, <i>n</i>, is invalid*.</p> <p>ECX This field reports 0 if the sub-leaf index, <i>n</i>, is invalid*; otherwise it is reserved.</p> <p>EDX This field reports 0 if the sub-leaf index, <i>n</i>, is invalid*; otherwise it is reserved.</p>
<i>Unimplemented CPUID Leaf Functions</i>		
4000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
<i>Extended Function CPUID Information</i>		

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (see Table 3-18).
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved
	ECX	Bit 00: LAHF/SAHF available in 64-bit mode Bits 31-01 Reserved
	EDX	Bits 10-00: Reserved Bit 11: SYSCALL/SYSRET available in 64-bit mode Bits 19-12: Reserved = 0 Bit 20: Execute Disable Bit available Bits 25-21: Reserved = 0 Bit 26: 1-GByte pages are available if 1 Bit 27: RDTSCP and IA32_TSC_AUX are available if 1 Bits 28: Reserved = 0 Bit 29: Intel® 64 Architecture available if 1 Bits 31-30: Reserved = 0
80000002H	EAX	Processor Brand String
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
80000003H	EAX	Processor Brand String Continued
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
80000004H	EAX	Processor Brand String Continued
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
80000005H	EAX	Reserved = 0
	EBX	Reserved = 0
	ECX	Reserved = 0
	EDX	Reserved = 0
80000006H	EAX	Reserved = 0
	EBX	Reserved = 0
	ECX	Bits 07-00: Cache Line size in bytes Bits 11-08: Reserved Bits 15-12: L2 Associativity field * Bits 31-16: Cache size in 1K units
	EDX	Reserved = 0

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	<p>NOTES:</p> <p>* L2 associativity field encodings: 00H - Disabled 01H - Direct mapped 02H - 2-way 04H - 4-way 06H - 8-way 08H - 16-way 0FH - Fully associative</p>	
80000007H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Bits 07-00: Reserved = 0 Bit 08: Invariant TSC available if 1 Bits 31-09: Reserved = 0
80000008H	EAX EBX ECX EDX	Linear/Physical Address size Bits 07-00: #Physical Address Bits* Bits 15-8: #Linear Address Bits Bits 31-16: Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0 <p>NOTES:</p> <p>* If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.</p>

...

CRC32 – Accumulate CRC32 Value

Opcode/ Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F2 0F 38 F0 /r CRC32 r32, r/m8	RM	Valid	Valid	Accumulate CRC32 on r/m8.
F2 REX 0F 38 F0 /r CRC32 r32, r/m8*	RM	Valid	N.E.	Accumulate CRC32 on r/m8.
F2 0F 38 F1 /r CRC32 r32, r/m16	RM	Valid	Valid	Accumulate CRC32 on r/m16.
F2 0F 38 F1 /r CRC32 r32, r/m32	RM	Valid	Valid	Accumulate CRC32 on r/m32.
F2 REX.W 0F 38 F0 /r CRC32 r64, r/m8	RM	Valid	N.E.	Accumulate CRC32 on r/m8.
F2 REX.W 0F 38 F1 /r CRC32 r64, r/m64	RM	Valid	N.E.	Accumulate CRC32 on r/m64.

NOTES:

*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

Description

Starting with an initial value in the first operand (destination operand), accumulates a CRC32 (polynomial 0x11EDC6F41) value for the second operand (source operand) and stores the result in the destination operand. The source operand can be a register or a memory location. The destination operand must be an r32 or r64 register. If the destination is an r64 register, then the 32-bit result is stored in the least significant double word and 00000000H is stored in the most significant double word of the r64 register.

The initial value supplied in the destination operand is a double word integer stored in the r32 register or the least significant double word of the r64 register. To incrementally accumulate a CRC32 value, software retains the result of the previous CRC32 operation in the destination operand, then executes the CRC32 instruction again with new input data in the source operand. Data contained in the source operand is processed in reflected bit order. This means that the most significant bit of the source operand is treated as the least significant bit of the quotient, and so on, for all the bits of the source operand. Likewise, the result of the CRC operation is stored in the destination operand in reflected bit order. This means that the most significant bit of the resulting CRC (bit 31) is stored in the least significant bit of the destination operand (bit 0), and so on, for all the bits of the CRC.

Operation

Notes:

```

BIT_REFLECT64: DST[63-0] = SRC[0-63]
BIT_REFLECT32: DST[31-0] = SRC[0-31]
BIT_REFLECT16: DST[15-0] = SRC[0-15]
BIT_REFLECT8: DST[7-0] = SRC[0-7]

```

MOD2: Remainder from Polynomial division modulus 2

CRC32 instruction for 64-bit source operand and 64-bit destination operand:

```
TEMP1[63-0] ← BIT_REFLECT64 (SRC[63-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[95-0] ← TEMP1[63-0] « 32
TEMP4[95-0] ← TEMP2[31-0] « 64
TEMP5[95-0] ← TEMP3[95-0] XOR TEMP4[95-0]
TEMP6[31-0] ← TEMP5[95-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
DEST[63-32] ← 00000000H
```

CRC32 instruction for 32-bit source operand and 32-bit destination operand:

```
TEMP1[31-0] ← BIT_REFLECT32 (SRC[31-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[63-0] ← TEMP1[31-0] « 32
TEMP4[63-0] ← TEMP2[31-0] « 32
TEMP5[63-0] ← TEMP3[63-0] XOR TEMP4[63-0]
TEMP6[31-0] ← TEMP5[63-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
```

CRC32 instruction for 16-bit source operand and 32-bit destination operand:

```
TEMP1[15-0] ← BIT_REFLECT16 (SRC[15-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[47-0] ← TEMP1[15-0] « 32
TEMP4[47-0] ← TEMP2[31-0] « 16
TEMP5[47-0] ← TEMP3[47-0] XOR TEMP4[47-0]
TEMP6[31-0] ← TEMP5[47-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
```

CRC32 instruction for 8-bit source operand and 64-bit destination operand:

```
TEMP1[7-0] ← BIT_REFLECT8(SRC[7-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[39-0] ← TEMP1[7-0] « 32
TEMP4[39-0] ← TEMP2[31-0] « 8
TEMP5[39-0] ← TEMP3[39-0] XOR TEMP4[39-0]
TEMP6[31-0] ← TEMP5[39-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
DEST[63-32] ← 00000000H
```

CRC32 instruction for 8-bit source operand and 32-bit destination operand:

```
TEMP1[7-0] ← BIT_REFLECT8(SRC[7-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[39-0] ← TEMP1[7-0] « 32
TEMP4[39-0] ← TEMP2[31-0] « 8
TEMP5[39-0] ← TEMP3[39-0] XOR TEMP4[39-0]
TEMP6[31-0] ← TEMP5[39-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
```


Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

unsigned int _mm_crc32_u8(unsigned int crc, unsigned char data)
unsigned int _mm_crc32_u16(unsigned int crc, unsigned short data)
unsigned int _mm_crc32_u32(unsigned int crc, unsigned int data)
unsigned __int64 _mm_crc32_u64(unsigned __int64 crc, unsigned __int64 data)

SIMD Floating Point Exceptions

None

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
#SS(0) If a memory operand effective address is outside the SS segment limit.
#PF (fault-code) For a page fault.
#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD If CPUID.01H: ECX.SSE4_2 [Bit 20] = 0.
If LOCK prefix is used.

Real-Address Mode Exceptions

#GP(0) If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0) If a memory operand effective address is outside the SS segment limit.
#UD If CPUID.01H: ECX.SSE4_2 [Bit 20] = 0.
If LOCK prefix is used.

Virtual 8086 Mode Exceptions

#GP(0) If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0) If a memory operand effective address is outside the SS segment limit.
#PF (fault-code) For a page fault.
#AC(0) If alignment checking is enabled and an unaligned memory reference is made.
#UD If CPUID.01H: ECX.SSE4_2 [Bit 20] = 0.
If LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0) If the memory address is in a non-canonical form.
#SS(0) If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code) For a page fault.
#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD If CPUID.01H: ECX.SSE4_2 [Bit 20] = 0.

If LOCK prefix is used.

...

DPPD – Dot Product of Packed Double Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 41 /r ib DPPD <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE4_1	Selectively multiply packed DP floating-point values from <i>xmm1</i> with packed DP floating-point values from <i>xmm2</i> , add and selectively store the packed DP floating-point values to <i>xmm1</i> .
VEX.NDS.128.66.0F3A.WIG 41 /r ib VDPPD <i>xmm1,xmm2, xmm3/m128, imm8</i>	RVMI	V/V	AVX	Selectively multiply packed DP floating-point values from <i>xmm2</i> with packed DP floating-point values from <i>xmm3</i> , add and selectively store the packed DP floating-point values to <i>xmm1</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RMI	ModRM:reg (r, w)	ModRM:r/m (r)	imm8	NA
RVMI	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	imm8

Description

Conditionally multiplies the packed double-precision floating-point values in the destination operand (first operand) with the packed double-precision floating-point values in the source (second operand) depending on a mask extracted from bits [5:4] of the immediate operand (third operand). If a condition mask bit is zero, the corresponding multiplication is replaced by a value of 0.0.

The two resulting double-precision values are summed into an intermediate result. The intermediate result is conditionally broadcasted to the destination using a broadcast mask specified by bits [1:0] of the immediate byte.

If a broadcast mask bit is "1", the intermediate result is copied to the corresponding qword element in the destination operand. If a broadcast mask bit is zero, the corresponding element in the destination is set to zero.

DPPD follows the NaN forwarding rules stated in the Software Developer's Manual, vol. 1, table 4.7. These rules do not cover horizontal prioritization of NaNs. Horizontal propagation of NaNs to the destination and the positioning of those NaNs in the destination is implementation dependent. NaNs on the input sources or computationally generated NaNs will have at least one NaN propagated to the destination.

128-bit Legacy SSE version: The second source can be an XMM register or a 128-bit memory location. The destination is not distinct from the first source XMM register and the upper bits (VLMAX-1:128) of the corresponding YMM register destination are unmodified.

VEX.128 encoded version: the first source operand is an XMM register or 128-bit memory location. The destination operand is an XMM register. The upper bits (VLMAX-1:128) of the corresponding YMM register destination are zeroed.

If VDPPD is encoded with VEX.L= 1, an attempt to execute the instruction encoded with VEX.L= 1 will cause an #UD exception.

...

DPPS — Dot Product of Packed Single Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 40 /r ib DPPS <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE4_1	Selectively multiply packed SP floating-point values from <i>xmm1</i> with packed SP floating-point values from <i>xmm2</i> , add and selectively store the packed SP floating-point values or zero values to <i>xmm1</i> .
VEX.NDS.128.66.0F3A.WIG 40 /r ib VDPPS <i>xmm1, xmm2, xmm3/m128, imm8</i>	RVMI	V/V	AVX	Multiply packed SP floating point values from <i>xmm1</i> with packed SP floating point values from <i>xmm2/mem</i> selectively add and store to <i>xmm1</i> .
VEX.NDS.256.66.0F3A.WIG 40 /r ib VDPPS <i>ymm1, ymm2, ymm3/m256, imm8</i>	RVMI	V/V	AVX	Multiply packed single-precision floating-point values from <i>ymm2</i> with packed SP floating point values from <i>ymm3/mem</i> , selectively add pairs of elements and store to <i>ymm1</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RMI	ModRM:reg (r, w)	ModRM:r/m (r)	imm8	NA
RVMI	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	imm8

Description

Conditionally multiplies the packed single precision floating-point values in the destination operand (first operand) with the packed single-precision floats in the source (second operand) depending on a mask extracted from the high 4 bits of the immediate byte (third operand). If a condition mask bit in `Imm8[7:4]` is zero, the corresponding multiplication is replaced by a value of 0.0.

The four resulting single-precision values are summed into an intermediate result. The intermediate result is conditionally broadcasted to the destination using a broadcast mask specified by bits `[3:0]` of the immediate byte.

If a broadcast mask bit is "1", the intermediate result is copied to the corresponding dword element in the destination operand. If a broadcast mask bit is zero, the corresponding element in the destination is set to zero.

DPPS follows the NaN forwarding rules stated in the Software Developer's Manual, vol. 1, table 4.7. These rules do not cover horizontal prioritization of NaNs. Horizontal propagation of NaNs to the destination and the positioning of those NaNs in the destination is implementation dependent. NaNs on the input sources or computationally generated NaNs will have at least one NaN propagated to the destination.

128-bit Legacy SSE version: The second source can be an XMM register or an 128-bit memory location. The destination is not distinct from the first source XMM register and the upper bits (`VLMAX-1:128`) of the corresponding YMM register destination are unmodified.

VEX.128 encoded version: the first source operand is an XMM register or 128-bit memory location. The destination operand is an XMM register. The upper bits (`VLMAX-1:128`) of the corresponding YMM register destination are zeroed.

VEX.256 encoded version: The first source operand is a YMM register. The second source operand can be a YMM register or a 256-bit memory location. The destination operand is a YMM register.

Operation

DP_primitive (SRC1, SRC2)

```
IF (imm8[4] = 1)
    THEN Temp1[31:0] ← DEST[31:0] * SRC[31:0];
    ELSE Temp1[31:0] ← +0.0; FI;
IF (imm8[5] = 1)
    THEN Temp1[63:32] ← DEST[63:32] * SRC[63:32];
    ELSE Temp1[63:32] ← +0.0; FI;
IF (imm8[6] = 1)
    THEN Temp1[95:64] ← DEST[95:64] * SRC[95:64];
    ELSE Temp1[95:64] ← +0.0; FI;
IF (imm8[7] = 1)
    THEN Temp1[127:96] ← DEST[127:96] * SRC[127:96];
    ELSE Temp1[127:96] ← +0.0; FI;
```

```
Temp2[31:0] ← Temp1[31:0] + Temp1[63:32];
Temp3[31:0] ← Temp1[95:64] + Temp1[127:96];
Temp4[31:0] ← Temp2[31:0] + Temp3[31:0];
```

```
IF (imm8[0] = 1)
    THEN DEST[31:0] ← Temp4[31:0];
    ELSE DEST[31:0] ← +0.0; FI;
IF (imm8[1] = 1)
    THEN DEST[63:32] ← Temp4[31:0];
    ELSE DEST[63:32] ← +0.0; FI;
IF (imm8[2] = 1)
    THEN DEST[95:64] ← Temp4[31:0];
    ELSE DEST[95:64] ← +0.0; FI;
IF (imm8[3] = 1)
    THEN DEST[127:96] ← Temp4[31:0];
    ELSE DEST[127:96] ← +0.0; FI;
```

DPPS (128-bit Legacy SSE version)

```
DEST[127:0] ← DP_Primitive(SRC1[127:0], SRC2[127:0]);
DEST[VLMAX-1:128] (Unmodified)
```

VDPPS (VEX.128 encoded version)

```
DEST[127:0] ← DP_Primitive(SRC1[127:0], SRC2[127:0]);
DEST[VLMAX-1:128] ← 0
```

VDPPS (VEX.256 encoded version)

```
DEST[127:0] ← DP_Primitive(SRC1[127:0], SRC2[127:0]);
DEST[255:128] ← DP_Primitive(SRC1[255:128], SRC2[255:128]);
```

Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

(V)DPPS: `__m128 __mm_dp_ps (__m128 a, __m128 b, const int mask);`

VDPPS: `__m256 __mm256_dp_ps (__m256 a, __m256 b, const int mask);`

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Exceptions are determined separately for each add and multiply operation, in the order of their execution. Unmasked exceptions will leave the destination operands unchanged.

Other Exceptions

See Exceptions Type 2.

...

FSIN—Sine

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 FE	FSIN	Valid	Valid	Replace ST(0) with its sine.

Description

Computes the sine of the source operand in register ST(0) and stores the result in ST(0). The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$. The following table shows the results obtained when taking the sine of various classes of numbers, assuming that underflow does not occur.

Table 3-45 FSIN Results

SRC (ST(0))	DEST (ST(0))
$-\infty$	*
-F	- 1 to +1
-0	-0
+0	+0
+F	- 1 to +1
$+\infty$	*
NaN	NaN

NOTES:

F Means finite floating-point value.

* Indicates floating-point invalid-arithmetic-operand (#IA) exception.

If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged. The instruction does not raise an exception when the source operand is out of range. It is up to the program to check the C2 flag for out-of-range conditions. Source values outside the range -2^{63} to $+2^{63}$ can be reduced to the range of the instruction by subtracting an appropriate integer multiple of 2π or by using the FPREM instruction with a divisor of 2π . See the section titled "Pi" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a discussion of the proper value to use for π in performing such reductions.

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

Operation

```
IF  $-2^{63} < ST(0) < 2^{63}$ 
  THEN
    C2  $\leftarrow$  0;
    ST(0)  $\leftarrow$  sin(ST(0));
  ELSE (* Source operand out of range *)
    C2  $\leftarrow$  1;
FI;
```

FPU Flags Affected

C1	Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C2	Set to 1 if outside range ($-2^{63} < \text{source operand} < +2^{63}$); otherwise, set to 0.
C0, C3	Undefined.

Floating-Point Exceptions

#IS	Stack underflow occurred.
#IA	Source operand is an SNaN value, ∞ , or unsupported format.
#D	Source operand is a denormal value.
#P	Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#MF	If there is a pending x87 FPU exception.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

INVD—Invalidate Internal Caches

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 08	INVD	NP	Valid	Valid	Flush internal caches; initiate flushing of external caches.

NOTES:

* See the IA-32 Architecture Compatibility section below.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Invalidates (flushes) the processor's internal caches and issues a special-function bus cycle that directs external caches to also flush themselves. Data held in internal caches is not written back to main memory.

After executing this instruction, the processor does not wait for the external caches to complete their flushing operation before proceeding with instruction execution. It is the responsibility of hardware to respond to the cache flush signal.

The INVD instruction is a privileged instruction. When the processor is running in protected mode, the CPL of a program or procedure must be 0 to execute this instruction.

The INVD instruction may be used when the cache is used as temporary memory and the cache contents need to be invalidated rather than written back to memory. When the cache is used as temporary memory, no external device should be actively writing data to main memory.

Use this instruction with care. Data cached internally and not written back to main memory will be lost. Note that any data from an external device to main memory (for example, via a PCIWrite) can be temporarily stored in the caches; these data can be lost when an INVD instruction is executed. Unless there is a specific requirement or benefit to flushing caches without writing back modified cache lines (for example, temporary memory, testing, or fault recovery where cache coherency with main memory is not a concern), software should instead use the WBINVD instruction.

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

IA-32 Architecture Compatibility

The INVD instruction is implementation dependent; it may be implemented differently on different families of Intel 64 or IA-32 processors. This instruction is not supported on IA-32 processors earlier than the Intel486 processor.

Operation

Flush(InternalCaches);
SignalFlush(ExternalCaches);
Continue (* Continue execution *)

Flags Affected

None.

Protected Mode Exceptions

- #GP(0) If the current privilege level is not 0.
- #UD If the LOCK prefix is used.

Real-Address Mode Exceptions

- #UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

- #GP(0) The INVD instruction cannot be executed in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

INVLPG—Invalidate TLB Entry

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F 01/7	INVLPG <i>m</i>	M	Valid	Valid	Invalidate TLB Entry for page that contains <i>m</i> .

NOTES:

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

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r)	NA	NA	NA

Description

Invalidates (flushes) the translation lookaside buffer (TLB) entry specified with the source operand. The source operand is a memory address. The processor determines the page that contains that address and flushes the TLB entry for that page.

The INVLPG instruction is a privileged instruction. When the processor is running in protected mode, the CPL must be 0 to execute this instruction.

The INVLPG instruction normally flushes the TLB entry only for the specified page; however, in some cases, it may flush more entries, even the entire TLB. The instruction is guaranteed to invalidate only TLB entries associated with the current PCID. (If PCIDs are disabled — CR4.PCIDE = 0 — the current PCID is 000H.) The instruction also invalidates any global TLB entries for the specified page, regardless of PCID.

For more details on operations that flush the TLB, see “MOV—Move to/from Control Registers” in Chapter 4 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B* and Section 4.10.4.1, “Operations that Invalidate TLBs and Paging-Structure Caches,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

This instruction’s operation is the same in all non-64-bit modes. It also operates the same in 64-bit mode, except if the memory address is in non-canonical form. In this case, INVLPG is the same as a NOP.

IA-32 Architecture Compatibility

The INVLPG instruction is implementation dependent, and its function may be implemented differently on different families of Intel 64 or IA-32 processors. This instruction is not supported on IA-32 processors earlier than the Intel486 processor.

Operation

Flush(RelevantTLBEntries);
Continue; (* Continue execution *)

Flags Affected

None.

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0.
#UD Operand is a register.
If the LOCK prefix is used.

Real-Address Mode Exceptions

#UD Operand is a register.
If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0) The INVLPG instruction cannot be executed at the virtual-8086 mode.

64-Bit Mode Exceptions

#GP(0) If the current privilege level is not 0.
#UD Operand is a register.
If the LOCK prefix is used.

...

IRET/IRETD—Interrupt Return

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
CF	IRET	NP	Valid	Valid	Interrupt return (16-bit operand size).
CF	IRETD	NP	Valid	Valid	Interrupt return (32-bit operand size).
REX.W + CF	IRETQ	NP	Valid	N.E.	Interrupt return (64-bit operand size).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Returns program control from an exception or interrupt handler to a program or procedure that was interrupted by an exception, an external interrupt, or a software-generated interrupt. These instructions are also used to perform a return from a nested task. (A nested task is created when a CALL instruction is used to initiate a task switch or when an interrupt or exception causes a task switch to an interrupt or exception handler.) See the section titled “Task Linking” in Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

IRET and IRETD are mnemonics for the same opcode. The IRETD mnemonic (interrupt return double) is intended for use when returning from an interrupt when using the 32-bit operand size; however, most assemblers use the IRET mnemonic interchangeably for both operand sizes.

In Real-Address Mode, the IRET instruction performs a far return to the interrupted program or procedure. During this operation, the processor pops the return instruction pointer, return code segment selector, and EFLAGS image from the stack to the EIP, CS, and EFLAGS registers, respectively, and then resumes execution of the interrupted program or procedure.

In Protected Mode, the action of the IRET instruction depends on the settings of the NT (nested task) and VM flags in the EFLAGS register and the VM flag in the EFLAGS image stored on the current stack. Depending on the setting of these flags, the processor performs the following types of interrupt returns:

- Return from virtual-8086 mode.
- Return to virtual-8086 mode.
- Intra-privilege level return.
- Inter-privilege level return.
- Return from nested task (task switch).

If the NT flag (EFLAGS register) is cleared, the IRET instruction performs a far return from the interrupt procedure, without a task switch. The code segment being returned to must be equally or less privileged than the interrupt handler routine (as indicated by the RPL field of the code segment selector popped from the stack).

As with a real-address mode interrupt return, the IRET instruction pops the return instruction pointer, return code segment selector, and EFLAGS image from the stack to the EIP, CS, and EFLAGS registers, respectively, and then resumes execution of the interrupted program or procedure. If the return is to another privilege level, the IRET instruction also pops the stack pointer and SS from the stack, before resuming program execution. If the return is to virtual-8086 mode, the processor also pops the data segment registers from the stack.

If the NT flag is set, the IRET instruction performs a task switch (return) from a nested task (a task called with a CALL instruction, an interrupt, or an exception) back to the calling or interrupted task. The updated state of the task executing the IRET instruction is saved in its TSS. If the task is re-entered later, the code that follows the IRET instruction is executed.

If the NT flag is set and the processor is in IA-32e mode, the IRET instruction causes a general protection exception.

In 64-bit mode, the instruction’s default operation size is 32 bits. Use of the REX.W prefix promotes operation to 64 bits (IRETQ). See the summary chart at the beginning of this section for encoding data and limits.

See “Changes to Instruction Behavior in VMX Non-Root Operation” in Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
IF PE = 0
  THEN
    GOTO REAL-ADDRESS-MODE;
  ELSE
```

```

    IF (IA32_EFER.LMA = 0)
        THEN (* Protected mode *)
            GOTO PROTECTED-MODE;
        ELSE (* IA-32e mode *)
            GOTO IA-32e-MODE;
    FI;
FI;
REAL-ADDRESS-MODE:
    IF OperandSize = 32
        THEN
            IF top 12 bytes of stack not within stack limits
                THEN #SS; FI;
            tempEIP ← 4 bytes at end of stack
            IF tempEIP[31:16] is not zero THEN #GP(0); FI;
            EIP ← Pop();
            CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            tempEFLAGS ← Pop();
            EFLAGS ← (tempEFLAGS AND 257FD5H) OR (EFLAGS AND 1A0000H);
        ELSE (* OperandSize = 16 *)
            IF top 6 bytes of stack are not within stack limits
                THEN #SS; FI;
            EIP ← Pop(); (* 16-bit pop; clear upper 16 bits *)
            CS ← Pop(); (* 16-bit pop *)
            EFLAGS[15:0] ← Pop();
    FI;
END;
PROTECTED-MODE:
    IF VM = 1 (* Virtual-8086 mode: PE = 1, VM = 1 *)
        THEN
            GOTO RETURN-FROM-VIRTUAL-8086-MODE; (* PE = 1, VM = 1 *)
    FI;
    IF NT = 1
        THEN
            GOTO TASK-RETURN; (* PE = 1, VM = 0, NT = 1 *)
    FI;
    IF OperandSize = 32
        THEN
            IF top 12 bytes of stack not within stack limits
                THEN #SS(0); FI;
            tempEIP ← Pop();
            tempCS ← Pop();
            tempEFLAGS ← Pop();
        ELSE (* OperandSize = 16 *)
            IF top 6 bytes of stack are not within stack limits
                THEN #SS(0); FI;
            tempEIP ← Pop();
            tempCS ← Pop();
            tempEFLAGS ← Pop();
            tempEIP ← tempEIP AND FFFFH;
            tempEFLAGS ← tempEFLAGS AND FFFFH;

```

```

FI;
IF tempEFLAGS(VM) = 1 and CPL = 0
    THEN
        GOTO RETURN-TO-VIRTUAL-8086-MODE;
    ELSE
        GOTO PROTECTED-MODE-RETURN;
FI;
IA-32e-MODE:
IF NT = 1
    THEN #GP(0);
ELSE IF OperandSize = 32
    THEN
        IF top 12 bytes of stack not within stack limits
            THEN #SS(0); FI;
        tempEIP ← Pop();
        tempCS ← Pop();
        tempEFLAGS ← Pop();
    ELSE IF OperandSize = 16
        THEN
            IF top 6 bytes of stack are not within stack limits
                THEN #SS(0); FI;
            tempEIP ← Pop();
            tempCS ← Pop();
            tempEFLAGS ← Pop();
            tempEIP ← tempEIP AND FFFFH;
            tempEFLAGS ← tempEFLAGS AND FFFFH;
        FI;
    ELSE (* OperandSize = 64 *)
        THEN
            tempRIP ← Pop();
            tempCS ← Pop();
            tempEFLAGS ← Pop();
            tempRSP ← Pop();
            tempSS ← Pop();
FI;
GOTO IA-32e-MODE-RETURN;

RETURN-FROM-VIRTUAL-8086-MODE:
(* Processor is in virtual-8086 mode when IRET is executed and stays in virtual-8086 mode *)
IF IOPL = 3 (* Virtual mode: PE = 1, VM = 1, IOPL = 3 *)
    THEN IF OperandSize = 32
        THEN
            IF top 12 bytes of stack not within stack limits
                THEN #SS(0); FI;
            IF instruction pointer not within code segment limits
                THEN #GP(0); FI;
            EIP ← Pop();
            CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            EFLAGS ← Pop();
            (* VM, IOPL, VIP and VIF EFLAG bits not modified by pop *)

```

```

        ELSE (* OperandSize = 16 *)
            IF top 6 bytes of stack are not within stack limits
                THEN #SS(0); FI;
            IF instruction pointer not within code segment limits
                THEN #GP(0); FI;
            EIP ← Pop();
            EIP ← EIP AND 0000FFFFH;
            CS ← Pop(); (* 16-bit pop *)
            EFLAGS[15:0] ← Pop(); (* IOPL in EFLAGS not modified by pop *)
        FI;
    ELSE
        #GP(0); (* Trap to virtual-8086 monitor: PE = 1, VM = 1, IOPL < 3 *)
    FI;
END;

RETURN-TO-VIRTUAL-8086-MODE:
(* Interrupted procedure was in virtual-8086 mode: PE = 1, CPL=0, VM = 1 in flag image *)
IF top 24 bytes of stack are not within stack segment limits
    THEN #SS(0); FI;
IF instruction pointer not within code segment limits
    THEN #GP(0); FI;
CS ← tempCS;
EIP ← tempEIP & FFFFH;
EFLAGS ← tempEFLAGS;
TempESP ← Pop();
TempSS ← Pop();
ES ← Pop(); (* Pop 2 words; throw away high-order word *)
DS ← Pop(); (* Pop 2 words; throw away high-order word *)
FS ← Pop(); (* Pop 2 words; throw away high-order word *)
GS ← Pop(); (* Pop 2 words; throw away high-order word *)
SS:ESP ← TempSS:TempESP;
CPL ← 3;
(* Resume execution in Virtual-8086 mode *)
END;

TASK-RETURN: (* PE = 1, VM = 0, NT = 1 *)
    Read segment selector in link field of current TSS;
    IF local/global bit is set to local
        or index not within GDT limits
            THEN #TS (TSS selector); FI;
    Access TSS for task specified in link field of current TSS;
    IF TSS descriptor type is not TSS or if the TSS is marked not busy
        THEN #TS (TSS selector); FI;
    IF TSS not present
        THEN #NP(TSS selector); FI;
    SWITCH-TASKS (without nesting) to TSS specified in link field of current TSS;
    Mark the task just abandoned as NOT BUSY;
    IF EIP is not within code segment limit
        THEN #GP(0); FI;
END;

```

```

PROTECTED-MODE-RETURN: (* PE = 1 *)
  IF return code segment selector is NULL
    THEN GP(0); FI;
  IF return code segment selector addresses descriptor beyond descriptor table limit
    THEN GP(selector); FI;
  Read segment descriptor pointed to by the return code segment selector;
  IF return code segment descriptor is not a code segment
    THEN #GP(selector); FI;
  IF return code segment selector RPL < CPL
    THEN #GP(selector); FI;
  IF return code segment descriptor is conforming
  and return code segment DPL > return code segment selector RPL
    THEN #GP(selector); FI;
  IF return code segment descriptor is not present
    THEN #NP(selector); FI;
  IF return code segment selector RPL > CPL
    THEN GOTO RETURN-OUTER-PRIVILEGE-LEVEL;
    ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL; FI;
END;

```

```

RETURN-TO-SAME-PRIVILEGE-LEVEL: (* PE = 1, RPL = CPL *)
  IF new mode ≠ 64-Bit Mode
    THEN
      IF tempEIP is not within code segment limits
        THEN #GP(0); FI;
      EIP ← tempEIP;
    ELSE (* new mode = 64-bit mode *)
      IF tempRIP is non-canonical
        THEN #GP(0); FI;
      RIP ← tempRIP;
  FI;
  CS ← tempCS; (* Segment descriptor information also loaded *)
  EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
  IF OperandSize = 32 or OperandSize = 64
    THEN EFLAGS(RF, AC, ID) ← tempEFLAGS; FI;
  IF CPL ≤ IOPL
    THEN EFLAGS(IF) ← tempEFLAGS; FI;
  IF CPL = 0
    THEN (* VM = 0 in flags image *)
      EFLAGS(IOPL) ← tempEFLAGS;
      IF OperandSize = 32 or OperandSize = 64
        THEN EFLAGS(VIF, VIP) ← tempEFLAGS; FI;
  FI;
END;

```

```

RETURN-TO-OUTER-PRIVILEGE-LEVEL:
  IF OperandSize = 32
    THEN
      IF top 8 bytes on stack are not within limits

```

```

        THEN #SS(0); FI;
    ELSE (* OperandSize = 16 *)
        IF top 4 bytes on stack are not within limits
            THEN #SS(0); FI;
    FI;
    Read return segment selector;
    IF stack segment selector is NULL
        THEN #GP(0); FI;
    IF return stack segment selector index is not within its descriptor table limits
        THEN #GP(SSselector); FI;
    Read segment descriptor pointed to by return segment selector;
    IF stack segment selector RPL ≠ RPL of the return code segment selector
    or the stack segment descriptor does not indicate a writable data segment;
    or the stack segment DPL ≠ RPL of the return code segment selector
        THEN #GP(SS selector); FI;
    IF stack segment is not present
        THEN #SS(SS selector); FI;
    IF new mode ≠ 64-Bit Mode
        THEN
            IF tempEIP is not within code segment limits
                THEN #GP(0); FI;
            EIP ← tempEIP;
        ELSE (* new mode = 64-bit mode *)
            IF tempRIP is non-canonical
                THEN #GP(0); FI;
            RIP ← tempRIP;
        FI;
    CS ← tempCS;
    EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
    IF OperandSize = 32
        THEN EFLAGS(RF, AC, ID) ← tempEFLAGS; FI;
    IF CPL ≤ IOPL
        THEN EFLAGS(IF) ← tempEFLAGS; FI;
    IF CPL = 0
        THEN
            EFLAGS(IOPL) ← tempEFLAGS;
            IF OperandSize = 32
                THEN EFLAGS(VM, VIF, VIP) ← tempEFLAGS; FI;
            IF OperandSize = 64
                THEN EFLAGS(VIF, VIP) ← tempEFLAGS; FI;
        FI;
    CPL ← RPL of the return code segment selector;
    FOR each of segment register (ES, FS, GS, and DS)
        DO
            IF segment register points to data or non-conforming code segment
            and CPL > segment descriptor DPL (* Stored in hidden part of segment register *)
                THEN (* Segment register invalid *)
                    SegmentSelector ← 0; (* NULL segment selector *)
            FI;
        OD;

```

END;

```
IA-32e-MODE-RETURN: (* IA32_EFER.LMA = 1, PE = 1 *)
  IF ( (return code segment selector is NULL) or (return RIP is non-canonical) or
      (SS selector is NULL going back to compatibility mode) or
      (SS selector is NULL going back to CPL3 64-bit mode) or
      (RPL <> CPL going back to non-CPL3 64-bit mode for a NULL SS selector) )
    THEN GP(0); FI;
  IF return code segment selector addresses descriptor beyond descriptor table limit
    THEN GP(selector); FI;
  Read segment descriptor pointed to by the return code segment selector;
  IF return code segment descriptor is not a code segment
    THEN #GP(selector); FI;
  IF return code segment selector RPL < CPL
    THEN #GP(selector); FI;
  IF return code segment descriptor is conforming
  and return code segment DPL > return code segment selector RPL
    THEN #GP(selector); FI;
  IF return code segment descriptor is not present
    THEN #NP(selector); FI;
  IF return code segment selector RPL > CPL
    THEN GOTO RETURN-OUTER-PRIVILEGE-LEVEL;
    ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL; FI;
END;
```

Flags Affected

All the flags and fields in the EFLAGS register are potentially modified, depending on the mode of operation of the processor. If performing a return from a nested task to a previous task, the EFLAGS register will be modified according to the EFLAGS image stored in the previous task's TSS.

Protected Mode Exceptions

#GP(0)	If the return code or stack segment selector is NULL. If the return instruction pointer is not within the return code segment limit.
#GP(selector)	If a segment selector index is outside its descriptor table limits. If the return code segment selector RPL is less than the CPL. If the DPL of a conforming-code segment is greater than the return code segment selector RPL. If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector. If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector. If the stack segment is not a writable data segment. If the stack segment selector RPL is not equal to the RPL of the return code segment selector. If the segment descriptor for a code segment does not indicate it is a code segment. If the segment selector for a TSS has its local/global bit set for local. If a TSS segment descriptor specifies that the TSS is not busy. If a TSS segment descriptor specifies that the TSS is not available.
#SS(0)	If the top bytes of stack are not within stack limits.

#NP(selector)	If the return code or stack segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If the return instruction pointer is not within the return code segment limit.
#SS	If the top bytes of stack are not within stack limits.

Virtual-8086 Mode Exceptions

#GP(0)	If the return instruction pointer is not within the return code segment limit. If IOPL not equal to 3.
#PF(fault-code)	If a page fault occurs.
#SS(0)	If the top bytes of stack are not within stack limits.
#AC(0)	If an unaligned memory reference occurs and alignment checking is enabled.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

#GP(0)	If EFLAGS.NT[bit 14] = 1.
--------	---------------------------

Other exceptions same as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If EFLAGS.NT[bit 14] = 1. If the return code segment selector is NULL. If the stack segment selector is NULL going back to compatibility mode. If the stack segment selector is NULL going back to CPL3 64-bit mode. If a NULL stack segment selector RPL is not equal to CPL going back to non-CPL3 64-bit mode. If the return instruction pointer is not within the return code segment limit. If the return instruction pointer is non-canonical.
#GP(Selector)	If a segment selector index is outside its descriptor table limits. If a segment descriptor memory address is non-canonical. If the segment descriptor for a code segment does not indicate it is a code segment. If the proposed new code segment descriptor has both the D-bit and L-bit set. If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector. If CPL is greater than the RPL of the code segment selector. If the DPL of a conforming-code segment is greater than the return code segment selector RPL. If the stack segment is not a writable data segment. If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector. If the stack segment selector RPL is not equal to the RPL of the return code segment selector.
#SS(0)	If an attempt to pop a value off the stack violates the SS limit.

	If an attempt to pop a value off the stack causes a non-canonical address to be referenced.
#NP(selector)	If the return code or stack segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.
#UD	If the LOCK prefix is used.
...	

7. Updates to Chapter 4, Volume 2B

Change bars show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-Z*.

MOV—Move

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
88 /r	MOV r/m8,r8	MR	Valid	Valid	Move r8 to r/m8.
REX + 88 /r	MOV r/m8 ^{***} ,r8 ^{***}	MR	Valid	N.E.	Move r8 to r/m8.
89 /r	MOV r/m16,r16	MR	Valid	Valid	Move r16 to r/m16.
89 /r	MOV r/m32,r32	MR	Valid	Valid	Move r32 to r/m32.
REX.W + 89 /r	MOV r/m64,r64	MR	Valid	N.E.	Move r64 to r/m64.
8A /r	MOV r8,r/m8	RM	Valid	Valid	Move r/m8 to r8.
REX + 8A /r	MOV r8 ^{***} ,r/m8 ^{***}	RM	Valid	N.E.	Move r/m8 to r8.
8B /r	MOV r16,r/m16	RM	Valid	Valid	Move r/m16 to r16.
8B /r	MOV r32,r/m32	RM	Valid	Valid	Move r/m32 to r32.
REX.W + 8B /r	MOV r64,r/m64	RM	Valid	N.E.	Move r/m64 to r64.
8C /r	MOV r/m16,Sreg ^{**}	MR	Valid	Valid	Move segment register to r/m16.
REX.W + 8C /r	MOV r/m64,Sreg ^{**}	MR	Valid	Valid	Move zero extended 16-bit segment register to r/m64.
8E /r	MOV Sreg,r/m16 ^{**}	RM	Valid	Valid	Move r/m16 to segment register.
REX.W + 8E /r	MOV Sreg,r/m64 ^{**}	RM	Valid	Valid	Move lower 16 bits of r/m64 to segment register.
A0	MOV AL,moffs8 [*]	FD	Valid	Valid	Move byte at (seg:offset) to AL.
REX.W + A0	MOV AL,moffs8 [*]	FD	Valid	N.E.	Move byte at (offset) to AL.
A1	MOV AX,moffs16 [*]	FD	Valid	Valid	Move word at (seg:offset) to AX.
A1	MOV EAX,moffs32 [*]	FD	Valid	Valid	Move doubleword at (seg:offset) to EAX.
REX.W + A1	MOV RAX,moffs64 [*]	FD	Valid	N.E.	Move quadword at (offset) to RAX.
A2	MOV moffs8,AL	TD	Valid	Valid	Move AL to (seg:offset).
REX.W + A2	MOV moffs8 ^{***} ,AL	TD	Valid	N.E.	Move AL to (offset).
A3	MOV moffs16 [*] ,AX	TD	Valid	Valid	Move AX to (seg:offset).

A3	MOV <i>moffs32*</i> ,EAX	TD	Valid	Valid	Move EAX to (<i>seg:offset</i>).
REX.W + A3	MOV <i>moffs64*</i> ,RAX	TD	Valid	N.E.	Move RAX to (<i>offset</i>).
B0+ <i>rb</i>	MOV <i>r8</i> , <i>imm8</i>	OI	Valid	Valid	Move <i>imm8</i> to <i>r8</i> .
REX + B0+ <i>rb</i>	MOV <i>r8***</i> , <i>imm8</i>	OI	Valid	N.E.	Move <i>imm8</i> to <i>r8</i> .
B8+ <i>rw</i>	MOV <i>r16</i> , <i>imm16</i>	OI	Valid	Valid	Move <i>imm16</i> to <i>r16</i> .
B8+ <i>rd</i>	MOV <i>r32</i> , <i>imm32</i>	OI	Valid	Valid	Move <i>imm32</i> to <i>r32</i> .
REX.W + B8+ <i>rd</i>	MOV <i>r64</i> , <i>imm64</i>	OI	Valid	N.E.	Move <i>imm64</i> to <i>r64</i> .
C6 /0	MOV <i>r/m8</i> , <i>imm8</i>	MI	Valid	Valid	Move <i>imm8</i> to <i>r/m8</i> .
REX + C6 /0	MOV <i>r/m8***</i> , <i>imm8</i>	MI	Valid	N.E.	Move <i>imm8</i> to <i>r/m8</i> .
C7 /0	MOV <i>r/m16</i> , <i>imm16</i>	MI	Valid	Valid	Move <i>imm16</i> to <i>r/m16</i> .
C7 /0	MOV <i>r/m32</i> , <i>imm32</i>	MI	Valid	Valid	Move <i>imm32</i> to <i>r/m32</i> .
REX.W + C7 /0	MOV <i>r/m64</i> , <i>imm32</i>	MI	Valid	N.E.	Move <i>imm32</i> sign extended to 64-bits to <i>r/m64</i> .

NOTES:

* The *moffs8*, *moffs16*, *moffs32* and *moffs64* operands specify a simple offset relative to the segment base, where 8, 16, 32 and 64 refer to the size of the data. The address-size attribute of the instruction determines the size of the offset, either 16, 32 or 64 bits.

** In 32-bit mode, the assembler may insert the 16-bit operand-size prefix with this instruction (see the following “Description” section for further information).

***In 64-bit mode, *r/m8* can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA
FD	AL/AX/EAX/RAX	Moffs	NA	NA
TD	Moffs (w)	AL/AX/EAX/RAX	NA	NA
OI	opcode + rd (w)	imm8/16/32/64	NA	NA
MI	ModRM:r/m (w)	imm8/16/32/64	NA	NA

Description

Copies the second operand (source operand) to the first operand (destination operand). The source operand can be an immediate value, general-purpose register, segment register, or memory location; the destination register can be a general-purpose register, segment register, or memory location. Both operands must be the same size, which can be a byte, a word, a doubleword, or a quadword.

The MOV instruction cannot be used to load the CS register. Attempting to do so results in an invalid opcode exception (#UD). To load the CS register, use the far JMP, CALL, or RET instruction.

If the destination operand is a segment register (DS, ES, FS, GS, or SS), the source operand must be a valid segment selector. In protected mode, moving a segment selector into a segment register automatically causes the segment descriptor information associated with that segment selector to be loaded into the hidden (shadow) part of the segment register. While loading this information, the segment selector and segment descriptor infor-

mation is validated (see the “Operation” algorithm below). The segment descriptor data is obtained from the GDT or LDT entry for the specified segment selector.

A NULL segment selector (values 0000-0003) can be loaded into the DS, ES, FS, and GS registers without causing a protection exception. However, any subsequent attempt to reference a segment whose corresponding segment register is loaded with a NULL value causes a general protection exception (#GP) and no memory reference occurs.

Loading the SS register with a MOV instruction inhibits all interrupts until after the execution of the next instruction. This operation allows a stack pointer to be loaded into the ESP register with the next instruction (MOV ESP, **stack-pointer value**) before an interrupt occurs¹. Be aware that the LSS instruction offers a more efficient method of loading the SS and ESP registers.

When operating in 32-bit mode and moving data between a segment register and a general-purpose register, the 32-bit IA-32 processors do not require the use of the 16-bit operand-size prefix (a byte with the value 66H) with this instruction, but most assemblers will insert it if the standard form of the instruction is used (for example, MOV DS, AX). The processor will execute this instruction correctly, but it will usually require an extra clock. With most assemblers, using the instruction form MOV DS, EAX will avoid this unneeded 66H prefix. When the processor executes the instruction with a 32-bit general-purpose register, it assumes that the 16 least-significant bits of the general-purpose register are the destination or source operand. If the register is a destination operand, the resulting value in the two high-order bytes of the register is implementation dependent. For the Pentium 4, Intel Xeon, and P6 family processors, the two high-order bytes are filled with zeros; for earlier 32-bit IA-32 processors, the two high order bytes are undefined.

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

Operation

DEST ← SRC;

Loading a segment register while in protected mode results in special checks and actions, as described in the following listing. These checks are performed on the segment selector and the segment descriptor to which it points.

```
IF SS is loaded
  THEN
    IF segment selector is NULL
      THEN #GP(0); FI;
    IF segment selector index is outside descriptor table limits
      or segment selector’s RPL ≠ CPL
      or segment is not a writable data segment
      or DPL ≠ CPL
      THEN #GP(selector); FI;
    IF segment not marked present
```

1. If a code instruction breakpoint (for debug) is placed on an instruction located immediately after a MOV SS instruction, the breakpoint may not be triggered. However, in a sequence of instructions that load the SS register, only the first instruction in the sequence is guaranteed to delay an interrupt.

In the following sequence, interrupts may be recognized before MOV ESP, EBP executes:

```
MOV SS, EDX
MOV SS, EAX
MOV ESP, EBP
```

```

        THEN #SS(selector);
        ELSE
            SS ← segment selector;
            SS ← segment descriptor; FI;
FI;

IF DS, ES, FS, or GS is loaded with non-NULL selector
THEN
    IF segment selector index is outside descriptor table limits
    or segment is not a data or readable code segment
    or ((segment is a data or nonconforming code segment)
    or ((RPL > DPL) and (CPL > DPL))
        THEN #GP(selector); FI;
    IF segment not marked present
        THEN #NP(selector);
        ELSE
            SegmentRegister ← segment selector;
            SegmentRegister ← segment descriptor; FI;
FI;

IF DS, ES, FS, or GS is loaded with NULL selector
THEN
    SegmentRegister ← segment selector;
    SegmentRegister ← segment descriptor;
FI;

```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If attempt is made to load SS register with NULL segment selector. If the destination operand is 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.
#GP(selector)	If segment selector index is outside descriptor table limits. If the SS register is being loaded and the segment selector's RPL and the segment descriptor's DPL are not equal to the CPL. If the SS register is being loaded and the segment pointed to is a non-writable data segment. If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment. If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#SS(selector)	If the SS register is being loaded and the segment pointed to is marked not present.
#NP	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is marked not present.
#PF(fault-code)	If a page fault occurs.

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

...

MOVD/MOVQ—Move Doubleword/Move Quadword

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
0F 6E /r MOVD mm, r/m32	RM	V/V	MMX	Move doubleword from r/m32 to mm.
REX.W + 0F 6E /r MOVQ mm, r/m64	RM	V/N.E.	MMX	Move quadword from r/m64 to mm.
0F 7E /r MOVD r/m32, mm	MR	V/V	MMX	Move doubleword from mm to r/m32.
REX.W + 0F 7E /r MOVQ r/m64, mm	MR	V/N.E.	MMX	Move quadword from mm to r/m64.
VEX.128.66.0F.W0 6E / VMOVD xmm1, r32/m32	RM	V/V	AVX	Move doubleword from r/m32 to xmm1.
VEX.128.66.0F.W1 6E /r VMOVQ xmm1, r64/m64	RM	V/N.E.	AVX	Move quadword from r/m64 to xmm1.
66 0F 6E /r MOVD xmm, r/m32	RM	V/V	SSE2	Move doubleword from r/m32 to xmm.
66 REX.W 0F 6E /r MOVQ xmm, r/m64	RM	V/N.E.	SSE2	Move quadword from r/m64 to xmm.

66 0F 7E /r MOVD r/m32, xmm	MR	V/V	SSE2	Move doubleword from <i>xmm</i> register to <i>r/m32</i> .
66 REX.W 0F 7E /r MOVQ r/m64, xmm	MR	V/N.E.	SSE2	Move quadword from <i>xmm</i> register to <i>r/m64</i> .
VEX.128.66.0F.W0 7E /r VMOVD r32/m32, xmm1	MR	V/V	AVX	Move doubleword from <i>xmm1</i> register to <i>r/m32</i> .
VEX.128.66.0F.W1 7E /r VMOVQ r64/m64, xmm1	MR	V/N.E.	AVX	Move quadword from <i>xmm1</i> register to <i>r/m64</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA

Description

Copies a doubleword from the source operand (second operand) to the destination operand (first operand). The source and destination operands can be general-purpose registers, MMX technology registers, XMM registers, or 32-bit memory locations. This instruction can be used to move a doubleword to and from the low doubleword of an MMX technology register and a general-purpose register or a 32-bit memory location, or to and from the low doubleword of an XMM register and a general-purpose register or a 32-bit memory location. The instruction cannot be used to transfer data between MMX technology registers, between XMM registers, between general-purpose registers, or between memory locations.

When the destination operand is an MMX technology register, the source operand is written to the low doubleword of the register, and the register is zero-extended to 64 bits. When the destination operand is an XMM register, the source operand is written to the low doubleword of the register, and the register is zero-extended to 128 bits.

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

Operation

MOVD (when destination operand is MMX technology register)

DEST[31:0] ← SRC;
DEST[63:32] ← 00000000H;

MOVD (when destination operand is XMM register)

DEST[31:0] ← SRC;
DEST[127:32] ← 000000000000000000000000H;
DEST[VLMAX-1:128] (Unmodified)

MOVD (when source operand is MMX technology or XMM register)

DEST ← SRC[31:0];

VMOVD (VEX-encoded version when destination is an XMM register)

DEST[31:0] ← SRC[31:0]
DEST[VLMAX-1:32] ← 0

MOVQ (when destination operand is XMM register)

DEST[63:0] ← SRC[63:0];
DEST[127:64] ← 0000000000000000H;
DEST[VLMAX-1:128] (Unmodified)

MOVQ (when destination operand is r/m64)

DEST[63:0] ← SRC[63:0];

MOVQ (when source operand is XMM register or r/m64)

DEST ← SRC[63:0];

VMOVQ (VEX-encoded version when destination is an XMM register)

DEST[63:0] ← SRC[63:0]
DEST[VLMAX-1:64] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

MOVD: __m64 _mm_cvtsi32_si64 (int i)
MOVD: int _mm_cvtsi64_si32 (__m64m)
MOVD: __m128i _mm_cvtsi32_si128 (int a)
MOVD: int _mm_cvtsi128_si32 (__m128i a)
MOVQ: __int64 _mm_cvtsi128_si64(__m128i);
MOVQ: __m128i _mm_cvtsi64_si128(__int64);

Flags Affected

None.

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 5; additionally

#UD If VEX.L = 1.
 If VEX.vvvv != 1111B.

...

MOVQ—Move Quadword

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
0F 6F /r MOVQ mm, mm/m64	RM	V/V	MMX	Move quadword from mm/m64 to mm.
0F 7F /r MOVQ mm/m64, mm	MR	V/V	MMX	Move quadword from mm to mm/m64.
F3 0F 7E MOVQ xmm1, xmm2/m64	RM	V/V	SSE2	Move quadword from xmm2/mem64 to xmm1.

VEX.128.F3.0F.WIG 7E /r VMOVQ xmm1, xmm2	RM	V/V	AVX	Move quadword from <i>xmm2</i> to <i>xmm1</i> .
VEX.128.F3.0F.WIG 7E /r VMOVQ xmm1, m64	RM	V/V	AVX	Load quadword from <i>m64</i> to <i>xmm1</i> .
66 0F D6 MOVQ <i>xmm2/m64, xmm1</i>	MR	V/V	SSE2	Move quadword from <i>xmm1</i> to <i>xmm2/mem64</i> .
VEX.128.66.0F.WIG D6 /r VMOVQ xmm1/m64, xmm2	MR	V/V	AVX	Move quadword from <i>xmm2</i> register to <i>xmm1/m64</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA

Description

Copies a quadword from the source operand (second operand) to the destination operand (first operand). The source and destination operands can be MMX technology registers, XMM registers, or 64-bit memory locations. This instruction can be used to move a quadword between two MMX technology registers or between an MMX technology register and a 64-bit memory location, or to move data between two XMM registers or between an XMM register and a 64-bit memory location. The instruction cannot be used to transfer data between memory locations.

When the source operand is an XMM register, the low quadword is moved; when the destination operand is an XMM register, the quadword is stored to the low quadword of the register, and the high quadword is cleared to all 0s.

In 64-bit mode, use of the REX prefix in the form of REX.R permits this instruction to access additional registers (XMM8-XMM15).

Note: In VEX.128.66.0F D6 instruction version, VEX.vvvv and VEX.L=1 are reserved and the former must be 1111b otherwise instructions will #UD.

Note: In VEX.128.F3.0F 7E version, VEX.vvvv and VEX.L=1 are reserved and the former must be 1111b, otherwise instructions will #UD.

Operation

MOVQ instruction when operating on MMX technology registers and memory locations:

DEST ← SRC;

MOVQ instruction when source and destination operands are XMM registers:

DEST[63:0] ← SRC[63:0];

DEST[127:64] ← 0000000000000000H;

MOVQ instruction when source operand is XMM register and destination operand is memory location:

DEST ← SRC[63:0];

MOVQ instruction when source operand is memory location and destination operand is XMM register:

DEST[63:0] ← SRC;

DEST[127:64] ← 0000000000000000H;

VMOVQ (VEX.NDS.128.F3.0F 7E) with XMM register source and destination:

DEST[63:0] ← SRC[63:0]
 DEST[VLMAX-1:64] ← 0

VMOVQ (VEX.128.66.0F D6) with XMM register source and destination:

DEST[63:0] ← SRC[63:0]
 DEST[VLMAX-1:64] ← 0

VMOVQ (7E) with memory source:

DEST[63:0] ← SRC[63:0]
 DEST[VLMAX-1:64] ← 0

VMOVQ (D6) with memory dest:

DEST[63:0] ← SRC2[63:0]

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

MOVQ: `m128i_mm_mov_epi64(__m128i a)`

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Table 22-8, “Exception Conditions for Legacy SIMD/MMX Instructions without FP Exception,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B*.

...

PCMPESTRI — Packed Compare Explicit Length Strings, Return Index

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 61 /r imm8 PCMPESTRI xmm1, xmm2/m128, imm8	RMI	V/V	SSE4_2	Perform a packed comparison of string data with explicit lengths, generating an index, and storing the result in ECX.
VEX.128.66.0F3A.WIG 61 /r ib VPCMPESTRI xmm1, xmm2/m128, imm8	RMI	V/V	AVX	Perform a packed comparison of string data with explicit lengths, generating an index, and storing the result in ECX.

Instruction Operand Encoding

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

Description

The instruction compares and processes data from two string fragments based on the encoded value in the Imm8 Control Byte (see Section 4.1, “Imm8 Control Byte Operation for PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMP-ISTRM”), and generates an index stored to the count register (ECX/RCX).

Each string fragment is represented by two values. The first value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). The second value is stored in an input length register. The input length register is EAX/RAX (for xmm1) or EDX/RDX (for xmm2/m128). The length represents the number of bytes/words which are valid for the respective xmm/m128 data.

The length of each input is interpreted as being the absolute-value of the value in the length register. The absolute-value computation saturates to 16 (for bytes) and 8 (for words), based on the value of imm8[bit3] when the value in the length register is greater than 16 (8) or less than -16 (-8).

The comparison and aggregation operations are performed according to the encoded value of Imm8 bit fields (see Section 4.1). The index of the first (or last, according to imm8[6]) set bit of IntRes2 (see Section 4.1.4) is returned in ECX. If no bits are set in IntRes2, ECX is set to 16 (8).

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

- CFlag - Reset if IntRes2 is equal to zero, set otherwise
- ZFlag - Set if absolute-value of EDX is < 16 (8), reset otherwise
- SFlag - Set if absolute-value of EAX is < 16 (8), reset otherwise
- OFlag - IntRes2[0]
- AFlag - Reset
- PFlag - Reset

Effective Operand Size

Operating mode/size	Operand 1	Operand 2	Length 1	Length 2	Result
16 bit	xmm	xmm/m128	EAX	EDX	ECX
32 bit	xmm	xmm/m128	EAX	EDX	ECX
64 bit	xmm	xmm/m128	EAX	EDX	ECX
64 bit + REX.W	xmm	xmm/m128	RAX	RDX	RCX

Intel C/C++ Compiler Intrinsic Equivalent For Returning Index

```
int  __mm_cmpestri (__m128i a, int la, __m128i b, int lb, const int mode);
```

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

```
int  __mm_cmpestra (__m128i a, int la, __m128i b, int lb, const int mode);
int  __mm_cmpestrc (__m128i a, int la, __m128i b, int lb, const int mode);
int  __mm_cmpestro (__m128i a, int la, __m128i b, int lb, const int mode);
int  __mm_cmpestrs (__m128i a, int la, __m128i b, int lb, const int mode);
int  __mm_cmpestrz (__m128i a, int la, __m128i b, int lb, const int mode);
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4; additionally, this instruction does not cause #GP if the memory operand is not aligned to 16 Byte boundary, and

#UD If VEX.L = 1.
If VEX.vvvv != 1111B.

...

PCMPESTRM – Packed Compare Explicit Length Strings, Return Mask

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 60 /r imm8 PCMPESTRM <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE4_2	Perform a packed comparison of string data with explicit lengths, generating a mask, and storing the result in <i>XMM0</i>
VEX.128.66.0F3A.WIG 60 /r ib VPCMPESTRM <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	AVX	Perform a packed comparison of string data with explicit lengths, generating a mask, and storing the result in <i>XMM0</i> .

Instruction Operand Encoding

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

Description

The instruction compares data from two string fragments based on the encoded value in the imm8 control byte (see Section 4.1, “Imm8 Control Byte Operation for PCMPESTRM / PCMPESTRM / PCMPISTRM / PCMPISTRM”), and generates a mask stored to XMM0.

Each string fragment is represented by two values. The first value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). The second value is stored in an input length register. The input length register is EAX/RAX (for xmm1) or EDX/RDX (for xmm2/m128). The length represents the number of bytes/words which are valid for the respective xmm/m128 data.

The length of each input is interpreted as being the absolute-value of the value in the length register. The absolute-value computation saturates to 16 (for bytes) and 8 (for words), based on the value of imm8[bit3] when the value in the length register is greater than 16 (8) or less than -16 (-8).

The comparison and aggregation operations are performed according to the encoded value of Imm8 bit fields (see Section 4.1). As defined by imm8[6], IntRes2 is then either stored to the least significant bits of XMM0 (zero extended to 128 bits) or expanded into a byte/word-mask and then stored to XMM0.

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

CFlag - Reset if IntRes2 is equal to zero, set otherwise
ZFlag - Set if absolute-value of EDX is < 16 (8), reset otherwise
SFlag - Set if absolute-value of EAX is < 16 (8), reset otherwise
OFlag - IntRes2[0]
AFlag - Reset
PFlag - Reset

Note: In VEX.128 encoded versions, bits (VLMAX-1:128) of XMM0 are zeroed. VEX.vvvv is reserved and must be 1111b, VEX.L must be 0, otherwise the instruction will #UD.

Effective Operand Size

Operating mode/size	Operand1	Operand 2	Length1	Length2	Result
16 bit	xmm	xmm/m128	EAX	EDX	XMM0
32 bit	xmm	xmm/m128	EAX	EDX	XMM0
64 bit	xmm	xmm/m128	EAX	EDX	XMM0
64 bit + REX.W	xmm	xmm/m128	RAX	RDX	XMM0

Intel C/C++ Compiler Intrinsic Equivalent For Returning Mask

```
__m128i _mm_cmpestrm (__m128i a, int la, __m128i b, int lb, const int mode);
```

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

```
int _mm_cmpestra (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrc (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestro (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrs (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrz (__m128i a, int la, __m128i b, int lb, const int mode);
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4; additionally, this instruction does not cause #GP if the memory operand is not aligned to 16 Byte boundary, and

```
#UD                If VEX.L = 1.
                   If VEX.vvvv != 1111B.
```

...

PCMPISTR1 – Packed Compare Implicit Length Strings, Return Index

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 63 /r imm8 PCMPISTR1 <i>xmm1, xmm2/m128, imm8</i>	RM	V/V	SSE4_2	Perform a packed comparison of string data with implicit lengths, generating an index, and storing the result in ECX.
VEX.128.66.0F3A.WIG 63 /r ib VPCMPISTR1 <i>xmm1, xmm2/m128, imm8</i>	RM	V/V	AVX	Perform a packed comparison of string data with implicit lengths, generating an index, and storing the result in ECX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r)	ModRM:r/m (r)	imm8	NA

Description

The instruction compares data from two strings based on the encoded value in the Imm8 Control Byte (see Section 4.1, “Imm8 Control Byte Operation for PCMPSTRM / PCMPSTRM / PCMPSTRM / PCMPSTRM”), and generates an index stored to ECX.

Each string is represented by a single value. The value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). Each input byte/word is augmented with a valid/invalid tag. A byte/word is considered valid only if it has a lower index than the least significant null byte/word. (The least significant null byte/word is also considered invalid.)

The comparison and aggregation operations are performed according to the encoded value of Imm8 bit fields (see Section 4.1). The index of the first (or last, according to imm8[6]) set bit of IntRes2 is returned in ECX. If no bits are set in IntRes2, ECX is set to 16 (8).

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

- CFlag - Reset if IntRes2 is equal to zero, set otherwise
- ZFlag - Set if any byte/word of xmm2/mem128 is null, reset otherwise
- SFlag - Set if any byte/word of xmm1 is null, reset otherwise
- OFlag - IntRes2[0]
- AFlag - Reset
- PFlag - Reset

Note: In VEX.128 encoded version, VEX.vvvv is reserved and must be 1111b, VEX.L must be 0, otherwise the instruction will #UD.

Effective Operand Size

Operating mode/size	Operand1	Operand 2	Result
16 bit	xmm	xmm/m128	ECX
32 bit	xmm	xmm/m128	ECX
64 bit	xmm	xmm/m128	ECX
64 bit + REX.W	xmm	xmm/m128	RCX

Intel C/C++ Compiler Intrinsic Equivalent For Returning Index

```
int __mm_cmpistri (__m128i a, __m128i b, const int mode);
```

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

```
int __mm_cmpistra (__m128i a, __m128i b, const int mode);
int __mm_cmpistrb (__m128i a, __m128i b, const int mode);
int __mm_cmpistrc (__m128i a, __m128i b, const int mode);
int __mm_cmpistro (__m128i a, __m128i b, const int mode);
int __mm_cmpistrs (__m128i a, __m128i b, const int mode);
```

int __mm_cmpistrz (__m128i a, __m128i b, const int mode);

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4; additionally, this instruction does not cause #GP if the memory operand is not aligned to 16 Byte boundary, and

#UD If VEX.L = 1.
If VEX.vvvv != 1111B.

...

PCMPISTRM – Packed Compare Implicit Length Strings, Return Mask

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 62 /r imm8 PCMPISTRM xmm1, xmm2/m128, imm8	RM	V/V	SSE4_2	Perform a packed comparison of string data with implicit lengths, generating a mask, and storing the result in XMM0.
VEX.128.66.0F3A.WIG 62 /r ib VPCMPISTRM xmm1, xmm2/m128, imm8	RM	V/V	AVX	Perform a packed comparison of string data with implicit lengths, generating a Mask, and storing the result in XMM0.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r)	ModRM:r/m (r)	imm8	NA

Description

The instruction compares data from two strings based on the encoded value in the imm8 byte (see Section 4.1, “Imm8 Control Byte Operation for PCMPSTRM / PCMPSTRM / PCMPISTRM / PCMPISTRM”) generating a mask stored to XMM0.

Each string is represented by a single value. The value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). Each input byte/word is augmented with a valid/invalid tag. A byte/word is considered valid only if it has a lower index than the least significant null byte/word. (The least significant null byte/word is also considered invalid.)

The comparison and aggregation operation are performed according to the encoded value of Imm8 bit fields (see Section 4.1). As defined by imm8[6], IntRes2 is then either stored to the least significant bits of XMM0 (zero extended to 128 bits) or expanded into a byte/word-mask and then stored to XMM0.

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

- CFlag – Reset if IntRes2 is equal to zero, set otherwise
- ZFlag – Set if any byte/word of xmm2/mem128 is null, reset otherwise
- SFlag – Set if any byte/word of xmm1 is null, reset otherwise
- OFlag – IntRes2[0]
- AFlag – Reset

PFlag - Reset

Note: In VEX.128 encoded versions, bits (VLMAX-1:128) of XMM0 are zeroed. VEX.vvvv is reserved and must be 1111b, VEX.L must be 0, otherwise the instruction will #UD.

Effective Operand Size

Operating mode/size	Operand1	Operand 2	Result
16 bit	xmm	xmm/m128	XMM0
32 bit	xmm	xmm/m128	XMM0
64 bit	xmm	xmm/m128	XMM0
64 bit + REX.W	xmm	xmm/m128	XMM0

Intel C/C++ Compiler Intrinsic Equivalent For Returning Mask

```
__m128i _mm_cmpistrm (__m128i a, __m128i b, const int mode);
```

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

```
int _mm_cmpistra (__m128i a, __m128i b, const int mode);
int _mm_cmpistrc (__m128i a, __m128i b, const int mode);
int _mm_cmpistro (__m128i a, __m128i b, const int mode);
int _mm_cmpistrs (__m128i a, __m128i b, const int mode);
int _mm_cmpistrz (__m128i a, __m128i b, const int mode);
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4; additionally, this instruction does not cause #GP if the memory operand is not aligned to 16 Byte boundary, and

```
#UD                If VEX.L = 1.
                   If VEX.vvvv != 1111B.
```

...

PINSRB/PINSRD/PINSRQ – Insert Byte/Dword/Qword

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 OF 3A 20 /r ib PINSRB <i>xmm1</i> , <i>r32/m8</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Insert a byte integer value from <i>r32/m8</i> into <i>xmm1</i> at the destination element in <i>xmm1</i> specified by <i>imm8</i> .
66 OF 3A 22 /r ib PINSRD <i>xmm1</i> , <i>r/m32</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Insert a dword integer value from <i>r/m32</i> into the <i>xmm1</i> at the destination element specified by <i>imm8</i> .
66 REX.W OF 3A 22 /r ib PINSRQ <i>xmm1</i> , <i>r/m64</i> , <i>imm8</i>	RMI	N. E./V	SSE4_1	Insert a qword integer value from <i>r/m64</i> into the <i>xmm1</i> at the destination element specified by <i>imm8</i> .

VEX.NDS.128.66.0F3A.W0 20 /r ib VPINSRB xmm1, xmm2, r32/m8, imm8	RVMI V ¹ /V	AVX	Merge a byte integer value from r32/m8 and rest from xmm2 into xmm1 at the byte offset in imm8.
VEX.NDS.128.66.0F3A.W0 22 /r ib VPINSRD xmm1, xmm2, r32/m32, imm8	RVMI V/V	AVX	Insert a dword integer value from r32/m32 and rest from xmm2 into xmm1 at the dword offset in imm8.
VEX.NDS.128.66.0F3A.W1 22 /r ib VPINSRQ xmm1, xmm2, r64/m64, imm8	RVMI V/I	AVX	Insert a qword integer value from r64/m64 and rest from xmm2 into xmm1 at the qword offset in imm8.

NOTES:

1. In 64-bit mode, VEX.W1 is ignored for VPINSRB (similar to legacy REX.W=1 prefix with PINSRB).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RMI	ModRM:reg (w)	ModRM:r/m (r)	imm8	NA
RVMI	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	imm8

Description

Copies a byte/dword/qword from the source operand (second operand) and inserts it in the destination operand (first operand) at the location specified with the count operand (third operand). (The other elements in the destination register are left untouched.) The source operand can be a general-purpose register or a memory location. (When the source operand is a general-purpose register, PINSRB copies the low byte of the register.) The destination operand is an XMM register. The count operand is an 8-bit immediate. When specifying a qword[dword, byte] location in an XMM register, the [2, 4] least-significant bit(s) of the count operand specify the location.

In 64-bit mode, using a REX prefix in the form of REX.R permits this instruction to access additional registers (XMM8-XMM15, R8-15). Use of REX.W permits the use of 64 bit general purpose registers.

128-bit Legacy SSE version: Bits (VLMAX-1:128) of the corresponding YMM destination register remain unchanged.

VEX.128 encoded version: Bits (VLMAX-1:128) of the destination YMM register are zeroed. VEX.L must be 0, otherwise the instruction will #UD. Attempt to execute VPINSRQ in non-64-bit mode will cause #UD.

Operation

CASE OF

```

PINSRB: SEL ← COUNT[3:0];
        MASK ← (OFFH << (SEL * 8));
        TEMP ← (((SRC[7:0] << (SEL * 8)) AND MASK);
PINSRD: SEL ← COUNT[1:0];
        MASK ← (OFFFFFFFFFH << (SEL * 32));
        TEMP ← (((SRC << (SEL * 32)) AND MASK) ;
PINSRQ: SEL ← COUNT[0]
        MASK ← (OFFFFFFFFFHH << (SEL * 64));
        TEMP ← (((SRC << (SEL * 32)) AND MASK) ;

```

ESAC;

```

DEST ← ((DEST AND NOT MASK) OR TEMP);

```

VPINSRB (VEX.128 encoded version)

SEL ← imm8[3:0]
 DEST[127:0] ← write_b_element(SEL, SRC2, SRC1)
 DEST[VLMAX-1:128] ← 0

VPINSRD (VEX.128 encoded version)

SEL ← imm8[1:0]
 DEST[127:0] ← write_d_element(SEL, SRC2, SRC1)
 DEST[VLMAX-1:128] ← 0

VPINSRQ (VEX.128 encoded version)

SEL ← imm8[0]
 DEST[127:0] ← write_q_element(SEL, SRC2, SRC1)
 DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

PINSRB: `__m128i _mm_insert_epi8 (__m128i s1, int s2, const int ndx);`
 PINSRD: `__m128i _mm_insert_epi32 (__m128i s2, int s, const int ndx);`
 PINSRQ: `__m128i _mm_insert_epi64(__m128i s2, __int64 s, const int ndx);`

Flags Affected

None.

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 5; additionally

#UD If VEX.L = 1.
 If VPINSRQ in non-64-bit mode with VEX.W=1.

...

POPCNT – Return the Count of Number of Bits Set to 1

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
F3 0F B8 /r	POPCNT <i>r16, r/m16</i>	RM	Valid	Valid	POPCNT on <i>r/m16</i>
F3 0F B8 /r	POPCNT <i>r32, r/m32</i>	RM	Valid	Valid	POPCNT on <i>r/m32</i>
F3 REX.W 0F B8 /r	POPCNT <i>r64, r/m64</i>	RM	Valid	N.E.	POPCNT on <i>r/m64</i>

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

Description

This instruction calculates of number of bits set to 1 in the second operand (source) and returns the count in the first operand (a destination register).

Operation

```
Count = 0;
For (i=0; i < OperandSize; i++)
{
    IF (SRC[ i] = 1) // i'th bit
        THEN Count++; FI;
}
DEST ← Count;
```

Flags Affected

OF, SF, ZF, AF, CF, PF are all cleared. ZF is set if SRC = 0, otherwise ZF is cleared

Intel C/C++ Compiler Intrinsic Equivalent

POPCNT: `int _mm_popcnt_u32(unsigned int a);`

POPCNT: `int64_t _mm_popcnt_u64(unsigned __int64 a);`

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.
#UD	If CPUID.01H: ECX.POPCNT [Bit 23] = 0. If LOCK prefix is used. Either the prefix REP (F3h) or REPN (F2H) is used.

Real-Address Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If CPUID.01H: ECX.POPCNT [Bit 23] = 0. If LOCK prefix is used. Either the prefix REP (F3h) or REPN (F2H) is used.

Virtual 8086 Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.
#UD	If CPUID.01H: ECX.POPCNT [Bit 23] = 0. If LOCK prefix is used.

Either the prefix REP (F3h) or REPN (F2H) is used.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

- #GP(0) If the memory address is in a non-canonical form.
- #SS(0) If a memory address referencing the SS segment is in a non-canonical form.
- #PF (fault-code) For a page fault.
- #AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
- #UD If CPUID.01H: ECX.POPCNT [Bit 23] = 0.
If LOCK prefix is used.
Either the prefix REP (F3h) or REPN (F2H) is used.

...

SBB—Integer Subtraction with Borrow

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
1C <i>ib</i>	SBB AL, <i>imm8</i>	I	Valid	Valid	Subtract with borrow <i>imm8</i> from AL.
1D <i>iw</i>	SBB AX, <i>imm16</i>	I	Valid	Valid	Subtract with borrow <i>imm16</i> from AX.
1D <i>id</i>	SBB EAX, <i>imm32</i>	I	Valid	Valid	Subtract with borrow <i>imm32</i> from EAX.
REX.W + 1D <i>id</i>	SBB RAX, <i>imm32</i>	I	Valid	N.E.	Subtract with borrow sign-extended <i>imm32</i> to 64-bits from RAX.
80 /3 <i>ib</i>	SBB <i>r/m8</i> , <i>imm8</i>	MI	Valid	Valid	Subtract with borrow <i>imm8</i> from <i>r/m8</i> .
REX + 80 /3 <i>ib</i>	SBB <i>r/m8*</i> , <i>imm8</i>	MI	Valid	N.E.	Subtract with borrow <i>imm8</i> from <i>r/m8</i> .
81 /3 <i>iw</i>	SBB <i>r/m16</i> , <i>imm16</i>	MI	Valid	Valid	Subtract with borrow <i>imm16</i> from <i>r/m16</i> .
81 /3 <i>id</i>	SBB <i>r/m32</i> , <i>imm32</i>	MI	Valid	Valid	Subtract with borrow <i>imm32</i> from <i>r/m32</i> .
REX.W + 81 /3 <i>id</i>	SBB <i>r/m64</i> , <i>imm32</i>	MI	Valid	N.E.	Subtract with borrow sign-extended <i>imm32</i> to 64-bits from <i>r/m64</i> .
83 /3 <i>ib</i>	SBB <i>r/m16</i> , <i>imm8</i>	MI	Valid	Valid	Subtract with borrow sign-extended <i>imm8</i> from <i>r/m16</i> .
83 /3 <i>ib</i>	SBB <i>r/m32</i> , <i>imm8</i>	MI	Valid	Valid	Subtract with borrow sign-extended <i>imm8</i> from <i>r/m32</i> .
REX.W + 83 /3 <i>ib</i>	SBB <i>r/m64</i> , <i>imm8</i>	MI	Valid	N.E.	Subtract with borrow sign-extended <i>imm8</i> from <i>r/m64</i> .
18 /r	SBB <i>r/m8</i> , <i>r8</i>	MR	Valid	Valid	Subtract with borrow <i>r8</i> from <i>r/m8</i> .
REX + 18 /r	SBB <i>r/m8*</i> , <i>r8</i>	MR	Valid	N.E.	Subtract with borrow <i>r8</i> from <i>r/m8</i> .
19 /r	SBB <i>r/m16</i> , <i>r16</i>	MR	Valid	Valid	Subtract with borrow <i>r16</i> from <i>r/m16</i> .
19 /r	SBB <i>r/m32</i> , <i>r32</i>	MR	Valid	Valid	Subtract with borrow <i>r32</i> from <i>r/m32</i> .
REX.W + 19 /r	SBB <i>r/m64</i> , <i>r64</i>	MR	Valid	N.E.	Subtract with borrow <i>r64</i> from <i>r/m64</i> .
1A /r	SBB <i>r8</i> , <i>r/m8</i>	RM	Valid	Valid	Subtract with borrow <i>r/m8</i> from <i>r8</i> .
REX + 1A /r	SBB <i>r8*</i> , <i>r/m8*</i>	RM	Valid	N.E.	Subtract with borrow <i>r/m8</i> from <i>r8</i> .

1B /r	SBB r16, r/m16	RM	Valid	Valid	Subtract with borrow r/m16 from r16.
1B /r	SBB r32, r/m32	RM	Valid	Valid	Subtract with borrow r/m32 from r32.
REX.W + 1B /r	SBB r64, r/m64	RM	Valid	N.E.	Subtract with borrow r/m64 from r64.

NOTES:

* In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
I	AL/AX/EAX/RAX	imm8/16/32	NA	NA
MI	ModRM:r/m (w)	imm8/16/32	NA	NA
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

Description

Adds the source operand (second operand) and the carry (CF) flag, and subtracts the result from the destination operand (first operand). The result of the subtraction is stored in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) The state of the CF flag represents a borrow from a previous subtraction.

When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The SBB instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a borrow in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

The SBB instruction is usually executed as part of a multibyte or multiword subtraction in which a SUB instruction is followed by a SBB instruction.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

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. See the summary chart at the beginning of this section for encoding data and limits.

Operation

DEST ← (DEST - (SRC + CF));

Intel C/C++ Compiler Intrinsic Equivalent

SBB: extern unsigned char _subborrow_u8(unsigned char c_in, unsigned char src1, unsigned char src2, unsigned char *diff_out);

SBB: extern unsigned char _subborrow_u16(unsigned char c_in, unsigned short src1, unsigned short src2, unsigned short *diff_out);

SBB: extern unsigned char _subborrow_u32(unsigned char c_in, unsigned int src1, unsigned char int, unsigned int *diff_out);

SBB: extern unsigned char _subborrow_u64(unsigned char c_in, unsigned __int64 src1, unsigned __int64 src2, unsigned __int64 *diff_out);

Flags Affected

The OF, SF, ZF, AF, PF, and CF flags are set according to the result.

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 but the destination is not a memory operand.

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 but the destination is not a memory operand.

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 but the destination is not a memory operand.

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 but the destination is not a memory operand.

...

SCAS/SCASB/SCASW/SCASD—Scan String

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
AE	SCAS <i>m8</i>	NP	Valid	Valid	Compare AL with byte at ES:(E)DI or RDI, then set status flags.*
AF	SCAS <i>m16</i>	NP	Valid	Valid	Compare AX with word at ES:(E)DI or RDI, then set status flags.*
AF	SCAS <i>m32</i>	NP	Valid	Valid	Compare EAX with doubleword at ES:(E)DI or RDI then set status flags.*
REX.W + AF	SCAS <i>m64</i>	NP	Valid	N.E.	Compare RAX with quadword at RDI or EDI then set status flags.
AE	SCASB	NP	Valid	Valid	Compare AL with byte at ES:(E)DI or RDI then set status flags.*
AF	SCASW	NP	Valid	Valid	Compare AX with word at ES:(E)DI or RDI then set status flags.*
AF	SCASD	NP	Valid	Valid	Compare EAX with doubleword at ES:(E)DI or RDI then set status flags.*
REX.W + AF	SCASQ	NP	Valid	N.E.	Compare RAX with quadword at RDI or EDI then set status flags.

NOTES:

* In 64-bit mode, only 64-bit (RDI) and 32-bit (EDI) address sizes are supported. In non-64-bit mode, only 32-bit (EDI) and 16-bit (DI) address sizes are supported.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

In non-64-bit modes and in default 64-bit mode: this instruction compares a byte, word, doubleword or quadword specified using a memory operand with the value in AL, AX, or EAX. It then sets status flags in EFLAGS recording the results. The memory operand address is read from ES: (E)DI register (depending on the address-size attribute of the instruction and the current operational mode). Note that ES cannot be overridden with a segment override prefix.

At the assembly-code level, two forms of this instruction are allowed. The explicit-operand form and the no-operands form. The explicit-operand form (specified using the SCAS mnemonic) allows a memory operand to be specified explicitly. The memory operand must be a symbol that indicates the size and location of the operand value. The register operand is then automatically selected to match the size of the memory operand (AL register for byte comparisons, AX for word comparisons, EAX for doubleword comparisons). The explicit-operand form is provided to allow documentation. Note that the documentation provided by this form can be misleading. That is, the memory operand symbol must specify the correct type (size) of the operand (byte, word, or doubleword) but it does not have to specify the correct location. The location is always specified by ES: (E)DI.

The no-operands form of the instruction uses a short form of SCAS. Again, ES: (E)DI is assumed to be the memory operand and AL, AX, or EAX is assumed to be the register operand. The size of operands is selected by the mnemonic: SCASB (byte comparison), SCASW (word comparison), or SCASD (doubleword comparison).

After the comparison, the (E)DI register is incremented or decremented automatically according to the setting of the DF flag in the EFLAGS register. If the DF flag is 0, the (E)DI register is incremented; if the DF flag is 1, the

(E)DI register is decremented. The register is incremented or decremented by 1 for byte operations, by 2 for word operations, and by 4 for doubleword operations.

SCAS, SCASB, SCASW, SCASD, and SCASQ can be preceded by the REP prefix for block comparisons of ECX bytes, words, doublewords, or quadwords. Often, however, these instructions will be used in a LOOP construct that takes some action based on the setting of status flags. See “REP/REPE/REPZ /REPNE/REPZ—Repeat String Operation Prefix” in this chapter for a description of the REP prefix.

In 64-bit mode, the instruction’s default address size is 64-bits, 32-bit address size is supported using the prefix 67H. Using a REX prefix in the form of REX.W promotes operation on doubleword operand to 64 bits. The 64-bit no-operand mnemonic is SCASQ. Address of the memory operand is specified in either RDI or EDI, and AL/AX/EAX/RAX may be used as the register operand. After a comparison, the destination register is incremented or decremented by the current operand size (depending on the value of the DF flag). See the summary chart at the beginning of this section for encoding data and limits.

Operation

Non-64-bit Mode:

```
IF (Byte comparison)
  THEN
    temp ← AL – SRC;
    SetStatusFlags(temp);
    THEN IF DF = 0
      THEN (E)DI ← (E)DI + 1;
      ELSE (E)DI ← (E)DI - 1; FI;
  ELSE IF (Word comparison)
    THEN
      temp ← AX – SRC;
      SetStatusFlags(temp);
      IF DF = 0
        THEN (E)DI ← (E)DI + 2;
        ELSE (E)DI ← (E)DI - 2; FI;
    FI;
  ELSE IF (Doubleword comparison)
    THEN
      temp ← EAX – SRC;
      SetStatusFlags(temp);
      IF DF = 0
        THEN (E)DI ← (E)DI + 4;
        ELSE (E)DI ← (E)DI - 4; FI;
    FI;
  FI;
```

64-bit Mode:

```
IF (Byte comparison)
  THEN
    temp ← AL – SRC;
    SetStatusFlags(temp);
    THEN IF DF = 0
      THEN (R)E)DI ← (R)E)DI + 1;
      ELSE (R)E)DI ← (R)E)DI - 1; FI;
```



```

ELSE IF (Word comparison)
  THEN
    temp ← AX – SRC;
    SetStatusFlags(temp);
    IF DF = 0
      THEN (R|E)DI ← (R|E)DI + 2;
      ELSE (R|E)DI ← (R|E)DI – 2; FI;
  FI;
ELSE IF (Doubleword comparison)
  THEN
    temp ← EAX – SRC;
    SetStatusFlags(temp);
    IF DF = 0
      THEN (R|E)DI ← (R|E)DI + 4;
      ELSE (R|E)DI ← (R|E)DI – 4; FI;
  FI;
ELSE IF (Quadword comparison using REX.W )
  THEN
    temp ← RAX – SRC;
    SetStatusFlags(temp);
    IF DF = 0
      THEN (R|E)DI ← (R|E)DI + 8;
      ELSE (R|E)DI ← (R|E)DI – 8;
  FI;
FI;
F

```

Flags Affected

The OF, SF, ZF, AF, PF, and CF flags are set according to the temporary result of the comparison.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the limit of the ES segment. If the ES register contains a NULL segment selector.
#PF(fault-code)	If an illegal memory operand effective address in the ES segment is given.
#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

- #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.

...

SWAPGS—Swap GS Base Register

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F 01 F8	SWAPGS	NP	Valid	Invalid	Exchanges the current GS base register value with the value contained in MSR address C0000102H.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

SWAPGS exchanges the current GS base register value with the value contained in MSR address C0000102H (IA32_KERNEL_GS_BASE). The SWAPGS instruction is a privileged instruction intended for use by system software.

When using SYSCALL to implement system calls, there is no kernel stack at the OS entry point. Neither is there a straightforward method to obtain a pointer to kernel structures from which the kernel stack pointer could be read. Thus, the kernel cannot save general purpose registers or reference memory.

By design, SWAPGS does not require any general purpose registers or memory operands. No registers need to be saved before using the instruction. SWAPGS exchanges the CPL 0 data pointer from the IA32_KERNEL_GS_BASE MSR with the GS base register. The kernel can then use the GS prefix on normal memory references to access kernel data structures. Similarly, when the OS kernel is entered using an interrupt or exception (where the kernel stack is already set up), SWAPGS can be used to quickly get a pointer to the kernel data structures.

The IA32_KERNEL_GS_BASE MSR itself is only accessible using RDMSR/WRMSR instructions. Those instructions are only accessible at privilege level 0. The WRMSR instruction ensures that the IA32_KERNEL_GS_BASE MSR contains a canonical address.

Operation

IF CS.L \neq 1 (* Not in 64-Bit Mode *)

THEN

#UD; FI;

IF CPL \neq 0

THEN #GP(0); FI;

tmp \leftarrow GS.base;

GS.base \leftarrow IA32_KERNEL_GS_BASE;

IA32_KERNEL_GS_BASE \leftarrow tmp;

Flags Affected

None

Protected Mode Exceptions

#UD If Mode \neq 64-Bit.

Real-Address Mode Exceptions

#UD If Mode \neq 64-Bit.

Virtual-8086 Mode Exceptions

#UD If Mode \neq 64-Bit.

Compatibility Mode Exceptions

#UD If Mode \neq 64-Bit.

64-Bit Mode Exceptions

#GP(0) If CPL \neq 0.

If the LOCK prefix is used.

...

SYSCALL—Fast System Call

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 05	SYSCALL	NP	Valid	Invalid	Fast call to privilege level 0 system procedures.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

SYSCALL invokes an OS system-call handler at privilege level 0. It does so by loading RIP from the IA32_LSTAR MSR (after saving the address of the instruction following SYSCALL into RCX). (The WRMSR instruction ensures that the IA32_LSTAR MSR always contain a canonical address.)

SYSCALL also saves RFLAGS into R11 and then masks RFLAGS using the IA32_FMASK MSR (MSR address C0000084H); specifically, the processor clears in RFLAGS every bit corresponding to a bit that is set in the IA32_FMASK MSR.

SYSCALL loads the CS and SS selectors with values derived from bits 47:32 of the IA32_STAR MSR. However, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSCALL instruction does not ensure this correspondence.

The SYSCALL instruction does not save the stack pointer (RSP). If the OS system-call handler will change the stack pointer, it is the responsibility of software to save the previous value of the stack pointer. This might be done prior to executing SYSCALL, with software restoring the stack pointer with the instruction following SYSCALL (which will be executed after SYSRET). Alternatively, the OS system-call handler may save the stack pointer and restore it before executing SYSRET.

Operation

IF (CS.L \neq 1) or (IA32_EFER.LMA \neq 1) or (IA32_EFER.SCE \neq 1)
(* Not in 64-Bit Mode or SYSCALL/SYSRET not enabled in IA32_EFER *)

THEN #UD;

FI;

RCX \leftarrow RIP; (* Will contain address of next instruction *)

RIP \leftarrow IA32_LSTAR;

R11 \leftarrow RFLAGS;

RFLAGS \leftarrow RFLAGS AND NOT(IA32_FMASK);

CS.Selector \leftarrow IA32_STAR[47:32] AND FFFCH (* Operating system provides CS; RPL forced to 0 *)

(* Set rest of CS to a fixed value *)

CS.Base \leftarrow 0; (* Flat segment *)

CS.Limit \leftarrow FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)

CS.Type \leftarrow 11; (* Execute/read code, accessed *)

CS.S \leftarrow 1;

CS.DPL \leftarrow 0;

CS.P \leftarrow 1;

CS.L \leftarrow 1; (* Entry is to 64-bit mode *)

CS.D \leftarrow 0; (* Required if CS.L = 1 *)

CS.G \leftarrow 1; (* 4-KByte granularity *)

CPL \leftarrow 0;

SS.Selector \leftarrow IA32_STAR[47:32] + 8; (* SS just above CS *)

(* Set rest of SS to a fixed value *)

SS.Base \leftarrow 0; (* Flat segment *)

SS.Limit \leftarrow FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)

SS.Type \leftarrow 3; (* Read/write data, accessed *)

SS.S \leftarrow 1;

SS.DPL \leftarrow 0;

SS.P ← 1;
 SS.B ← 1; (* 32-bit stack segment *)
 SS.G ← 1; (* 4-KByte granularity *)

Flags Affected

All.

Protected Mode Exceptions

#UD The SYSCALL instruction is not recognized in protected mode.

Real-Address Mode Exceptions

#UD The SYSCALL instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The SYSCALL instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The SYSCALL instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If IA32_EFER.SCE = 0.
 If the LOCK prefix is used.

...

SYSENTER—Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F 34	SYSENTER	NP	Valid	Valid	Fast call to privilege level 0 system procedures.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Executes a fast call to a level 0 system procedure or routine. SYSENTER is a companion instruction to SYSEXIT. The instruction is optimized to provide the maximum performance for system calls from user code running at privilege level 3 to operating system or executive procedures running at privilege level 0.

When executed in IA-32e mode, the SYSENTER instruction transitions the logical processor to 64-bit mode; otherwise, the logical processor remains in protected mode.

Prior to executing the SYSENTER instruction, software must specify the privilege level 0 code segment and code entry point, and the privilege level 0 stack segment and stack pointer by writing values to the following MSRs:

- **IA32_SYSENTER_CS** (MSR address 174H) — The lower 16 bits of this MSR are the segment selector for the privilege level 0 code segment. This value is also used to determine the segment selector of the privilege level 0 stack segment (see the Operation section). This value cannot indicate a null selector.
- **IA32_SYSENTER_EIP** (MSR address 175H) — The value of this MSR is loaded into RIP (thus, this value references the first instruction of the selected operating procedure or routine). In protected mode, only bits 31:0 are loaded.
- **IA32_SYSENTER_ESP** (MSR address 176H) — The value of this MSR is loaded into RSP (thus, this value contains the stack pointer for the privilege level 0 stack). This value cannot represent a non-canonical address. In protected mode, only bits 31:0 are loaded.

These MSRs can be read from and written to using RDMSR/WRMSR. The WRMSR instruction ensures that the IA32_SYSENTER_EIP and IA32_SYSENTER_ESP MSRs always contain canonical addresses.

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

The SYSENTER instruction can be invoked from all operating modes except real-address mode.

The SYSENTER and SYSEXIT instructions are companion instructions, but they do not constitute a call/return pair. When executing a SYSENTER instruction, the processor does not save state information for the user code (e.g., the instruction pointer), and neither the SYSENTER nor the SYSEXIT instruction supports passing parameters on the stack.

To use the SYSENTER and SYSEXIT instructions as companion instructions for transitions between privilege level 3 code and privilege level 0 operating system procedures, the following conventions must be followed:

- The segment descriptors for the privilege level 0 code and stack segments and for the privilege level 3 code and stack segments must be contiguous in a descriptor table. This convention allows the processor to compute the segment selectors from the value entered in the SYSENTER_CS_MSR MSR.
- The fast system call “stub” routines executed by user code (typically in shared libraries or DLLs) must save the required return IP and processor state information if a return to the calling procedure is required. Likewise, the operating system or executive procedures called with SYSENTER instructions must have access to and use this saved return and state information when returning to the user code.

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

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

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

Operation

```
IF CRO.PE = 0 OR IA32_SYSENTER_CS[15:2] = 0 THEN #GP(0); FI;

RFLAGS.VM ← 0; (* Ensures protected mode execution *)
RFLAGS.IF ← 0; (* Mask interrupts *)
IF in IA-32e mode
  THEN
    RSP ← IA32_SYSENTER_ESP;
    RIP ← IA32_SYSENTER_EIP;
  ELSE
    ESP ← IA32_SYSENTER_ESP[31:0];
    EIP ← IA32_SYSENTER_EIP[31:0];
FI;

CS.Selector ← IA32_SYSENTER_CS[15:0] AND FFFCH;
(* Operating system provides CS; RPL forced to 0 *)

(* Set rest of CS to a fixed value *)
CS.Base ← 0; (* Flat segment *)
CS.Limit ← FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)
CS.Type ← 11; (* Execute/read code, accessed *)
CS.S ← 1;
CS.DPL ← 0;
CS.P ← 1;
IF in IA-32e mode
  THEN
    CS.L ← 1; (* Entry is to 64-bit mode *)
    CS.D ← 0; (* Required if CS.L = 1 *)
  ELSE
    CS.L ← 0;
    CS.D ← 1; (* 32-bit code segment*)
FI;
CS.G ← 1; (* 4-KByte granularity *)
CPL ← 0;

SS.Selector ← CS.Selector + 8; (* SS just above CS *)
(* Set rest of SS to a fixed value *)
SS.Base ← 0; (* Flat segment *)
SS.Limit ← FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)
SS.Type ← 3; (* Read/write data, accessed *)
SS.S ← 1;
SS.DPL ← 0;
SS.P ← 1;
SS.B ← 1; (* 32-bit stack segment*)
SS.G ← 1; (* 4-KByte granularity *)
```

Flags Affected

VM, IF (see Operation above)

Protected Mode Exceptions

#GP(0) If IA32_SYSENTER_CS[15:2] = 0.

#UD If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP The SYSENTER instruction is not recognized in real-address mode.

#UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

SYSEXIT—Fast Return from Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F 35	SYSEXIT	NP	Valid	Valid	Fast return to privilege level 3 user code.
REX.W + 0F 35	SYSEXIT	NP	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
NP	NA	NA	NA	NA

Description

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

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

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

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

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

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

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

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

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

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

Operation

```
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;

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

Same exceptions as in protected mode.

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.
...

```

SYSRET—Return From Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F 07	SYSRET	NP	Valid	Invalid	Return to compatibility mode from fast system call
REX.W + 0F 07	SYSRET	NP	Valid	Invalid	Return to 64-bit mode from fast system call

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

SYSRET is a companion instruction to the SYSCALL instruction. It returns from an OS system-call handler to user code at privilege level 3. It does so by loading RIP from RCX and loading RFLAGS from R11.¹ With a 64-bit operand size, SYSRET remains in 64-bit mode; otherwise, it enters compatibility mode and only the low 32 bits of the registers are loaded.

SYSRET loads the CS and SS selectors with values derived from bits 63:48 of the IA32_STAR MSR. However, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSRET instruction does not ensure this correspondence.

The SYSRET instruction does not modify the stack pointer (ESP or RSP). For that reason, it is necessary for software to switch to the user stack. The OS may load the user stack pointer (if it was saved after SYSCALL) before executing SYSRET; alternatively, user code may load the stack pointer (if it was saved before SYSCALL) after receiving control from SYSRET.

If the OS loads the stack pointer before executing SYSRET, it must ensure that the handler of any interrupt or exception delivered between restoring the stack pointer and successful execution of SYSRET is not invoked with the user stack. It can do so using approaches such as the following:

- External interrupts. The OS can prevent an external interrupt from being delivered by clearing EFLAGS.IF before loading the user stack pointer.
- Nonmaskable interrupts (NMIs). The OS can ensure that the NMI handler is invoked with the correct stack by using the interrupt stack table (IST) mechanism for gate 2 (NMI) in the IDT (see Section 6.14.5, “Interrupt Stack Table,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*).
- General-protection exceptions (#GP). The SYSRET instruction generates #GP(0) if the value of RCX is not canonical. The OS can address this possibility using one or more of the following approaches:
 - Confirming that the value of RCX is canonical before executing SYSRET.
 - Using paging to ensure that the SYSCALL instruction will never save a non-canonical value into RCX.
 - Using the IST mechanism for gate 13 (#GP) in the IDT.

1. Regardless of the value of R11, the RF and VM flags are always 0 in RFLAGS after execution of SYSRET. In addition, all reserved bits in RFLAGS retain the fixed values.

Operation

```
IF (CS.L ≠ 1 ) or (IA32_EFER.LMA ≠ 1) or (IA32_EFER.SCE ≠ 1)
(* Not in 64-Bit Mode or SYSCALL/SYSRET not enabled in IA32_EFER *)
  THEN #UD; FI;
IF (CPL ≠ 0) OR (RCX is not canonical) THEN #GP(0); FI;

IF (operand size is 64-bit)
  THEN (* Return to 64-Bit Mode *)
    RIP ← RCX;
  ELSE (* Return to Compatibility Mode *)
    RIP ← ECX;
FI;
RFLAGS ← (R11 & 3C7FD7H) | 2;          (* Clear RF, VM, reserved bits; set bit 2 *)

IF (operand size is 64-bit)
  THEN CS.Selector ← IA32_STAR[63:48]+16;
  ELSE CS.Selector ← IA32_STAR[63:48];
FI;
CS.Selector ← CS.Selector OR 3;        (* RPL forced to 3 *)
(* Set rest of CS to a fixed value *)
CS.Base ← 0;                          (* Flat segment *)
CS.Limit ← FFFFFFFH;                  (* With 4-KByte granularity, implies a 4-GByte limit *)
CS.Type ← 11;                          (* Execute/read code, accessed *)
CS.S ← 1;
CS.DPL ← 3;
CS.P ← 1;
IF (operand size is 64-bit)
  THEN (* Return to 64-Bit Mode *)
    CS.L ← 1;                          (* 64-bit code segment *)
    CS.D ← 0;                          (* Required if CS.L = 1 *)
  ELSE (* Return to Compatibility Mode *)
    CS.L ← 0;                          (* Compatibility mode *)
    CS.D ← 1;                          (* 32-bit code segment *)
FI;
CS.G ← 1;                              (* 4-KByte granularity *)
CPL ← 0;

SS.Selector ← (IA32_STAR[63:48]+8) OR 3; (* RPL forced to 3 *)
(* Set rest of SS to a fixed value *)
SS.Base ← 0;                          (* Flat segment *)
SS.Limit ← FFFFFFFH;                  (* With 4-KByte granularity, implies a 4-GByte limit *)
SS.Type ← 3;                          (* Read/write data, accessed *)
SS.S ← 1;
SS.DPL ← 3;
SS.P ← 1;
SS.B ← 1;                              (* 32-bit stack segment*)
SS.G ← 1;                              (* 4-KByte granularity *)
```

Flags Affected

All.

Protected Mode Exceptions

#UD The SYSRET instruction is not recognized in protected mode.

Real-Address Mode Exceptions

#UD The SYSRET instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The SYSRET instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The SYSRET instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If IA32_EFER.SCE = 0.

If the LOCK prefix is used.

#GP(0) If CPL ≠ 0.

If RCX contains a non-canonical address.

...

UCOMISD—Unordered Compare Scalar Double-Precision Floating-Point Values and Set EFLAGS

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 2E /r UCOMISD <i>xmm1</i> , <i>xmm2/m64</i>	RM	V/V	SSE2	Compares (unordered) the low double-precision floating-point values in <i>xmm1</i> and <i>xmm2/m64</i> and set the EFLAGS accordingly.
VEX.LIG.66.0F.WIG 2E /r VUCOMISD <i>xmm1</i> , <i>xmm2/m64</i>	RM	V/V	AVX	Compare low double precision floating-point values in <i>xmm1</i> and <i>xmm2/mem64</i> and set the EFLAGS flags accordingly.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r)	ModRM:r/m (r)	NA	NA

Description

Performs an unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN).

Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location.

The UCOMISD instruction differs from the COMISD instruction in that it signals a SIMD floating-point invalid operation exception (#1) only when a source operand is an SNaN. The COMISD instruction signals an invalid operation exception if a source operand is either a QNaN or an SNaN.

The EFLAGS register is not updated if an unmasked SIMD floating-point exception is generated.

In 64-bit mode, using a REX prefix in the form of REX.R permits this instruction to access additional registers (XMM8-XMM15).

Note: In VEX-encoded versions, VEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

Operation

RESULT ← UnorderedCompare(SRC1[63:0] < > SRC2[63:0]) {

(* Set EFLAGS *)

CASE (RESULT) OF

UNORDERED: ZF, PF, CF ← 111;

GREATER_THAN: ZF, PF, CF ← 000;

LESS_THAN: ZF, PF, CF ← 001;

EQUAL: ZF, PF, CF ← 100;

ESAC;

OF, AF, SF ← 0;

Intel C/C++ Compiler Intrinsic Equivalent

int __mm_ucomieq_sd(__m128d a, __m128d b)

int __mm_ucomilt_sd(__m128d a, __m128d b)

int __mm_ucomile_sd(__m128d a, __m128d b)

int __mm_ucomigt_sd(__m128d a, __m128d b)

int __mm_ucomige_sd(__m128d a, __m128d b)

int __mm_ucomineq_sd(__m128d a, __m128d b)

SIMD Floating-Point Exceptions

Invalid (if SNaN operands), Denormal.

Other Exceptions

See Exceptions Type 3; additionally

#UD If VEX.vvvv != 1111B.

...

WRMSR—Write to Model Specific Register

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 30	WRMSR	NP	Valid	Valid	Write the value in EDX:EAX to MSR specified by ECX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Writes the contents of registers EDX:EAX into the 64-bit model specific register (MSR) specified in the ECX register. (On processors that support the Intel 64 architecture, the high-order 32 bits of RCX are ignored.) The contents of the EDX register are copied to high-order 32 bits of the selected MSR and the contents of the EAX register are copied to low-order 32 bits of the MSR. (On processors that support the Intel 64 architecture, the high-order 32 bits of each of RAX and RDX are ignored.) Undefined or reserved bits in an MSR should be set to values previously read.

This instruction must be executed at privilege level 0 or in real-address mode; otherwise, a general protection exception #GP(0) is generated. Specifying a reserved or unimplemented MSR address in ECX will also cause a general protection exception. The processor will also generate a general protection exception if software attempts to write to bits in a reserved MSR.

When the WRMSR instruction is used to write to an MTRR, the TLBs are invalidated. This includes global entries (see “Translation Lookaside Buffers (TLBs)” in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*).

MSRs control functions for testability, execution tracing, performance-monitoring and machine check errors. Chapter 35, “Model-Specific Registers (MSRs)”, in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*, lists all MSRs that can be written with this instruction and their addresses. Note that each processor family has its own set of MSRs.

The WRMSR instruction is a serializing instruction (see “Serializing Instructions” in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). Note that WRMSR to the IA32_TSC_DEADLINE MSR (MSR index 6E0H) and the X2APIC MSRs (MSR indices 802H to 83FH) are not serializing.

The CPUID instruction should be used to determine whether MSRs are supported (CPUID.01H:EDX[5] = 1) before using this instruction.

IA-32 Architecture Compatibility

The MSRs and the ability to read them with the WRMSR instruction were introduced into the IA-32 architecture with the Pentium processor. Execution of this instruction by an IA-32 processor earlier than the Pentium processor results in an invalid opcode exception #UD.

Operation

MSR[ECX] ← EDX:EAX;

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the value in ECX specifies a reserved or unimplemented MSR address. If the value in EDX:EAX sets bits that are reserved in the MSR specified by ECX. If the source register contains a non-canonical address and ECX specifies one of the following MSRs: IA32_DS_AREA, IA32_FS_BASE, IA32_GS_BASE, IA32_KERNEL_GS_BASE, IA32_LSTAR, IA32_SYSENTER_EIP, IA32_SYSENTER_ESP.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If the value in ECX specifies a reserved or unimplemented MSR address.
-----	--

If the value in EDX:EAX sets bits that are reserved in the MSR specified by ECX.

If the source register contains a non-canonical address and ECX specifies one of the following MSRs: IA32_DS_AREA, IA32_FS_BASE, IA32_GS_BASE, IA32_KERNEL_GS_BASE, IA32_LSTAR, IA32_SYSENTER_EIP, IA32_SYSENTER_ESP.

#UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

8. Updates to Chapter 5, Volume 2C

Change bars show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference*.

...

GETSEC[ENTERACCS] - Execute Authenticated Chipset Code

Opcode	Instruction	Description
OF 37 (EAX = 2)	GETSEC[ENTERACCS]	Enter authenticated code execution mode. EBX holds the authenticated code module physical base address. ECX holds the authenticated code module size (bytes).

Description

The GETSEC[ENTERACCS] function loads, authenticates and executes an authenticated code module using an Intel® TXT platform chipset's public key. The ENTERACCS leaf of GETSEC is selected with EAX set to 2 at entry.

There are certain restrictions enforced by the processor for the execution of the GETSEC[ENTERACCS] instruction:

- Execution is not allowed unless the processor is in protected mode or IA-32e mode with CPL = 0 and EFLAGS.VM = 0.
- Processor cache must be available and not disabled, that is, CR0.CD and CR0.NW bits must be 0.
- For processor packages containing more than one logical processor, CR0.CD is checked to ensure consistency between enabled logical processors.
- For enforcing consistency of operation with numeric exception reporting using Interrupt 16, CR0.NE must be set.

- An Intel TXT-capable chipset must be present as communicated to the processor by sampling of the power-on configuration capability field after reset.
- The processor can not already be in authenticated code execution mode as launched by a previous GETSEC[ENTERACCS] or GETSEC[SENDER] instruction without a subsequent exiting using GETSEC[EXITAC]).
- To avoid potential operability conflicts between modes, the processor is not allowed to execute this instruction if it currently is in SMM or VMX operation.
- To insure consistent handling of SIPI messages, the processor executing the GETSEC[ENTERACCS] instruction must also be designated the BSP (boot-strap processor) as defined by A32_APIC_BASE.BSP (Bit 8).

Failure to conform to the above conditions results in the processor signaling a general protection exception.

Prior to execution of the ENTERACCS leaf, other logical processors, i.e. RLPs, in the platform must be:

- idle in a wait-for-SIPI state (as initiated by an INIT assertion or through reset for non-BSP designated processors), or
- in the SENTER sleep state as initiated by a GETSEC[SENDER] from the initiating logical processor (ILP).

If other logical processor(s) in the same package are not idle in one of these states, execution of ENTERACCS signals a general protection exception. The same requirement and action applies if the other logical processor(s) of the same package do not have CRO.CD = 0.

A successful execution of ENTERACCS results in the ILP entering an authenticated code execution mode. Prior to reaching this point, the processor performs several checks. These include:

- Establish and check the location and size of the specified authenticated code module to be executed by the processor.
- Inhibit the ILP's response to the external events: INIT, A20M, NMI and SMI.
- Broadcast a message to enable protection of memory and I/O from other processor agents.
- Load the designated code module into an authenticated code execution area.
- Isolate the contents of the authenticated code execution area from further state modification by external agents.
- Authenticate the authenticated code module.
- Initialize the initiating logical processor state based on information contained in the authenticated code module header.
- Unlock the Intel® TXT-capable chipset private configuration space and TPM locality 3 space.
- Begin execution in the authenticated code module at the defined entry point.

The GETSEC[ENTERACCS] function requires two additional input parameters in the general purpose registers EBX and ECX. EBX holds the authenticated code (AC) module physical base address (the AC module must reside below 4 GBytes in physical address space) and ECX holds the AC module size (in bytes). The physical base address and size are used to retrieve the code module from system memory and load it into the internal authenticated code execution area. The base physical address is checked to verify it is on a modulo-4096 byte boundary. The size is verified to be a multiple of 64, that it does not exceed the internal authenticated code execution area capacity (as reported by GETSEC[CAPABILITIES]), and that the top address of the AC module does not exceed 32 bits. An error condition results in an abort of the authenticated code execution launch and the signaling of a general protection exception.

As an integrity check for proper processor hardware operation, execution of GETSEC[ENTERACCS] will also check the contents of all the machine check status registers (as reported by the MSRs IA32_MCi_STATUS) for any valid uncorrectable error condition. In addition, the global machine check status register IA32_MCG_STATUS MCIP bit must be cleared and the IERR processor package pin (or its equivalent) must not be asserted, indicating that no machine check exception processing is currently in progress. These checks are performed prior to initiating the load of the authenticated code module. Any outstanding valid uncorrectable machine check error condition present in these status registers at this point will result in the processor signaling a general protection violation.

The ILP masks the response to the assertion of the external signals INIT#, A20M, NMI#, and SMI#. This masking remains active until optionally unmasked by GETSEC[EXITAC] (this defined unmasking behavior assumes GETSEC[ENTERACCS] was not executed by a prior GETSEC[SENDER]). The purpose of this masking control is to prevent exposure to existing external event handlers that may not be under the control of the authenticated code module.

The ILP sets an internal flag to indicate it has entered authenticated code execution mode. The state of the A20M pin is likewise masked and forced internally to a de-asserted state so that any external assertion is not recognized during authenticated code execution mode.

To prevent other (logical) processors from interfering with the ILP operating in authenticated code execution mode, memory (excluding implicit write-back transactions) access and I/O originating from other processor agents are blocked. This protection starts when the ILP enters into authenticated code execution mode. Only memory and I/O transactions initiated from the ILP are allowed to proceed. Exiting authenticated code execution mode is done by executing GETSEC[EXITAC]. The protection of memory and I/O activities remains in effect until the ILP executes GETSEC[EXITAC].

Prior to launching the authenticated execution module using GETSEC[ENTERACCS] or GETSEC[SENDER], the processor's MTRRs (Memory Type Range Registers) must first be initialized to map out the authenticated RAM addresses as WB (writeback). Failure to do so may affect the ability for the processor to maintain isolation of the loaded authenticated code module. If the processor detected this requirement is not met, it will signal an Intel® TXT reset condition with an error code during the loading of the authenticated code module.

While physical addresses within the load module must be mapped as WB, the memory type for locations outside of the module boundaries must be mapped to one of the supported memory types as returned by GETSEC[PARAMETERS] (or UC as default).

To conform to the minimum granularity of MTRR MSR for specifying the memory type, authenticated code RAM (ACRAM) is allocated to the processor in 4096 byte granular blocks. If an AC module size as specified in ECX is not a multiple of 4096 then the processor will allocate up to the next 4096 byte boundary for mapping as ACRAM with indeterminate data. This pad area will not be visible to the authenticated code module as external memory nor can it depend on the value of the data used to fill the pad area.

At the successful completion of GETSEC[ENTERACCS], the architectural state of the processor is partially initialized from contents held in the header of the authenticated code module. The processor GDTR, CS, and DS selectors are initialized from fields within the authenticated code module. Since the authenticated code module must be relocatable, all address references must be relative to the authenticated code module base address in EBX. The processor GDTR base value is initialized to the AC module header field GDTRBasePtr + module base address held in EBX and the GDTR limit is set to the value in the GDTRLimit field. The CS selector is initialized to the AC module header SegSel field, while the DS selector is initialized to CS + 8. The segment descriptor fields are implicitly initialized to BASE=0, LIMIT=FFFFh, G=1, D=1, P=1, S=1, read/write access for DS, and execute/read access for CS. The processor begins the authenticated code module execution with the EIP set to the AC module header EntryPoint field + module base address (EBX). The AC module based fields used for initializing the processor state are checked for consistency and any failure results in a shutdown condition.

A summary of the register state initialization after successful completion of GETSEC[ENTERACCS] is given for the processor in Table 5-4. The paging is disabled upon entry into authenticated code execution mode. The authenticated code module is loaded and initially executed using physical addresses. It is up to the system software after execution of GETSEC[ENTERACCS] to establish a new (or restore its previous) paging environment with an appropriate mapping to meet new protection requirements. EBP is initialized to the authenticated code module base physical address for initial execution in the authenticated environment. As a result, the authenticated code can reference EBP for relative address based references, given that the authenticated code module must be position independent.

Table 5-4 Register State Initialization after GETSEC[ENTERACCS]

Register State	Initialization Status	Comment
CR0	PG←0, AM←0, WP←0: Others unchanged	Paging, Alignment Check, Write-protection are disabled
CR4	MCE←0: Others unchanged	Machine Check Exceptions Disabled
EFLAGS	00000002H	
IA32_EFER	0H	IA-32e mode disabled
EIP	AC.base + EntryPoint	AC.base is in EBX as input to GETSEC[ENTERACCS]
[E R]BX	Pre-ENTERACCS state: Next [E R]IP prior to GETSEC[ENTERACCS]	Carry forward 64-bit processor state across GETSEC[ENTERACCS]
ECX	Pre-ENTERACCS state: [31:16]=GDTR.limit; [15:0]=CS.sel	Carry forward processor state across GETSEC[ENTERACCS]
[E R]DX	Pre-ENTERACCS state: GDTR base	Carry forward 64-bit processor state across GETSEC[ENTERACCS]
EBP	AC.base	
CS	Sel=[SegSel], base=0, limit=FFFFFh, G=1, D=1, AR=9BH	
DS	Sel=[SegSel] +8, base=0, limit=FFFFFh, G=1, D=1, AR=93H	
GDTR	Base= AC.base (EBX) + [GDTBasePtr], Limit=[GDTLimit]	
DR7	00000400H	
IA32_DEBUGCTL	0H	
IA32_MISC_ENABLE	see Table 5-5 for example	The number of initialized fields may change due to processor implementation

The segmentation related processor state that has not been initialized by GETSEC[ENTERACCS] requires appropriate initialization before use. Since a new GDT context has been established, the previous state of the segment selector values held in ES, SS, FS, GS, TR, and LDTR might not be valid.

The MSR IA32_EFER is also unconditionally cleared as part of the processor state initialized by ENTERACCS. Since paging is disabled upon entering authenticated code execution mode, a new paging environment will have to be reestablished in order to establish IA-32e mode while operating in authenticated code execution mode.

Debug exception and trap related signaling is also disabled as part of GETSEC[ENTERACCS]. This is achieved by resetting DR7, TF in EFLAGS, and the MSR IA32_DEBUGCTL. These debug functions are free to be re-enabled once supporting exception handler(s), descriptor tables, and debug registers have been properly initialized following entry into authenticated code execution mode. Also, any pending single-step trap condition will have been cleared upon entry into this mode.

The IA32_MISC_ENABLE MSR is initialized upon entry into authenticated execution mode. Certain bits of this MSR are preserved because preserving these bits may be important to maintain previously established platform settings (See the footnote for Table 5-5.). The remaining bits are cleared for the purpose of establishing a more consistent environment for the execution of authenticated code modules. One of the impacts of initializing this MSR is any previous condition established by the MONITOR instruction will be cleared.

To support the possible return to the processor architectural state prior to execution of GETSEC[ENTERACCS], certain critical processor state is captured and stored in the general-purpose registers at instruction completion. [E|R]BX holds effective address ([E|R]IP) of the instruction that would execute next after GETSEC[ENTERACCS], ECX[15:0] holds the CS selector value, ECX[31:16] holds the GDTR limit field, and [E|R]DX holds the GDTR base field. The subsequent authenticated code can preserve the contents of these registers so that this state can be manually restored if needed, prior to exiting authenticated code execution mode with GETSEC[EXITAC]. For the processor state after exiting authenticated code execution mode, see the description of GETSEC[SEXIT].

Table 5-5 IA32_MISC_ENABLE MSR Initialization¹ by ENTERACCS and SENTER

Field	Bit position	Description
Fast strings enable	0	Clear to 0
FOPCODE compatibility mode enable	2	Clear to 0
Thermal monitor enable	3	Set to 1 if other thermal monitor capability is not enabled. ²
Split-lock disable	4	Clear to 0
Bus lock on cache line splits disable	8	Clear to 0
Hardware prefetch disable	9	Clear to 0
GV1/2 legacy enable	15	Clear to 0
MONITOR/MWAIT s/m enable	18	Clear to 0
Adjacent sector prefetch disable	19	Clear to 0

NOTES:

1. The number of IA32_MISC_ENABLE fields that are initialized may vary due to processor implementations.
2. ENTERACCS (and SENTER) initialize the state of processor thermal throttling such that at least a minimum level is enabled. If thermal throttling is already enabled when executing one of these GETSEC leaves, then no change in the thermal throttling control settings will occur. If thermal throttling is disabled, then it will be enabled via setting of the thermal throttle control bit 3 as a result of executing these GETSEC leaves.

The IDTR will also require reloading with a new IDT context after entering authenticated code execution mode, before any exceptions or the external interrupts INTR and NMI can be handled. Since external interrupts are re-enabled at the completion of authenticated code execution mode (as terminated with EXITAC), it is recommended that a new IDT context be established before this point. Until such a new IDT context is established, the programmer must take care in not executing an INT n instruction or any other operation that would result in an exception or trap signaling.

Prior to completion of the GETSEC[ENTERACCS] instruction and after successful authentication of the AC module, the private configuration space of the Intel TXT chipset is unlocked. The authenticated code module alone can gain access to this normally restricted chipset state for the purpose of securing the platform.

Once the authenticated code module is launched at the completion of GETSEC[ENTERACCS], it is free to enable interrupts by setting EFLAGS.IF and enable NMI by execution of IRET. This presumes that it has re-established interrupt handling support through initialization of the IDT, GDT, and corresponding interrupt handling code.

Operation in a Uni-Processor Platform

(* The state of the internal flag ACMODEFLAG persists across instruction boundary *)

```
IF (CR4.SMXE=0)
    THEN #UD;
ELSIF (in VMX non-root operation)
    THEN VM Exit (reason="GETSEC instruction");
```

```

ELSIF (GETSEC leaf unsupported)
    THEN #UD;
ELSIF ((in VMX operation) or
    (CRO.PE=0) or (CRO.CD=1) or (CRO.NW=1) or (CRO.NE=0) or
    (CPL>0) or (EFLAGS.VM=1) or
    (IA32_APIC_BASE.BSP=0) or
    (TXT chipset not present) or
    (ACMODEFLAG=1) or (IN_SMM=1))
    THEN #GP(0);
IF (GETSEC[PARAMETERS].Parameter_Type = 5, MCA_Handling (bit 6) = 0)
    FOR I = 0 to IA32_MCG_CAP.COUNT-1 DO
        IF (IA32_MC[I]_STATUS = uncorrectable error)
            THEN #GP(0);
    OD;
FI;
IF (IA32_MCG_STATUS.MCIP=1) or (IERR pin is asserted)
    THEN #GP(0);
ACBASE← EBX;
ACSIZE← ECX;
IF (((ACBASE MOD 4096) != 0) or ((ACSIZE MOD 64) != 0) or (ACSIZE < minimum module size) OR (ACSIZE > authenticated RAM
capacity)) or ((ACBASE+ACSIZE) > (2^32 -1)))
    THEN #GP(0);
IF (secondary thread(s) CRO.CD = 1) or ((secondary thread(s) NOT(wait-for-SIPI)) and
    (secondary thread(s) not in SENTER sleep state)
    THEN #GP(0);
Mask SMI, INIT, A2OM, and NMI external pin events;
IA32_MISC_ENABLE← (IA32_MISC_ENABLE & MASK_CONST*)
(* The hexadecimal value of MASK_CONST may vary due to processor implementations *)
A2OM← 0;
IA32_DEBUGCTL← 0;
Invalidate processor TLB(s);
Drain Outgoing Transactions;
ACMODEFLAG← 1;
SignalTXTMessage(ProcessorHold);
Load the internal ACRAM based on the AC module size;
(* Ensure that all ACRAM loads hit Write Back memory space *)
IF (ACRAM memory type != WB)
    THEN TXT-SHUTDOWN(#BadACMMType);
IF (AC module header version isnot supported) OR (ACRAM[ModuleType] <> 2)
    THEN TXT-SHUTDOWN(#UnsupportedACM);
(* Authenticate the AC Module and shutdown with an error if it fails *)
KEY← GETKEY(ACRAM, ACBASE);
KEYHASH← HASH(KEY);
CSKEYHASH← READ(TXT.PUBLIC.KEY);
IF (KEYHASH <> CSKEYHASH)
    THEN TXT-SHUTDOWN(#AuthenticateFail);
SIGNATURE← DECRYPT(ACRAM, ACBASE, KEY);
(* The value of SIGNATURE_LEN_CONST is implementation-specific*)
FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
    ACRAM[SCRATCH.I]← SIGNATURE[I];

```

```

COMPUTEDSIGNATURE← HASH(ACRAM, ACBASE, ACSIZE);
FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
    ACRAM[SCRATCH.SIGNATURE_LEN_CONST+I]← COMPUTEDSIGNATURE[I];
IF (SIGNATURE<>COMPUTEDSIGNATURE)
    THEN TXT-SHUTDOWN(#AuthenticateFail);
ACMCONTROL← ACRAM[CodeControl];
IF ((ACMCONTROL.0 = 0) and (ACMCONTROL.1 = 1) and (snoop hit to modified line detected on ACRAM load))
    THEN TXT-SHUTDOWN(#UnexpectedHITM);
IF (ACMCONTROL reserved bits are set)
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[GDTBasePtr] < (ACRAM[HeaderLen] * 4 + Scratch_size)) OR
    ((ACRAM[GDTBasePtr] + ACRAM[GDTLimit]) >= ACSIZE))
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACMCONTROL.0 = 1) and (ACMCONTROL.1 = 1) and (snoop hit to modified line detected on ACRAM load))
    THEN ACRAM[EntryPoint]← ACBASE+ACRAM[ErrorEntryPoint];
ELSE
    ACRAM[EntryPoint]← ACBASE+ACRAM[EntryPoint];
IF ((ACRAMEnterPoint >= ACSIZE) OR (ACRAMEnterPoint < (ACRAM[HeaderLen] * 4 + Scratch_size)))THEN TXT-SHUTDOWN(#BadACMFormat);
IF (ACRAM[GDTLimit] & FFFF0000h)
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[SegSel] > (ACRAM[GDTLimit] - 15)) OR (ACRAM[SegSel] < 8))
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[SegSel].TI=1) OR (ACRAM[SegSel].RPLI=0))
    THEN TXT-SHUTDOWN(#BadACMFormat);
CRO.[PG.AM.WP]← 0;
CR4.MCE← 0;
EFLAGS← 00000002h;
IA32_EFER← 0h;
[EIP]BX← [EIP]IP of the instruction after GETSEC[ENTERACCS];
ECX← Pre-GETSEC[ENTERACCS] GDT.limit:CS.sel;
[EIP]DX← Pre-GETSEC[ENTERACCS] GDT.base;
EBP← ACBASE;
GDTR.BASE← ACBASE+ACRAM[GDTBasePtr];
GDTR.LIMIT← ACRAM[GDTLimit];
CS.SEL← ACRAM[SegSel];
CS.BASE← 0;
CS.LIMIT← FFFFFFFh;
CS.G← 1;
CS.D← 1;
CS.AR← 9Bh;
DS.SEL← ACRAM[SegSel]+8;
DS.BASE← 0;
DS.LIMIT← FFFFFFFh;
DS.G← 1;
DS.D← 1;
DS.AR← 93h;
DR7← 00000400h;
IA32_DEBUGCTL← 0;
SignalTXTMsg(OpenPrivate);
SignalTXTMsg(OpenLocality3);

```

EIP ← ACEntryPoint;
END;

Flags Affected

All flags are cleared.

Use of Prefixes

LOCK	Causes #UD
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ)
Operand size	Causes #UD
Segment overrides	Ignored
Address size	Ignored
REX	Ignored

Protected Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.CD = 1 or CR0.NW = 1 or CR0.NE = 0 or CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1. If a Intel® TXT-capable chipset is not present. If in VMX root operation. If the initiating processor is not designated as the bootstrap processor via the MSR bit IA32_APIC_BASE.BSP. If the processor is already in authenticated code execution mode. If the processor is in SMM. If a valid uncorrectable machine check error is logged in IA32_MC[I]_STATUS. If the authenticated code base is not on a 4096 byte boundary. If the authenticated code size > processor internal authenticated code area capacity. If the authenticated code size is not modulo 64. If other enabled logical processor(s) of the same package CR0.CD = 1. If other enabled logical processor(s) of the same package are not in the wait-for-SIPI or SENTER sleep state.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[ENTERACCS] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[ENTERACCS] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

#GP IF AC code module does not reside in physical address below $2^{32} - 1$.

64-Bit Mode Exceptions

All protected mode exceptions apply.

#GP IF AC code module does not reside in physical address below $2^{32} - 1$.

VM-exit Condition

Reason (GETSEC) IF in VMX non-root operation.

...

9. Updates to Appendix A, Volume 2C

Change bars show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference*.

Table A-3 Two-byte Opcode Map: 00H – 77H (First Byte is 0FH) *

	px	0	1	2	3	4	5	6	7
0		Grp 6 ^{1A}	Grp 7 ^{1A}	LAR Gv, Ew	LSL Gv, Ew		SYSCALL ⁰⁶⁴	CLTS	SYSRET ⁰⁶⁴
1		vmovups Vps, Wps	vmovups Wps, Vps	vmovlps Vq, Hq, Mq vmovhlps Vq, Hq, Uq	vmovlps Mq, Vq	vunpcklps Vx, Hx, Wx	vunpckhps Vx, Hx, Wx	vmovhps ^{v1} Vdq, Hq, Mq vmovhlps Vdq, Hq, Uq	vmovhps ^{v1} Mq, Vq
	66	vmovupd Vpd, Wpd	vmovupd Wpd, Vpd	vmovlpd Vq, Hq, Mq	vmovlpd Mq, Vq	vunpcklpd Vx, Hx, Wx	vunpckhpd Vx, Hx, Wx	vmovhpd ^{v1} Vdq, Hq, Mq	vmovhpd ^{v1} Mq, Vq
	F3	vmovss Vx, Hx, Wss	vmovss Wss, Hx, Vss	vmovsldup Vx, Wx				vmovshdup Vx, Wx	
	F2	vmovsd Vx, Hx, Wsd	vmovsd Wsd, Hx, Vsd	vmovddup Vx, Wx					
2		MOV Rd, Cd	MOV Rd, Dd	MOV Cd, Rd	MOV Dd, Rd				
3		WRMSR	RDTSC	RDMSR	RDPMSR	SYSENTER	SYSEXIT		GETSEC
4		CMOVcc, (Gv, Ev) - Conditional Move							
		O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE

	pxf	0	1	2	3	4	5	6	7
5		vmovmskps Gy, Ups	vsqrtps Vps, Wps	vrsqrtps Vps, Wps	vrcpps Vps, Wps	vandps Vps, Hps, Wps	vandnps Vps, Hps, Wps	vorps Vps, Hps, Wps	vxorps Vps, Hps, Wps
	66	vmovmskpd Gy, Upd	vsqrtpd Vpd, Wpd			vandpd Vpd, Hpd, Wpd	vandnpd Vpd, Hpd, Wpd	vorpd Vpd, Hpd, Wpd	vxorpd Vpd, Hpd, Wpd
	F3		vsqrtps Vss, Hss, Wss	vrsqrtps Vss, Hss, Wss	vrcpps Vss, Hss, Wss				
	F2		vsqrtps Vsd, Hsd, Wsd						
6		punpcklbw Pq, Qd	punpcklwd Pq, Qd	punpckldq Pq, Qd	packsswb Pq, Qq	pcmpgtb Pq, Qq	pcmpgtw Pq, Qq	pcmpgtd Pq, Qq	packuswb Pq, Qq
	66	vpunpcklbw Vx, Hx, Wx	vpunpcklwd Vx, Hx, Wx	vpunpckldq Vx, Hx, Wx	vpacksswb Vx, Hx, Wx	vpcmpgtb Vx, Hx, Wx	vpcmpgtw Vx, Hx, Wx	vpcmpgtd Vx, Hx, Wx	vpackuswb Vx, Hx, Wx
	F3								
7		pshufw Pq, Qq, Ib	(Grp 12 ^{1A})	(Grp 13 ^{1A})	(Grp 14 ^{1A})	pcmpeqb Pq, Qq	pcmpeqw Pq, Qq	pcmpeqd Pq, Qq	emms vzeroupper ^V vzeroall ^V
	66	vpshufd Vx, Wx, Ib				vpcmpeqb Vx, Hx, Wx	vpcmpeqw Vx, Hx, Wx	vpcmpeqd Vx, Hx, Wx	
	F3	vpshufw Vx, Wx, Ib							
	F2	vpshufw Vx, Wx, Ib							

...

Table A-3. Two-byte Opcode Map: 08H – 7FH (First Byte is 0FH) *

	pxf	8	9	A	B	C	D	E	F
0		INVD	WBINVD		2-byte Illegal Opcodes UD2 ^{1B}		prefetchw(/1) Ev		
1		Prefetch ^{1C} (Grp 16 ^{1A})							NOP /0 Ev
2		vmovaps Vps, Wps	vmovaps Wps, Vps	cvtpi2ps Vps, Qpi	vmovntps Mps, Vps	cvttps2pi Ppi, Wps	cvtps2pi Ppi, Wps	vucomiss Vss, Wss	vcomiss Vss, Wss
	66	vmovapd Vpd, Wpd	vmovapd Wpd, Vpd	cvtpi2pd Vpd, Qpi	vmovntpd Mpd, Vpd	cvttpd2pi Ppi, Wpd	cvtpd2pi Qpi, Wpd	vucomisd Vsd, Wsd	vcomisd Vsd, Wsd
	F3			vcvttsi2ss Vss, Hss, Ey		vcvtts2si Gy, Wss	vcvtss2si Gy, Wss		
	F2			vcvttsi2sd Vsd, Hsd, Ey		vcvttsd2si Gy, Wsd	vcvtssd2si Gy, Wsd		
3		3-byte escape (Table A-4)		3-byte escape (Table A-5)					
4		CMOVcc(Gv, Ev) - Conditional Move							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G

	px	8	9	A	B	C	D	E	F
5		vaddps Vps, Hps, Wps	vmulps Vps, Hps, Wps	vcvtps2pd Vpd, Wps	vcvtqd2ps Vps, Wdq	vsubps Vps, Hps, Wps	vminps Vps, Hps, Wps	vdivps Vps, Hps, Wps	vmaxps Vps, Hps, Wps
	66	vaddpd Vpd, Hpd, Wpd	vmulpd Vpd, Hpd, Wpd	vcvtpd2ps Vps, Wpd	vcvtps2dq Vdq, Wps	vsubpd Vpd, Hpd, Wpd	vminpd Vpd, Hpd, Wpd	vdivpd Vpd, Hpd, Wpd	vmaxpd Vpd, Hpd, Wpd
	F3	vaddss Vss, Hss, Wss	vmulss Vss, Hss, Wss	vcvtss2sd Vsd, Hx, Wss	vcvtps2dq Vdq, Wps	vsubss Vss, Hss, Wss	vminss Vss, Hss, Wss	vdivss Vss, Hss, Wss	vmaxss Vss, Hss, Wss
	F2	vaddsd Vsd, Hsd, Wsd	vmulsd Vsd, Hsd, Wsd	vcvtss2sd Vss, Hx, Wsd		vsubsd Vsd, Hsd, Wsd	vminsd Vsd, Hsd, Wsd	vdivsd Vsd, Hsd, Wsd	vmaxsd Vsd, Hsd, Wsd
6		punpckhbw Pq, Qd	punpckhwd Pq, Qd	punpckhdq Pq, Qd	packssdw Pq, Qd			movd/q Pd, Ey	movq Pq, Qq
	66	vpunpckhbw Vx, Hx, Wx	vpunpckhwd Vx, Hx, Wx	vpunpckhdq Vx, Hx, Wx	vpackssdw Vx, Hx, Wx	vpunpckldq Vx, Hx, Wx	vpunpckhdq Vx, Hx, Wx	vmovd/q Vy, Ey	vmovdqa Vx, Wx
	F3								vmovdqu Vx, Wx
7		VMREAD Ey, Gy	VMWRITE Gy, Ey					movd/q Ey, Pd	movq Qq, Pq
	66					vhaddpd Vpd, Hpd, Wpd	vsubpd Vpd, Hpd, Wpd	vmovd/q Ey, Vy	vmovdqa Wx, Vx
	F3							vmovq Vq, Wq	vmovdqu Wx, Vx
	F2					vhaddps Vps, Hps, Wps	vsubps Vps, Hps, Wps		

...

Table A-3. Two-byte Opcode Map: 80H – F7H (First Byte is 0FH) *

	px	0	1	2	3	4	5	6	7	
8		Jcc ⁶⁴ , Jz - Long-displacement jump on condition								
9		SETcc, Eb - Byte Set on condition								
A		PUSH ^{d64} FS	POP ^{d64} FS	CPUID	BT Ev, Gv	SHLD Ev, Gv, Ib	SHLD Ev, Gv, CL			
B		CMPXCHG Eb, Gb		Ev, Gv	LSS Gv, Mp	BTR Ev, Gv	LFS Gv, Mp	LGS Gv, Mp	MOVZX Gv, Eb	Gv, Ew
C		XADD Eb, Gb	XADD Ev, Gv	vcmpps Vps,Hps,Wps,Ib	movnti My, Gy	pinsrw Pq,Ry/Mw,Ib	pextrw Gd, Nq, Ib	vshufps Vps,Hps,Wps,Ib	Grp 9 ^{1A}	
	66			vcmpdd Vpd,Hpd,Wpd,Ib		vpinsrw Vdq,Hdq,Ry/Mw,Ib	vpextrw Gd, Udq, Ib	vshufpd Vpd,Hpd,Wpd,Ib		
	F3			vcmpps Vss,Hss,Wss,Ib						
	F2			vcmpsd Vsd,Hsd,Wsd,Ib						
D			psrlw Pq, Qq	psrld Pq, Qq	psrlq Pq, Qq	paddq Pq, Qq	pmullw Pq, Qq		pmovmskb Gd, Nq	
	66	vaddsubpd Vpd, Hpd, Wpd	vpsrlw Vx, Hx, Wx	vpsrld Vx, Hx, Wx	vpsrlq Vx, Hx, Wx	vpaddq Vx, Hx, Wx	vpmullw Vx, Hx, Wx	vmovq Wq, Vq	vpmovmskb Gd, Ux	
	F3							movq2dq Vdq, Nq		
	F2	vaddsubps Vps, Hps, Wps							movq2q Pq, Uq	

	pxf	0	1	2	3	4	5	6	7
E		pavgb Pq, Qq	psraw Pq, Qq	psrad Pq, Qq	pavgw Pq, Qq	pmulhw Pq, Qq	pmulhw Pq, Qq		movntq Mq, Pq
	66	vpavgb Vx, Hx, Wx	vpsraw Vx, Hx, Wx	vpsrad Vx, Hx, Wx	vpavgw Vx, Hx, Wx	vpmulhw Vx, Hx, Wx	vpmulhw Vx, Hx, Wx	vcvtpd2dq Vx, Wpd	vmovntdq Mx, Vx
	F3							vcvtdq2pd Vx, Wpd	
	F2							vcvtpd2dq Vx, Wpd	
F			psllw Pq, Qq	pslld Pq, Qq	psllq Pq, Qq	pmuludq Pq, Qq	pmaddwd Pq, Qq	psadbw Pq, Qq	maskmovq Pq, Nq
	66		vpsllw Vx, Hx, Wx	vpslld Vx, Hx, Wx	vpsllq Vx, Hx, Wx	vpmuludq Vx, Hx, Wx	vpmaddwd Vx, Hx, Wx	vpsadbw Vx, Hx, Wx	vmaskmovdqu Vdq, Udq
	F2	vlddqu Vx, Mx							

Table A-4 Three-byte Opcode Map: 00H – F7H (First Two Bytes are 0F 38H) *

	pxf	0	1	2	3	4	5	6	7
0		pshufb Pq, Qq	phaddw Pq, Qq	phadd Pq, Qq	phaddsw Pq, Qq	pmaddubsw Pq, Qq	phsubw Pq, Qq	phsubd Pq, Qq	phsubsw Pq, Qq
	66	vpshufb Vx, Hx, Wx	vphaddw Vx, Hx, Wx	vphadd Vx, Hx, Wx	vphaddsw Vx, Hx, Wx	vpmaddubsw Vx, Hx, Wx	vphsubw Vx, Hx, Wx	vphsubd Vx, Hx, Wx	vphsubsw Vx, Hx, Wx
1	66	pblendvb Vdq, Wdq			vcvtp2ps ^v Vx, Wx, Ib	blendvps Vdq, Wdq	blendvpd Vdq, Wdq	vpermps ^v Vq, Hq, Wq	vptest Vx, Wx
	66	vmovsxbw Vx, Ux/Mq	vmovsxbd Vx, Ux/Md	vmovsxbq Vx, Ux/Mw	vmovsxdw Vx, Ux/Mq	vmovsxwq Vx, Ux/Md	vmovsxdq Vx, Ux/Mq		
3	66	vmovzxbw Vx, Ux/Mq	vmovzxbd Vx, Ux/Md	vmovzxbq Vx, Ux/Mw	vmovzxdw Vx, Ux/Mq	vmovzxwq Vx, Ux/Md	vmovzxdq Vx, Ux/Mq	vpermd ^v Vq, Hq, Wq	vpcmpgtq Vx, Hx, Wx
4	66	vmulld Vx, Hx, Wx	vphminposuw Vdq, Wdq				vpsrlvd/q ^v Vx, Hx, Wx	vpsravd ^v Vx, Hx, Wx	vpsllvd/q ^v Vx, Hx, Wx
5									
6									
7									
8	66	INVEPT Gy, Mdq	INVVPID Gy, Mdq	INVPCID Gy, Mdq					
9	66	vgatherdd/q ^v Vx,Hx,Wx	vgatherqd/q ^v Vx,Hx,Wx	vgatherdps/d ^v Vx,Hx,Wx	vgatherqps/d ^v Vx,Hx,Wx			vfmaddsub132ps/d ^v Vx,Hx,Wx	vfmsubadd132ps/d ^v Vx,Hx,Wx
A	66							vfmaddsub213ps/d ^v Vx,Hx,Wx	vfmsubadd213ps/d ^v Vx,Hx,Wx
B	66							vfmaddsub231ps/d ^v Vx,Hx,Wx	vfmsubadd231ps/d ^v Vx,Hx,Wx
C									
D									
E									
F		MOVBE Gy, My	MOVBE My, Gy	ANDN ^v Gy, By, Ey	Grp 17 ^{1A}		BZHI ^v Gy, Ey, By		BEXTR ^v Gy, Ey, By
	66	MOVBE Gw, Mw	MOVBE Mw, Gw					ADCX Gy, Ey	SHLX ^v Gy, Ey, By
	F3						PEXT ^v Gy, By, Ey	ADOX Gy, Ey	SARX ^v Gy, Ey, By
	F2	CRC32 Gd, Eb	CRC32 Gd, Ey				PDEP ^v Gy, By, Ey	MULX ^v By,Gy,rDX,Ey	SHRX ^v Gy, Ey, By
	66 & F2	CRC32 Gd, Eb	CRC32 Gd, Ew						

Table A-4. Three-byte Opcode Map: 08H – FFH (First Two Bytes are 0F 38H) *

	px	8	9	A	B	C	D	E	F
0		psignb Pq, Qq	psignw Pq, Qq	psignd Pq, Qq	pmulhrsw Pq, Qq				
	66	vpsignb Vx, Hx, Wx	vpsignw Vx, Hx, Wx	vpsignd Vx, Hx, Wx	vpmulhrsw Vx, Hx, Wx	vpermilps ^V Vx,Hx,Wx	vpermilpd ^V Vx,Hx,Wx	vtestps ^V Vx, Wx	vtestpd ^V Vx, Wx
1						pabsb Pq, Qq	pabsw Pq, Qq	pabsd Pq, Qq	
	66	vbroadcastss ^V Vx, Wd	vbroadcastsd ^V Vqq, Wq	vbroadcastf128 ^V Vqq, Mdq		vpabsb Vx, Wx	vpabsw Vx, Wx	vpabsd Vx, Wx	
2	66	vpmuldq Vx, Hx, Wx	vpcmpeqq Vx, Hx, Wx	vmovntdqa Vx, Mx	vpackusdw Vx, Hx, Wx	vmaskmovps ^V Vx,Hx,Mx	vmaskmovpd ^V Vx,Hx,Mx	vmaskmovps ^V Mx,Hx,Vx	vmaskmovpd ^V Mx,Hx,Vx
3	66	vpminsb Vx, Hx, Wx	vpminsd Vx, Hx, Wx	vpmiunw Vx, Hx, Wx	vpmiunw Vx, Hx, Wx	vpmaxsb Vx, Hx, Wx	vpmaxsd Vx, Hx, Wx	vpmaxuw Vx, Hx, Wx	vpmaxud Vx, Hx, Wx
4									
5	66	vpbroadcastd ^V Vx, Wx	vpbroadcastq ^V Vx, Wx	vpbroadcasti128 ^V Vqq, Mdq					
6									
7	66	vpbroadcastb ^V Vx, Wx	vpbroadcastw ^V Vx, Wx						
8	66					vpmaskmovd/q ^V Vx,Hx,Mx		vpmaskmovd/q ^V Mx,Vx,Hx	
9	66	vfmadd132ps/d ^V Vx, Hx, Wx	vfmadd132ss/d ^V Vx, Hx, Wx	vfmsub132ps/d ^V Vx, Hx, Wx	vfmsub132ss/d ^V Vx, Hx, Wx	vfnmadd132ps/d ^V Vx, Hx, Wx	vfnmadd132ss/d ^V Vx, Hx, Wx	vfnmsub132ps/d ^V Vx, Hx, Wx	vfnmsub132ss/d ^V Vx, Hx, Wx
A	66	vfmadd213ps/d ^V Vx, Hx, Wx	vfmadd213ss/d ^V Vx, Hx, Wx	vfmsub213ps/d ^V Vx, Hx, Wx	vfmsub213ss/d ^V Vx, Hx, Wx	vfnmadd213ps/d ^V Vx, Hx, Wx	vfnmadd213ss/d ^V Vx, Hx, Wx	vfnmsub213ps/d ^V Vx, Hx, Wx	vfnmsub213ss/d ^V Vx, Hx, Wx
B	66	vfmadd231ps/d ^V Vx, Hx, Wx	vfmadd231ss/d ^V Vx, Hx, Wx	vfmsub231ps/d ^V Vx, Hx, Wx	vfmsub231ss/d ^V Vx, Hx, Wx	vfnmadd231ps/d ^V Vx, Hx, Wx	vfnmadd231ss/d ^V Vx, Hx, Wx	vfnmsub231ps/d ^V Vx, Hx, Wx	vfnmsub231ss/d ^V Vx, Hx, Wx
C									
D	66				VAESIMC Vdq, Wdq	VAESENQ Vdq,Hdq,Wdq	VAESENCLAST Vdq,Hdq,Wdq	VAESDEC Vdq,Hdq,Wdq	VAESDECLAST Vdq,Hdq,Wdq
E									
F									
	66								
	F3								
	F2								
	66 & F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

...

Table A-5 Three-byte Opcode Map: 00H – F7H (First two bytes are 0F 3AH) *

	px	0	1	2	3	4	5	6	7
0	66	vpermq ^y Vqq, Wqq, Ib	vpermpd ^y Vqq, Wqq, Ib	vpblendd ^y Vx,Hx,Wx,Ib		vpermilps ^y Vx, Wx, Ib	vpermilpd ^y Vx, Wx, Ib	vperm2f128 ^y Vqq,Hqq,Wqq,Ib	
1	66					vpextrb Rd/Mb, Vdq, Ib	vpextrw Rd/Mw, Vdq, Ib	vpextrd/q Ey, Vdq, Ib	vextractps Ed, Vdq, Ib
2	66	vpinsrb Vdq,Hdq,Ry/Mb,Ib	vinsertps Vdq,Hdq,Udq/Md,Ib	vpinsrd/q Vdq,Hdq,Ey,Ib					
3									
4	66	vdpps Vx,Hx,Wx,Ib	vdppd Vdq,Hdq,Wdq,Ib	vmpsadbw Vx,Hx,Wx,Ib		vpclmulqdq Vdq,Hdq,Wdq,Ib		vperm2i128 ^y Vqq,Hqq,Wqq,Ib	
5									
6	66	vpcmpstrm Vdq, Wdq, Ib	vpcmpstri Vdq, Wdq, Ib	vpcmpstrm Vdq, Wdq, Ib	vpcmpstri Vdq, Wdq, Ib				
7									
8									
9									
A									
B									
C									
D									
E									
F	F2	RORX ^y Gy, Ey, Ib							

...

Table A-5. Three-byte Opcode Map: 08H – FFH (First Two Bytes are 0F 3AH) *

	pxf	8	9	A	B	C	D	E	F
0									palignr Pq, Qq, lb
	66	vroundps Vx, Wx, lb	vroundpd Vx, Wx, lb	vroundss Vss, Wss, lb	vroundsd Vsd, Wsd, lb	vblendps Vx, Hx, Wx, lb	vblendpd Vx, Hx, Wx, lb	vpblendw Vx, Hx, Wx, lb	vpalignr Vx, Hx, Wx, lb
1	66	vinsertf128 ^v Vqq, Hqq, Wqq, lb	vextractf128 ^v Wdq, Vqq, lb				vcvtps2ph ^v Wx, Vx, lb		
2									
3	66	vinserti128 ^v Vqq, Hqq, Wqq, lb	vextracti128 ^v Wdq, Vqq, lb						
4	66			vblendvps ^v Vx, Hx, Wx, Lx	vblendvpd ^v Vx, Hx, Wx, Lx	vpblendvb ^v Vx, Hx, Wx, Lx			
5									
6									
7									
8									
9									
A									
B									
C									
D	66								VAESKEYGEN Vdq, Wdq, lb
E									
F									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

...

Table A-6 Opcode Extensions for One- and Two-byte Opcodes by Group Number *

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
80-83	1	mem, 11B		ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
8F	1A	mem, 11B		POP							
C0,C1 reg, imm D0, D1 reg, 1 D2, D3 reg, CL	2	mem, 11B		ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
F6, F7	3	mem, 11B		TEST lb/lz		NOT	NEG	MUL AL/rAX	IMUL AL/rAX	DIV AL/rAX	IDIV AL/rAX
FE	4	mem, 11B		INC Eb	DEC Eb						
FF	5	mem, 11B		INC Ev	DEC Ev	CALLN ⁶⁴ Ev	CALLF Ep	JMPN ⁶⁴ Ev	JMPF Mp	PUSH ^{d64} Ev	
0F 00	6	mem, 11B		SLDT Rv/Mw	STR Rv/Mw	LLDT Ew	LTR Ew	VERR Ew	VERW Ew		
0F 01	7	mem		SGDT Ms	SIDT Ms	LGDT Ms	LIDT Ms	SMSW Mw/Rv		LMSW Ew	INVLPG Mb
		11B		VMCALL (001) VMLAUNCH (010) VMRESUME (011) VMXOFF (100)	MONITOR (000) MWAIT (001) CLAC (010) STAC (011)	XGETBV (000) XSETBV (001) VMFUNC (100) XEND (101) XTEST (110)				SWAPGS ^{o64} (000) RDTSCP (001)	
0F BA	8	mem, 11B						BT	BTS	BTR	BTC
0F C7	9	mem			CMPXCH8B Mq CMPXCHG16B Mdq					VMPTRLD Mq	VMPTRST Mq
			66							VMCLEAR Mq	
		F3								VMXON Mq	VMPTRST Mq
0F B9	10	mem								RDRAND Rv	RDSEED Rv
		11B									
C6	11	mem		MOV Eb, lb							
11B										XABORT (000) lb	
C7	11	mem		MOV Ev, lz							
11B										XBEGIN (000) Jz	
0F 71	12	mem				psrlw Nq, lb		psraw Nq, lb		psllw Nq, lb	
		11B				vpsrlw Hx,Ux,lb		vpsraw Hx,Ux,lb		vpsllw Hx,Ux,lb	
0F 72	13	mem				psrld Nq, lb		psrad Nq, lb		pslld Nq, lb	
		11B				vpsrld Hx,Ux,lb		vpsrad Hx,Ux,lb		vpslld Hx,Ux,lb	
0F 73	14	mem				psrlq Nq, lb				psllq Nq, lb	
		11B				vpsrlq Hx,Ux,lb	vpsrldq Hx,Ux,lb			vpsllq Hx,Ux,lb	vpslldq Hx,Ux,lb

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
OF AE	15	mem		fxsave	fxrstor	ldmxcsr	stmxcsr	XSAVE	XRSTOR	XSAVEOPT	clflush
		11B	F3	RDFSBASE Ry	RDGSBASE Ry	WRFSBASE Ry	WRGSBASE Ry		lfence	mfence	sfence
OF 18	16	mem		prefetch NTA	prefetch T0	prefetch T1	prefetch T2				
		11B									
VEX.0F38 F3	17	mem			BLSR ^v By, Ey	BLSMSK ^v By, Ey	BLSI ^v By, Ey				
		11B									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

10. Updates to Chapter 1, Volume 3A

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

...

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

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

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

- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Core™2 Extreme QX9000 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel® Atom™ processor family
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Xeon® processor E5 family
- Intel® Xeon® processor E3-1200 family
- 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 v2 product family
- 3rd generation Intel® Core™ processors
- Next generation Intel® Core™ processors

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

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

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

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

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

The Intel® Atom™ processor family is based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and the Intel® Core™ i5 processor are based on the Intel® microarchitecture code name Nehalem and support Intel 64 architecture.

Processors based on Intel® microarchitecture code name Westmere 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, 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Intel® microarchitecture code name Sandy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Intel® microarchitecture code name Ivy Bridge and support Intel 64 architecture.

The Next Generation Intel® Core™ processors are based on the Intel® microarchitecture code name Haswell and support 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 processors, 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.

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 a superset of and compatible with IA-32 architecture.

1.2 OVERVIEW OF THE SYSTEM PROGRAMMING GUIDE

A description of this manual's content follows:

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

Chapter 2 — System Architecture Overview. Describes the modes of operation used by Intel 64 and IA-32 processors and the mechanisms provided by the architectures to support operating systems and executives, including the system-oriented registers and data structures and the system-oriented instructions. The steps necessary for switching between real-address and protected modes are also identified.

Chapter 3 — Protected-Mode Memory Management. Describes the data structures, registers, and instructions that support segmentation and paging. The chapter explains how they can be used to implement a "flat" (unsegmented) memory model or a segmented memory model.

Chapter 4 — Paging. Describes the paging modes supported by Intel 64 and IA-32 processors.

Chapter 5 — Protection. Describes the support for page and segment protection provided in the Intel 64 and IA-32 architectures. This chapter also explains the implementation of privilege rules, stack switching, pointer validation, user and supervisor modes.

Chapter 6 — Interrupt and Exception Handling. Describes the basic interrupt mechanisms defined in the Intel 64 and IA-32 architectures, shows how interrupts and exceptions relate to protection, and describes how the architecture handles each exception type. Reference information for each exception is given in this chapter. Includes programming the LINT0 and LINT1 inputs and gives an example of how to program the LINT0 and LINT1 pins for specific interrupt vectors.

Chapter 7 — Task Management. Describes mechanisms the Intel 64 and IA-32 architectures provide to support multitasking and inter-task protection.

Chapter 8 — Multiple-Processor Management. Describes the instructions and flags that support multiple processors with shared memory, memory ordering, and Intel® Hyper-Threading Technology. Includes MP initialization for P6 family processors and gives an example of how to use of the MP protocol to boot P6 family processors in an MP system.

Chapter 9 — Processor Management and Initialization. Defines the state of an Intel 64 or IA-32 processor after reset initialization. This chapter also explains how to set up an Intel 64 or IA-32 processor for real-address mode operation and protected- mode operation, and how to switch between modes.

Chapter 10 — Advanced Programmable Interrupt Controller (APIC). Describes the programming interface to the local APIC and gives an overview of the interface between the local APIC and the I/O APIC. Includes APIC

bus message formats and describes the message formats for messages transmitted on the APIC bus for P6 family and Pentium processors.

Chapter 11 — Memory Cache Control. Describes the general concept of caching and the caching mechanisms supported by the Intel 64 or IA-32 architectures. This chapter also describes the memory type range registers (MTRRs) and how they can be used to map memory types of physical memory. Information on using the new cache control and memory streaming instructions introduced with the Pentium III, Pentium 4, and Intel Xeon processors is also given.

Chapter 12 — Intel® MMX™ Technology System Programming. Describes those aspects of the Intel® MMX™ technology that must be handled and considered at the system programming level, including: task switching, exception handling, and compatibility with existing system environments.

Chapter 13 — System Programming For Instruction Set Extensions And Processor Extended States. Describes the operating system requirements to support SSE/SSE2/SSE3/SSSE3/SSE4 extensions, including task switching, exception handling, and compatibility with existing system environments. The latter part of this chapter describes the extensible framework of operating system requirements to support processor extended states. Processor extended state may be required by instruction set extensions beyond those of SSE/SSE2/SSE3/SSSE3/SSE4 extensions.

Chapter 14 — Power and Thermal Management. Describes facilities of Intel 64 and IA-32 architecture used for power management and thermal monitoring.

Chapter 15 — Machine-Check Architecture. Describes the machine-check architecture and machine-check exception mechanism found in the Pentium 4, Intel Xeon, and P6 family processors. Additionally, a signaling mechanism for software to respond to hardware corrected machine check error is covered.

Chapter 16 — Interpreting Machine-Check Error Codes. Gives an example of how to interpret the error codes for a machine-check error that occurred on a P6 family processor.

Chapter 17 — Debugging, Branch Profiles and Time-Stamp Counter. Describes the debugging registers and other debug mechanism provided in Intel 64 or IA-32 processors. This chapter also describes the time-stamp counter.

Chapter 18 — Performance Monitoring. Describes the Intel 64 and IA-32 architectures' facilities for monitoring performance.

Chapter 19 — Performance-Monitoring Events. Lists architectural performance events. Non-architectural performance events (i.e. model-specific events) are listed for each generation of microarchitecture.

Chapter 20 — 8086 Emulation. Describes the real-address and virtual-8086 modes of the IA-32 architecture.

Chapter 21 — Mixing 16-Bit and 32-Bit Code. Describes how to mix 16-bit and 32-bit code modules within the same program or task.

Chapter 22 — IA-32 Architecture Compatibility. Describes architectural compatibility among IA-32 processors.

Chapter 23 — Introduction to Virtual-Machine Extensions. Describes the basic elements of virtual machine architecture and the virtual-machine extensions for Intel 64 and IA-32 Architectures.

Chapter 24 — Virtual-Machine Control Structures. Describes components that manage VMX operation. These include the working-VMCS pointer and the controlling-VMCS pointer.

Chapter 25 — VMX Non-Root Operation. Describes the operation of a VMX non-root operation. Processor operation in VMX non-root mode can be restricted programmatically such that certain operations, events or conditions can cause the processor to transfer control from the guest (running in VMX non-root mode) to the monitor software (running in VMX root mode).

Chapter 26 — VM Entries. Describes VM entries. VM entry transitions the processor from the VMM running in VMX root-mode to a VM running in VMX non-root mode. VM-Entry is performed by the execution of VMLAUNCH or VMRESUME instructions.

Chapter 27 — VM Exits. Describes VM exits. Certain events, operations or situations while the processor is in VMX non-root operation may cause VM-exit transitions. In addition, VM exits can also occur on failed VM entries.

Chapter 28 — VMX Support for Address Translation. Describes virtual-machine extensions that support address translation and the virtualization of physical memory.

Chapter 29 — APIC Virtualization and Virtual Interrupts. Describes the VMCS including controls that enable the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC).

Chapter 30 — VMX Instruction Reference. Describes the virtual-machine extensions (VMX). VMX is intended for a system executive to support virtualization of processor hardware and a system software layer acting as a host to multiple guest software environments.

Chapter 31 — Virtual-Machine Monitoring Programming Considerations. Describes programming considerations for VMMs. VMMs manage virtual machines (VMs).

Chapter 32 — Virtualization of System Resources. Describes the virtualization of the system resources. These include: debugging facilities, address translation, physical memory, and microcode update facilities.

Chapter 33 — Handling Boundary Conditions in a Virtual Machine Monitor. Describes what a VMM must consider when handling exceptions, interrupts, error conditions, and transitions between activity states.

Chapter 34 — System Management Mode. Describes Intel 64 and IA-32 architectures' system management mode (SMM) facilities.

Chapter 35 — Model-Specific Registers (MSRs). Lists the MSRs available in the Pentium processors, the P6 family processors, the Pentium 4, Intel Xeon, Intel Core Solo, Intel Core Duo processors, and Intel Core 2 processor family and describes their functions.

Appendix A — VMX Capability Reporting Facility. Describes the VMX capability MSRs. Support for specific VMX features is determined by reading capability MSRs.

Appendix B — Field Encoding in VMCS. Enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.).

Appendix C — VM Basic Exit Reasons. Describes the 32-bit fields that encode reasons for a VM exit. Examples of exit reasons include, but are not limited to: software interrupts, processor exceptions, software traps, NMIs, external interrupts, and triple faults.

...

11. Updates to Chapter 2, Volume 3A

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

...

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 4.6, "Access Rights"). The layout of the IA32_EFER MSR is shown in Figure 2-4.

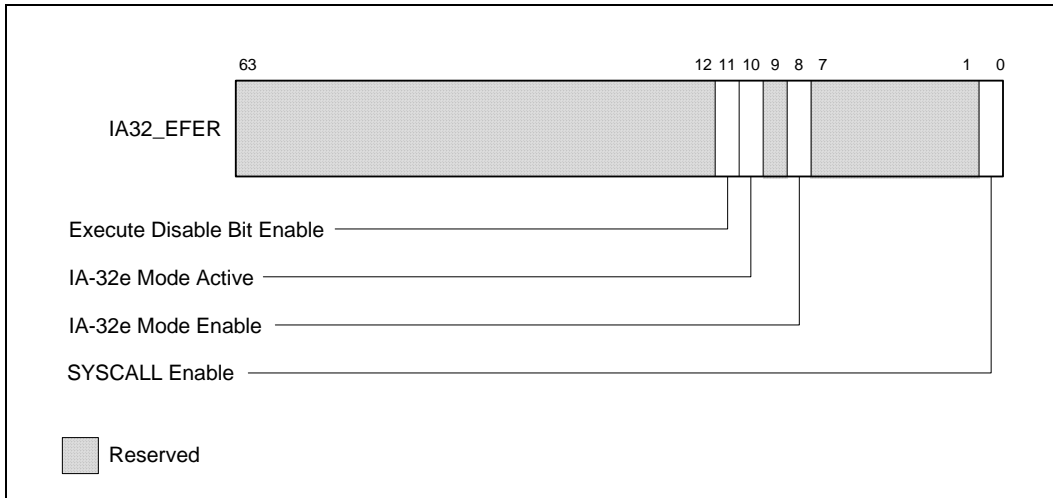


Figure 2-4 IA32_EFER MSR Layout

Table 2-1 IA32_EFER MSR Information

Bit	Description
0	SYSCALL Enable (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.
7:1	Reserved.
8	IA-32e Mode Enable (R/W) Enables IA-32e mode operation.
9	Reserved.
10	IA-32e Mode Active (R) Indicates IA-32e mode is active when set.
11	Execute Disable Bit Enable (R/W) Enables page access restriction by preventing instruction fetches from PAE pages with the XD bit set (See Section 4.6).
63:12	Reserved.

...

12. Updates to Chapter 4, Volume 3A

Change bars show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

...

4.1.1 Three Paging Modes

If CR0.PG = 0, paging is not used. The logical processor treats all linear addresses as if they were physical addresses. CR4.PAE and IA32_EFER.LME are ignored by the processor, as are CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, and IA32_EFER.NXE.

Paging is enabled if CR0.PG = 1. Paging can be enabled only if protection is enabled (CR0.PE = 1). If paging is enabled, one of three paging modes is used. The values of CR4.PAE and IA32_EFER.LME determine which paging mode is used:

- If CR0.PG = 1 and CR4.PAE = 0, **32-bit paging** is used. 32-bit paging is detailed in Section 4.3. 32-bit paging uses CR0.WP, CR4.PSE, CR4.PGE, and CR4.SMEP as described in Section 4.1.3.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, **PAE paging** is used. PAE paging is detailed in Section 20. PAE paging uses CR0.WP, CR4.PGE, CR4.SMEP, and IA32_EFER.NXE as described in Section 4.1.3.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 1, **IA-32e paging** is used.¹ IA-32e paging is detailed in Section 4.5. IA-32e paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, and IA32_EFER.NXE as described in Section 4.1.3. IA-32e paging is available only on processors that support the Intel 64 architecture.

The three paging modes differ with regard to the following details:

- Linear-address width. The size of the linear addresses that can be translated.
- Physical-address width. The size of the physical addresses produced by paging.
- Page size. The granularity at which linear addresses are translated. Linear addresses on the same page are translated to corresponding physical addresses on the same page.
- Support for execute-disable access rights. In some paging modes, software can be prevented from fetching instructions from pages that are otherwise readable.
- Support for PCIDs. In some paging modes, software can enable a facility by which a logical processor caches information for multiple linear-address spaces. The processor may retain cached information when software switches between different linear-address spaces.

Table 4-1 illustrates the key differences between the three paging modes.

Table 4-1 Properties of Different Paging Modes

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	Lin.-Addr. Width	Phys.-Addr. Width ¹	Page Sizes	Supports Execute-Disable?	Supports PCIDs?
None	0	N/A	N/A	32	32	N/A	No	No
32-bit	1	0	0 ²	32	Up to 40 ³	4 KB 4 MB ⁴	No	No
PAE	1	1	0	32	Up to 52	4 KB 2 MB	Yes ⁵	No
IA-32e	1	1	1	48	Up to 52	4 KB 2 MB 1 GB ⁶	Yes ⁵	Yes ⁷

NOTES:

1. The physical-address width is always bounded by MAXPHYADDR; see Section 4.1.4.
2. The processor ensures that IA32_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.

1. The LMA flag in the IA32_EFER MSR (bit 10) is a status bit that indicates whether the logical processor is in IA-32e mode (and thus using IA-32e paging). The processor always sets IA32_EFER.LMA to CR0.PG & IA32_EFER.LME. Software cannot directly modify IA32_EFER.LMA; an execution of WRMSR to the IA32_EFER MSR ignores bit 10 of its source operand.

3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 4.1.4 and Section 4.3.
4. 4-MByte pages are used with 32-bit paging only if CR4.PSE = 1; see Section 4.3.
5. Execute-disable access rights are applied only if IA32_EFER.NXE = 1; see Section 4.6.
6. Not all processors that support IA-32e paging support 1-GByte pages; see Section 4.1.4.
7. PCIDs are used only if CR4.PCIDE = 1; see Section 4.10.1.

Because they are used only if IA32_EFER.LME = 0, 32-bit paging and PAE paging is used only in legacy protected mode. Because legacy protected mode cannot produce linear addresses larger than 32 bits, 32-bit paging and PAE paging translate 32-bit linear addresses.

Because it is used only if IA32_EFER.LME = 1, IA-32e paging is used only in IA-32e mode. (In fact, it is the use of IA-32e paging that defines IA-32e mode.) IA-32e mode has two sub-modes:

- Compatibility mode. This mode uses only 32-bit linear addresses. IA-32e paging treats bits 47:32 of such an address as all 0.
- 64-bit mode. While this mode produces 64-bit linear addresses, the processor ensures that bits 63:47 of such an address are identical.¹ IA-32e paging does not use bits 63:48 of such addresses.

...

13. Updates to Chapter 5, Volume 3A

Change bars show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

...

5.8 PRIVILEGE LEVEL CHECKING WHEN TRANSFERRING PROGRAM CONTROL BETWEEN CODE SEGMENTS

To transfer program control from one code segment to another, the segment selector for the destination code segment must be loaded into the code-segment register (CS). As part of this loading process, the processor examines the segment descriptor for the destination code segment and performs various limit, type, and privilege checks. If these checks are successful, the CS register is loaded, program control is transferred to the new code segment, and program execution begins at the instruction pointed to by the EIP register.

Program control transfers are carried out with the JMP, CALL, RET, SYSENTER, SYSEXIT, SYSCALL, SYSRET, INT *n*, and IRET instructions, as well as by the exception and interrupt mechanisms. Exceptions, interrupts, and the IRET instruction are special cases discussed in Chapter 6, "Interrupt and Exception Handling." This chapter discusses only the JMP, CALL, RET, SYSENTER, SYSEXIT, SYSCALL, and SYSRET instructions.

A JMP or CALL instruction can reference another code segment in any of four ways:

- The target operand contains the segment selector for the target code segment.
- The target operand points to a call-gate descriptor, which contains the segment selector for the target code segment.
- The target operand points to a TSS, which contains the segment selector for the target code segment.

1. Such an address is called **canonical**. Use of a non-canonical linear address in 64-bit mode produces a general-protection exception (#GP(0)); the processor does not attempt to translate non-canonical linear addresses using IA-32e paging.

- The target operand points to a task gate, which points to a TSS, which in turn contains the segment selector for the target code segment.

The following sections describe first two types of references. See Section 7.3, “Task Switching,” for information on transferring program control through a task gate and/or TSS.

The SYSENTER and SYSEXIT instructions are special instructions for making fast calls to and returns from operating system or executive procedures. These instructions are discussed in Section 5.8.7, “Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions.”

The SYSCALL and SYSRET instructions are special instructions for making fast calls to and returns from operating system or executive procedures in 64-bit mode. These instructions are discussed in Section 5.8.8, “Fast System Calls in 64-Bit Mode.”

...

5.8.7.1 SYSENTER and SYSEXIT Instructions in IA-32e Mode

For Intel 64 processors, the SYSENTER and SYSEXIT instructions are enhanced to allow fast system calls from user code running at privilege level 3 (in compatibility mode or 64-bit mode) to 64-bit executive procedures running at privilege level 0. IA32_SYSENTER_EIP MSR and IA32_SYSENTER_ESP MSR are expanded to hold 64-bit addresses. If IA-32e mode is inactive, only the lower 32-bit addresses stored in these MSRs are used. The WRMSR instruction ensures that the addresses stored in these MSRs are canonical. Note that, in 64-bit mode, IA32_SYSENTER_CS must not contain a NULL selector.

When SYSENTER transfers control, the following fields are generated and bits set:

- **Target code segment** — Reads non-NULL selector from IA32_SYSENTER_CS.
- **New CS attributes** — CS base = 0, CS limit = FFFFFFFFH.
- **Target instruction** — Reads 64-bit canonical address from IA32_SYSENTER_EIP.
- **Stack segment** — Computed by adding 8 to the value from IA32_SYSENTER_CS.
- **Stack pointer** — Reads 64-bit canonical address from IA32_SYSENTER_ESP.
- **New SS attributes** — SS base = 0, SS limit = FFFFFFFFH.

When the SYSEXIT instruction transfers control to 64-bit mode user code using REX.W, the following fields are generated and bits set:

- **Target code segment** — Computed by adding 32 to the value in IA32_SYSENTER_CS.
- **New CS attributes** — L-bit = 1 (go to 64-bit mode).
- **Target instruction** — Reads 64-bit canonical address in RDX.
- **Stack segment** — Computed by adding 40 to the value of IA32_SYSENTER_CS.
- **Stack pointer** — Update RSP using 64-bit canonical address in RCX.

When SYSEXIT transfers control to compatibility mode user code when the operand size attribute is 32 bits, the following fields are generated and bits set:

- **Target code segment** — Computed by adding 16 to the value in IA32_SYSENTER_CS.
- **New CS attributes** — L-bit = 0 (go to compatibility mode).
- **Target instruction** — Fetch the target instruction from 32-bit address in EDX.
- **Stack segment** — Computed by adding 24 to the value in IA32_SYSENTER_CS.
- **Stack pointer** — Update ESP from 32-bit address in ECX.

5.8.8 Fast System Calls in 64-Bit Mode

The SYSCALL and SYSRET instructions are designed for operating systems that use a flat memory model (segmentation is not used). The instructions, along with SYSENTER and SYSEXIT, are suited for IA-32e mode operation. SYSCALL and SYSRET, however, are not supported in compatibility mode (or in protected mode). Use CPUID to check if SYSCALL and SYSRET are available (CPUID.80000001H.EDX[bit 11] = 1).

SYSCALL is intended for use by user code running at privilege level 3 to access operating system or executive procedures running at privilege level 0. SYSRET is intended for use by privilege level 0 operating system or executive procedures for fast returns to privilege level 3 user code.

Stack pointers for SYSCALL/SYSRET are not specified through model specific registers. The clearing of bits in RFLAGS is programmable rather than fixed. SYSCALL/SYSRET save and restore the RFLAGS register.

For SYSCALL, the processor saves RFLAGS into R11 and the RIP of the next instruction into RCX; it then gets the privilege-level 0 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[47:32].
- **Target instruction pointer** — Reads a 64-bit address from IA32_LSTAR. (The WRMSR instruction ensures that the value of the IA32_LSTAR MSR is canonical.)
- **Stack segment** — Computed by adding 8 to the value in IA32_STAR[47:32].
- **Flags** — The processor sets RFLAGS to the logical-AND of its current value with the complement of the value in the IA32_FMASK MSR.

When SYSRET transfers control to 64-bit mode user code using REX.W, the processor gets the privilege level 3 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[63:48] + 16.
- **Target instruction pointer** — Copies the value in RCX into RIP.
- **Stack segment** — IA32_STAR[63:48] + 8.
- **EFLAGS** — Loaded from R11.

When SYSRET transfers control to 32-bit mode user code using a 32-bit operand size, the processor gets the privilege level 3 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[63:48].
- **Target instruction pointer** — Copies the value in ECX into EIP.
- **Stack segment** — IA32_STAR[63:48] + 8.
- **EFLAGS** — Loaded from R11.

It is the responsibility of the OS to ensure the descriptors in the GDT/LDT correspond to the selectors loaded by SYSCALL/SYSRET (consistent with the base, limit, and attribute values forced by the instructions).

See Figure 5-14 for the layout of IA32_STAR, IA32_LSTAR and IA32_FMASK.

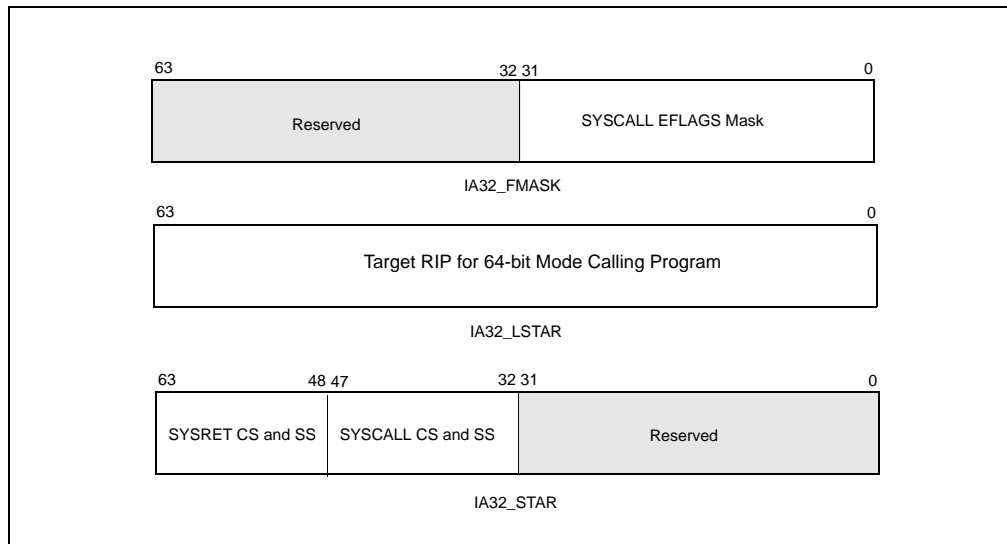


Figure 5-14 MSRs Used by SYSCALL and SYSRET

The SYSCALL instruction does not save the stack pointer, and the SYSRET instruction does not restore it. It is likely that the OS system-call handler will change the stack pointer from the user stack to the OS stack. If so, it is the responsibility of software first to save the user stack pointer. This might be done by user code, prior to executing SYSCALL, or by the OS system-call handler after SYSCALL.

Because the SYSRET instruction does not modify the stack pointer, it is necessary for software to switch back to the user stack. The OS may load the user stack pointer (if it was saved after SYSCALL) before executing SYSRET; alternatively, user code may load the stack pointer (if it was saved before SYSCALL) after receiving control from SYSRET.

If the OS loads the stack pointer before executing SYSRET, it must ensure that the handler of any interrupt or exception delivered between restoring the stack pointer and successful execution of SYSRET is not invoked with the user stack. It can do so using approaches such as the following:

- External interrupts. The OS can prevent an external interrupt from being delivered by clearing EFLAGS.IF before loading the user stack pointer.
- Nonmaskable interrupts (NMIs). The OS can ensure that the NMI handler is invoked with the correct stack by using the interrupt stack table (IST) mechanism for gate 2 (NMI) in the IDT (see Section 6.14.5, “Interrupt Stack Table”).
- General-protection exceptions (#GP). The SYSRET instruction generates #GP(0) if the value of RCX is not canonical. The OS can address this possibility using one or more of the following approaches:
 - Confirming that the value of RCX is canonical before executing SYSRET.
 - Using paging to ensure that the SYSCALL instruction will never save a non-canonical value into RCX.
 - Using the IST mechanism for gate 13 (#GP) in the IDT.

...

14. Updates to Chapter 11, Volume 3A

Change bars show changes to Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

...

11.11.2.4 System-Management Range Register Interface

If IA32_MTRRCAP[bit 11] is set, the processor supports the SMRR interface to restrict access to a specified memory address range used by system-management mode (SMM) software (see Section 34.4.2.1). If the SMRR interface is supported, SMM software is strongly encouraged to use it to protect the SMI code and data stored by SMI handler in the SMRAM region.

The system-management range registers consist of a pair of MSRs (see Figure 11-8). The IA32_SMRR_PHYSBASE MSR defines the base address for the SMRAM memory range and the memory type used to access it in SMM. The IA32_SMRR_PHYSMASK MSR contains a valid bit and a mask that determines the SMRAM address range protected by the SMRR interface. These MSRs may be written only in SMM; an attempt to write them outside of SMM causes a general-protection exception.¹

Figure 11-8 shows flags and fields in these registers. The functions of these flags and fields are the following:

- **Type field, bits 0 through 7** — Specifies the memory type for the range (see Table 11-8 for the encoding of this field).
- **PhysBase field, bits 12 through 31** — Specifies the base address of the address range. The address must be less than 4 GBytes and is automatically aligned on a 4-KByte boundary.
- **PhysMask field, bits 12 through 31** — Specifies a mask that determines the range of the region being mapped, according to the following relationships:
 - $\text{Address_Within_Range AND PhysMask} = \text{PhysBase AND PhysMask}$
 - This value is extended by 12 bits at the low end to form the mask value. For more information: see Section 11.11.3, "Example Base and Mask Calculations."
- **V (valid) flag, bit 11** — Enables the register pair when set; disables register pair when clear.

Before attempting to access these SMRR registers, software must test bit 11 in the IA32_MTRRCAP register. If SMRR is not supported, reads from or writes to registers cause general-protection exceptions.

When the valid flag in the IA32_SMRR_PHYSMASK MSR is 1, accesses to the specified address range are treated as follows:

- If the logical processor is in SMM, accesses uses the memory type in the IA32_SMRR_PHYSBASE MSR.
- If the logical processor is not in SMM, write accesses are ignored and read accesses return a fixed value for each byte. The uncacheable memory type (UC) is used in this case.

The above items apply even if the address range specified overlaps with a range specified by the MTRRs.

1. For some processor models, these MSRs can be accessed by RDMSR and WRMSR only if the SMRR interface has been enabled using a model-specific bit in the IA32_FEATURE_CONTROL MSR.

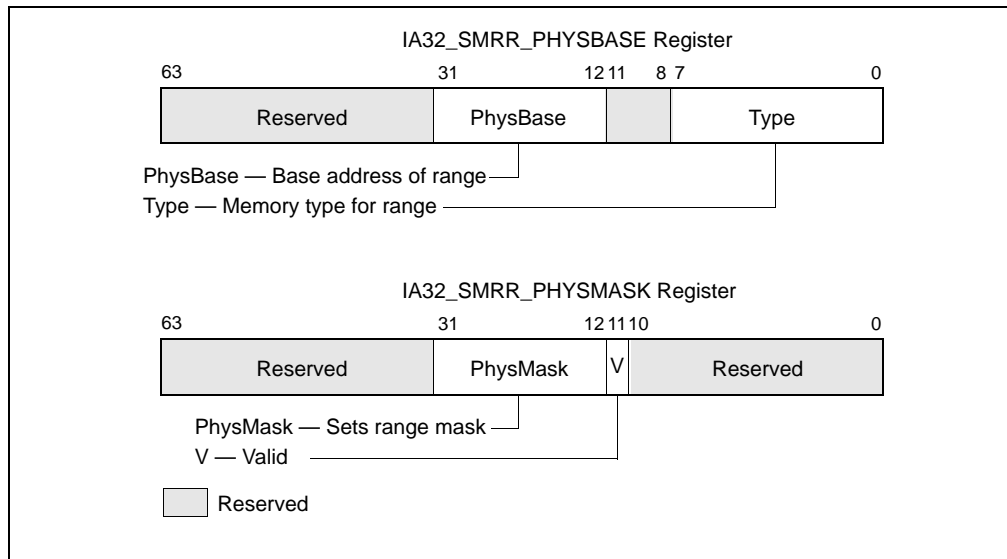


Figure 11-8 IA32_SMRR_PHYSBASE and IA32_SMRR_PHYSMASK SMRR Pair

...

15. Updates to Chapter 16, Volume 3B

Change bars show changes to Chapter 16 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

...

16.4.1 Internal Machine Check Errors

Table 16-13 Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes ¹	0-15	MCACOD	
Model specific errors	19:16	Reserved except for the following	0000b - No Error 0001b - Non_IMem_Sel 0010b - I_Parity_Error 0011b - Bad_OpCode 0100b - I_Stack_Underflow 0101b - I_Stack_Overflow 0110b - D_Stack_Underflow 0111b - D_Stack_Overflow 1000b - Non-DMem_Sel 1001b - D_Parity_Error
	23-20	Reserved	Reserved
	31-24	Reserved except for the following	00h - No Error 0Dh - MC_IMC_FORCE_SR_S3_TIMEOUT 0Eh - MC_CPD_UNCPD_ST_TIMEOUT 0Fh - MC_PKGS_SAFE_WP_TIMEOUT 43h - MC_PECI_MAILBOX QUIESCE_TIMEOUT 5Ch - MC_MORE_THAN_ONE_LT_AGENT 60h - MC_INVALID_PKGS_REQ_PCH 61h - MC_INVALID_PKGS_REQ_QPI 62h - MC_INVALID_PKGS_RES_QPI 63h - MC_INVALID_PKGC_RES_PCH 64h - MC_INVALID_PKG_STATE_CONFIG 70h - MC_WATCHDG_TIMEOUT_PKGC_SLAVE 71h - MC_WATCHDG_TIMEOUT_PKGC_MASTER 72h - MC_WATCHDG_TIMEOUT_PKGS_MASTER 7ah - MC_HA_FAILSTS_CHANGE_DETECTED 81h - MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56-32	Reserved	Reserved
Status register validity indicators ¹	57-63		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

...

16.4.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC8_STATUS-IA32_MC11_STATUS. The supported error codes are follows the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, “Machine-Check Architecture,”). MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in IA32_MCi_STATUS and IA32_MCi_MISC; (i = 8, 11).

Table 16-15 Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 8, 11)

Type	Bit No.	Bit Function	Bit Description
MCA error codes ¹	0-15	MCACOD	Bus error format: 1PPTRRRRIILL
Model specific errors	31:16	Reserved except for the following	0x001 - Address parity error 0x002 - HA Wrt buffer Data parity error 0x004 - HA Wrt byte enable parity error 0x008 - Corrected patrol scrub error 0x010 - Uncorrected patrol scrub error 0x020 - Corrected spare error 0x040 - Uncorrected spare error
Model specific errors	36-32	Other info	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first device error when corrected error is detected during normal read.
	37	Reserved	Reserved
	56-38		See Chapter 15, “Machine-Check Architecture,”
Status register validity indicators ¹	57-63		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, “Machine-Check Architecture,” for more information.

Table 16-16 Intel IMC MC Error Codes for IA32_MCi_MISC (i= 8, 11)

Type	Bit No.	Bit Function	Bit Description
MCA addr info ¹	0-8		See Chapter 15, “Machine-Check Architecture,”
Model specific errors	13:9		<ul style="list-style-type: none"> When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second device error when corrected error is detected during normal read. Otherwise contain parity error if MCi_Status indicates HA_WB_Data or HA_W_BE parity error.
Model specific errors	29-14		When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error bit mask.
Model specific errors	45-30		When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error bit mask.
	50:46		When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error failing rank.

Type	Bit No.	Bit Function	Bit Description
	55:51		When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error failing rank.
	58:56		When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error failing DIMM slot.
	61-59		When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error failing DIMM slot.
	62-63		Reserved

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

...

16. Updates to Chapter 17, Volume 3B

Change bars show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

...

17.4 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING OVERVIEW

P6 family processors introduced the ability to set breakpoints on taken branches, interrupts, and exceptions, and to single-step from one branch to the next. This capability has been modified and extended in the Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel® Atom™ processors to allow logging of branch trace messages in a branch trace store (BTS) buffer in memory.

See the following sections for processor specific implementation of last branch, interrupt and exception recording:

- Section 17.5, "Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel® Atom™ Processor Family)"
- Section 17.6, "Last Branch, Interrupt, and Exception Recording for Processors based on Intel® Microarchitecture code name Nehalem"
- Section 17.7, "Last Branch, Interrupt, and Exception Recording for Processors based on Intel® Microarchitecture code name Sandy Bridge"
- Section 17.8, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Intel® Microarchitecture code name Haswell"
- Section 17.9, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)"
- Section 17.10, "Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors)"
- Section 17.11, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)"
- Section 17.12, "Last Branch, Interrupt, and Exception Recording (P6 Family Processors)"

The following subsections of Section 17.4 describe common features of profiling branches. These features are generally enabled using the IA32_DEBUGCTL MSR (older processor may have implemented a subset or model-specific features, see definitions of MSR_DEBUGCTLA, MSR_DEBUGCTLB, MSR_DEBUGCTL).

...

17.4.8.1 LBR Stack and Intel® 64 Processors

LBR MSRs 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 branch records store 64-bit addresses; in compatibility mode, the upper 32-bits of last branch records are cleared.

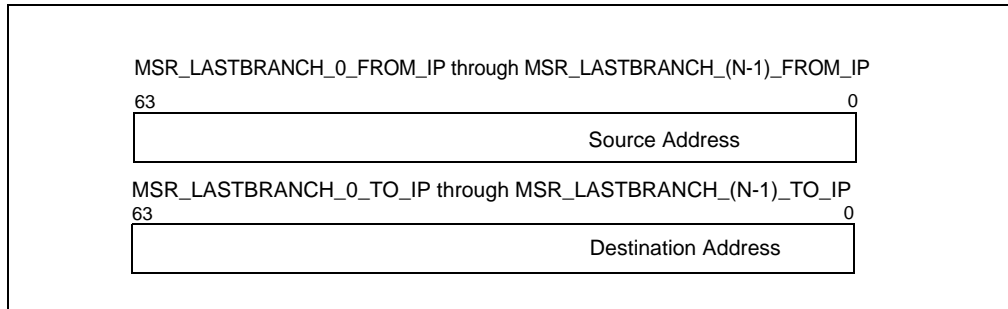


Figure 0-1. 64-bit Address Layout of LBR MSR

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. LBR flags are supported in the upper bits of 'FROM' register in the LBR stack. See LBR stack details below for flag support and definition.
- **000011B (64-bit EIP record format), Flags and TSX** — Stores 64-bit offset (effective address) of respective source/destination. LBR flags are supported in the upper bits of 'FROM' register in the LBR stack. TSX fields are also supported.

Processor's support for the architectural MSR IA32_PERF_CAPABILITIES is provided by CPUID.01H:ECX[PERF_CAPAB_MSR] (bit 15).

...

17.7 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Generally, all of the last branch record, interrupt and exception recording facility described in Section 17.6, "Last Branch, Interrupt, and Exception Recording for Processors based on Intel® Microarchitecture code name Nehalem", apply to processors based on Intel® microarchitecture code name Sandy Bridge. For processors based on Intel® microarchitecture code name Ivy Bridge, the same holds true.

One difference of note is that MSR_LBR_SELECT is shared between two logical processors in the same core. In Intel microarchitecture code name Sandy Bridge, each logical processor has its own MSR_LBR_SELECT. The filtering semantics for “Near_ind_jmp” and “Near_rel_jmp” has been enhanced, see Table 17-10.

Table 17-10 MSR_LBR_SELECT for Intel microarchitecture code name Sandy Bridge

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches occurring in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches occurring in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

17.8 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON INTEL® MICROARCHITECTURE CODE NAME HASWELL

Generally, all of the last branch record, interrupt and exception recording facility described in Section 17.7, “Last Branch, Interrupt, and Exception Recording for Processors based on Intel® Microarchitecture code name Sandy Bridge”, apply to next generation processors based on Intel® Microarchitecture code name Haswell.

The LBR facility also supports an alternate capability to profile call stack profiles. Configuring the LBR facility to conduct call stack profiling is by writing 1 to the MSR_LBR_SELECT.EN_CALLSTACK[bit 9]; see Table 17-11. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 17.7.

Table 17-11 MSR_LBR_SELECT for Intel microarchitecture code name Haswell

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches occurring in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches occurring 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	9		Enable LBR stack to use LIFO filtering to capture Call stack profile
Reserved	63:10		Must be zero

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.

17.8.1 LBR Stack Enhancement

Processors based on Intel microarchitecture code name Haswell provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is enumerated by IA32_PERF_CAPABILITIES[5:0] = 04H, and is shown in Table 17-12 and Table 17-7.

Table 17-12 IA32_LASTBRANCH_x_FROM_IP with TSX Information

Bit Field	Bit Offset	Access	Description
Data	47:0	R/O	The linear address of the branch instruction itself, this is the "branch from" address.
SIGN_EXT	60:48	R/O	Signed extension of bit 47 of this register.
TSX_ABORT	61	R/O	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/O	When set, indicates the entry occurred in a TSX region
MISPRED	63	R/O	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

17.9 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 MSR for the most recent taken branches, interrupts, and/or exceptions in MSR. A branch record consist of a branch-from and a branch-to instruction address.
- Send the branch records out on the system bus as branch trace messages (BTMs).
- Log BTMs in a memory-resident branch trace store (BTS) buffer.

To support these functions, the processor provides the following MSRs and related facilities:

- **MSR_DEBUGCTLA MSR** — Enables last branch, interrupt, and exception recording; single-stepping on taken branches; branch trace messages (BTMs); and branch trace store (BTS). This register is named DebugCtlMSR in the P6 family processors.
- **Debug store (DS) feature flag (CPUID.1:EDX.DS[bit 21])** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer.
- **CPL-qualified debug store (DS) feature flag (CPUID.1:ECX.DS-CPL[bit 4])** — Indicates that the processor provides a CPL-qualified debug store (DS) mechanism, which allows software to selectively skip sending and storing BTMs, according to specified current privilege level settings, into a memory-resident BTS buffer.
- **IA32_MISC_ENABLE MSR** — Indicates that the processor provides the BTS facilities.
- **Last branch record (LBR) stack** — The LBR stack is a circular stack that consists of four MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. The LBR stack consists of 16 MSR pairs (MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_15_FROM_IP and MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_15_TO_IP) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H].
- **Last branch record top-of-stack (TOS) pointer** — The TOS Pointer MSR contains a 2-bit pointer (0-3) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. This pointer becomes a 4-bit pointer (0-15) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H]. See also: Table 17-12, Figure 17-12, and Section 17.9.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **Last exception record** — See Section 17.9.3, “Last Exception Records.”

...

17.13.3 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.(EAX=07H, ECX=0H):EBX.TSC_ADJUST (bit 1).

...

17. Updates to Chapter 18, Volume 3B

Change bars show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

...

18.1 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, Pentium 4 and Intel Xeon processors introduced a new performance monitoring mechanism and new set of performance events.

The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, Pentium 4, and Intel Xeon processors 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. Processors based on Intel Core microarchitecture and Intel® Atom™ microarchitecture 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 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 18.3, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)." Non-architectural events for a given microarchitecture can not be enumerated using CPUID; and they are listed in Chapter 19, "Performance-Monitoring Events."

The second class of performance monitoring capabilities is referred to as architectural performance monitoring. This class supports the same counting and 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 18.2.

See also:

- Section 18.2, "Architectural Performance Monitoring"

- Section 18.3, “Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)”
- Section 18.4, “Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)”
- Section 18.5, “Performance Monitoring (Processors Based on Intel® Atom™ Microarchitecture)”
- Section 18.6, “Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Nehalem”
- Section 18.7, “Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Westmere”
- Section 18.8, “Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Sandy Bridge”
- Section 18.8.8, “Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility”
- Section 18.9, “3rd Generation Intel® Core™ Processor Performance Monitoring Facility”
- Section 18.10, “Next Generation Intel® Core™ Processor Performance Monitoring Facility”
- Section 18.11, “Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)”
- Section 18.12, “Performance Monitoring and Intel Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture”
- Section 18.15, “Performance Monitoring and Dual-Core Technology”
- Section 18.16, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache”
- Section 18.18, “Performance Monitoring (P6 Family Processor)”
- Section 18.19, “Performance Monitoring (Pentium Processors)”

...

18.2.2.2 Architectural Performance Monitoring Version 3 Facilities

The facilities provided by architectural performance monitoring version 1 and 2 are also supported by architectural performance monitoring version 3. Additionally version 3 provides enhancements to support 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. Specifically,

- CPUID leaf 0AH provides enumeration mechanisms to query:
 - The number of general-purpose performance counters (IA32_PMCx) is reported in CPUID.0AH:EAX[15:8], the bit width of general-purpose performance counters (see also Section 18.2.1.1) is reported in CPUID.0AH:EAX[23:16].
 - The bit vector representing the set of architectural performance monitoring events supported (see Section 18.2.3)
 - The number of fixed-function performance counters, the bit width of fixed-function performance counters (see also Section 18.2.1).
- Each general-purpose performance counter IA32_PMCx (starting at MSR address 0C1H) is associated with a corresponding IA32_PERFEVTSELx MSR (starting at MSR address 186H). The Bit field layout of IA32_PERFEVTSELx MSRs is defined architecturally in Figure 18-6.

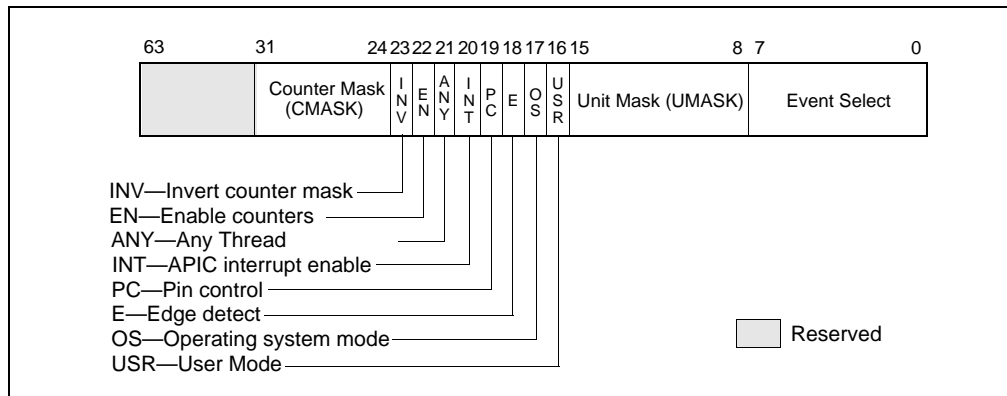


Figure 18-6 Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

Bit 21 (AnyThread) of IA32_PERFEVTSELx is supported in architectural performance monitoring version 3. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring across all logical processors sharing a processor core. When bit 21 is 0, the counter only increments the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring in the logical processor which programmed the IA32_PERFEVTSELx MSR.

- Each fixed-function performance counter IA32_FIXED_CTRx (starting at MSR address 309H) is configured by a 4-bit control block in the IA32_PERF_FIXED_CTR_CTRL MSR. The control block also allow thread-specificity configuration using an AnyThread bit. The layout of IA32_PERF_FIXED_CTR_CTRL MSR is shown.

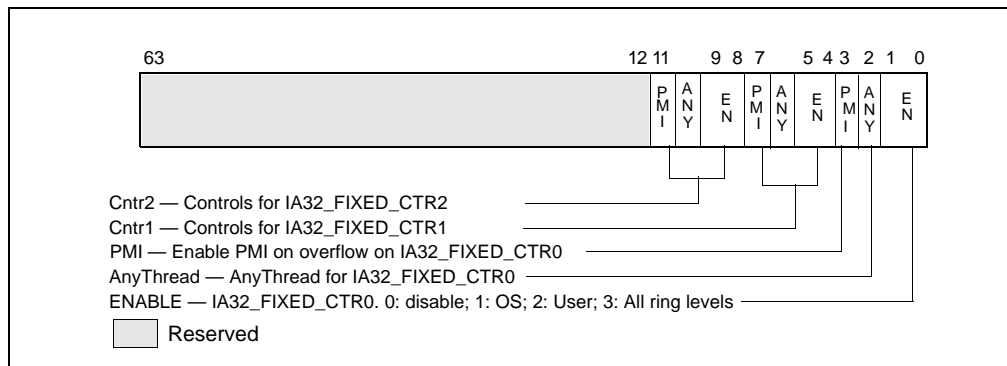


Figure 18-7 Layout of IA32_PERF_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3

Each control block for a fixed-function performance counter provides a **AnyThread** (bit position $2 + 4 * N$, $N = 0, 1$, etc.) bit. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the ENABLE setting of the corresponding control block of IA32_PERF_FIXED_CTR_CTRL) occurring across all logical processors sharing a processor core. When an **AnyThread** bit is 0 in IA32_PERF_FIXED_CTR_CTRL, the corresponding fixed counter only increments the associated event conditions occurring in the logical processor which programmed the IA32_PERF_FIXED_CTR_CTRL MSR.

- The IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_OVF_CTRL MSRs provide single-bit controls/status for each general-purpose and fixed-function performance counter. Figure 18-8 and Figure 18-9 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 Intel Atom processor family supports two general-purpose performance monitoring counters (i.e. $N = 2$ in Figure 18-9), other processor families in Intel 64 architecture may support a different value of N in Figure 18-9. The number N is reported by CPUID.0AH:EAX[15:8]. The Intel Core i7 processor supports four general-purpose performance monitoring counters (i.e. $N = 4$ in Figure 18-9).

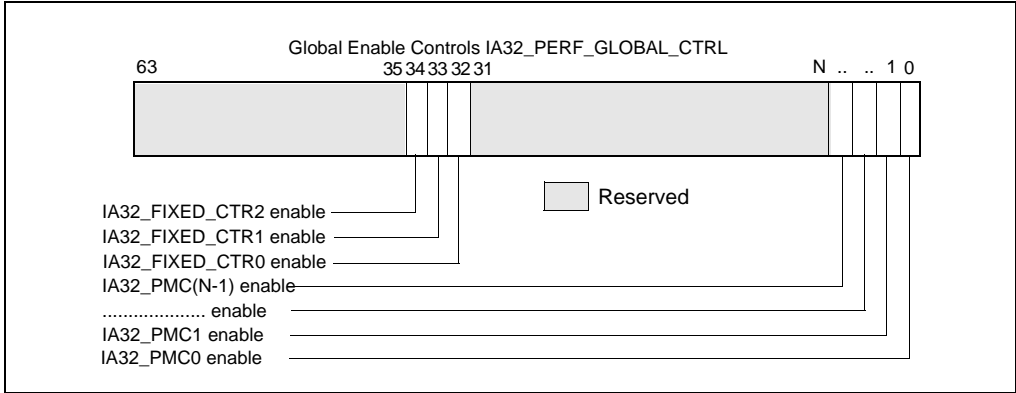


Figure 18-8 Layout of Global Performance Monitoring Control MSR

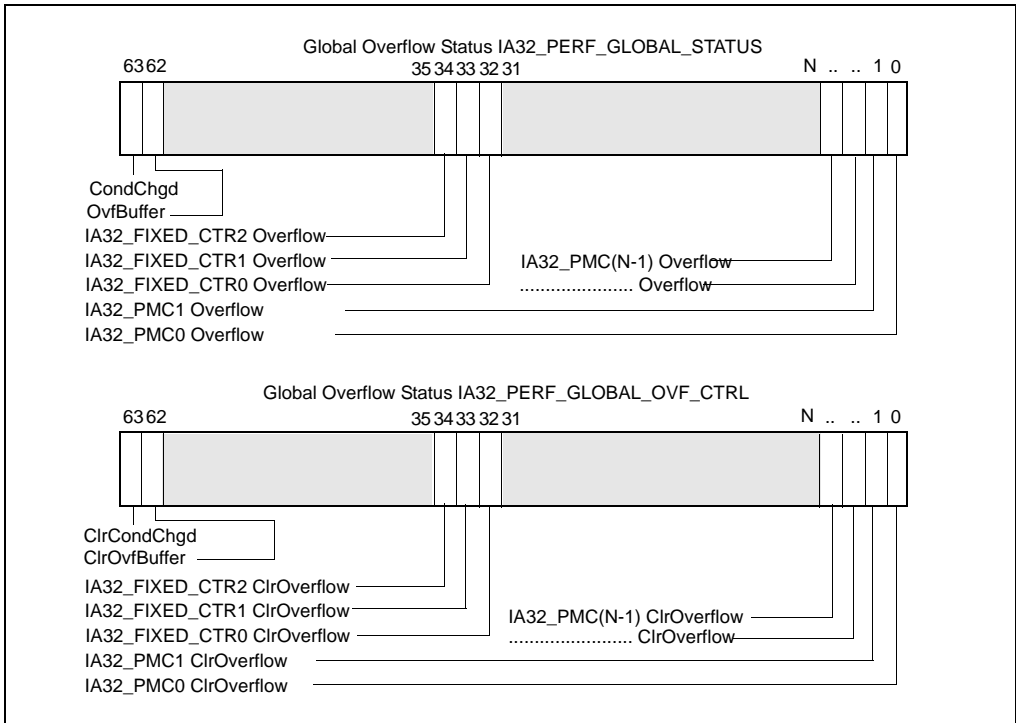


Figure 18-9 Global Performance Monitoring Overflow Status and Control MSRs

18.6.1.1 Precise Event Based Sampling (PEBS)

All four 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 Intel microarchitecture code name Nehalem is shown in Figure 18-15.

When a counter is enabled to capture machine state (PEBS_EN_PMCx = 1), the processor will write machine state information to a memory buffer specified by software as detailed below. When the counter IA32_PMCx overflows from maximum count to zero, the PEBS hardware is armed.

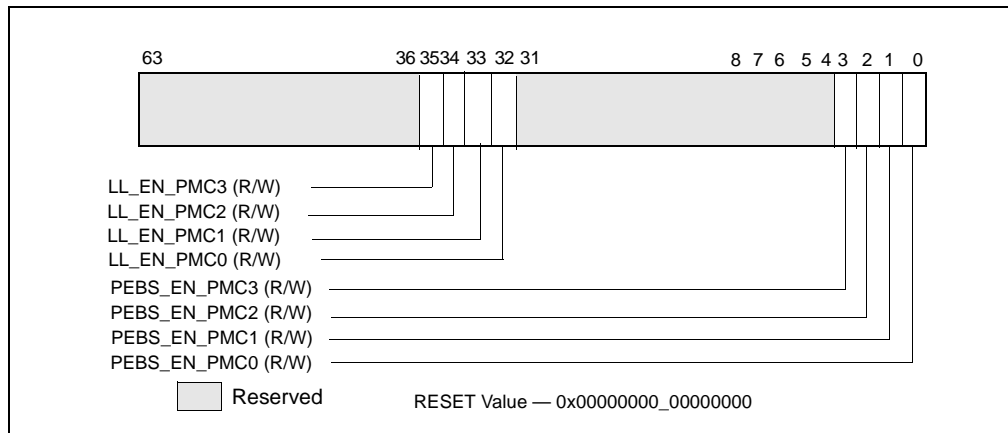


Figure 18-15 Layout of IA32_PEBS_ENABLE MSR

Upon occurrence of the next PEBS event, the PEBS hardware triggers an assist and causes a PEBS record to be written. The format of the PEBS record is indicated by the bit field IA32_PERF_CAPABILITIES[11:8] (see Figure 18-40).

The behavior of PEBS assists is reported by IA32_PERF_CAPABILITIES[6] (see Figure 18-40). 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 18-12, and each field in the PEBS record is 64 bits long. The PEBS record format, along with debug/store area storage format, does not change regardless of IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

...

18.6.1.3 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 18-14 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 18-14 Off-Core Response Event Encoding

Event code in IA32_PERFEVTSELx	Mask Value in IA32_PERFEVTSELx	Required Off-core Response MSR
0xB7	0x01	MSR_OFFCORE_RSP_0 (address 0x1A6)

The layout of MSR_OFFCORE_RSP_0 is shown in Figure 18-18. Bits 7:0 specifies the request type of a transaction request to the uncore. Bits 15:8 specifies the response of the uncore subsystem.

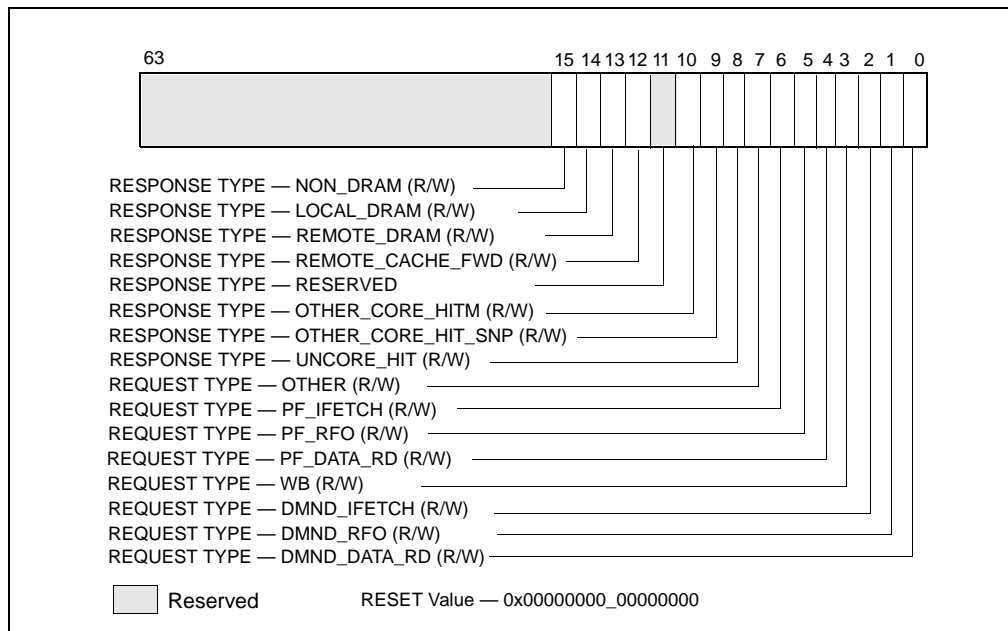


Figure 18-18 Layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 to Configure Off-core Response Events

...

18.8 PERFORMANCE MONITORING FOR PROCESSORS BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel® Xeon® processor E3-1200 family are based on Intel microarchitecture code name Sandy Bridge; 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 18.2.2.2) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring events and non-architectural monitoring events are programmed using fixed counters and programmable counters/event select MSRS described in Section 18.2.2.2.

The core PMU's capability is similar to those described in Section 18.6.1 and Section 18.7, with some differences and enhancements relative to Intel microarchitecture code name Westmere summarized in Table 18-19.

Table 18-19 Core PMU Comparison

Box	Sandy Bridge	Westmere	Comment
# of Fixed counters per thread	3	3	Use CPUID to enumerate # of counters.
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48 , W: 32/48	R:48, W:32	See Section 18.2.2.3.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4	Use CPUID to enumerate # of counters.
Precise Event Based Sampling (PEBS) Events	See Table 18-21	See Table 18-10	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 18.8.4.2; Data source encoding, STLB miss encoding, Lock transaction encoding	Data source encoding	
PEBS-Precise Store	Section 18.8.4.3	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.

...

18.8.5 Off-core Response Performance Monitoring

The core PMU in processors based on Intel microarchitecture code name Sandy Bridge 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 18-24 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 18-24 Off-Core Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-3	0xB7	0x01	MSR_OFFCORE_RSP_0 (address 0x1A6)
PMC0-3	0xBB	0x01	MSR_OFFCORE_RSP_1 (address 0x1A7)

The layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 are shown in Figure 18-30 and Figure 18-31. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

...

18.8.7 Intel® Xeon® Processor E5 Family Performance Monitoring Facility

The Intel® Xeon® Processor E5 Family (and Intel® Core™ i7-3930K Processor) are based on Intel microarchitecture code name Sandy Bridge. While the processor cores share the same microarchitecture as those of the Intel® Xeon® Processor E3 Family and second 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 18.8.8.

Thus, the performance monitoring facilities in the processor core generally are the same as those described in Section 18.8 through Section 18.8.5. However, the MSR_OFFCORE_RSP_0/MSR_OFFCORE_RSP_1 Response Supplier Info field shown in Table 18-26 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 18-29. Additionally, there are some small differences in the non-architectural performance monitoring events (see Table 19-7).

Table 18-29 MSR_OFFCORE_RSP_x Supplier Info Field Definitions

Subtype	Bit Name	Offset	Description
Common	Any	16	(R/W). Catch all value for any response types.
Supplier Info	NO_SUPP	17	(R/W). No Supplier Information available
	LLC_HITM	18	(R/W). M-state initial lookup stat in L3.
	LLC_HITE	19	(R/W). E-state
	LLC_HITS	20	(R/W). S-state
	LLC_HITF	21	(R/W). F-state
	LOCAL	22	(R/W). Local DRAM Controller
	Remote	30:23	(R/W); Remote DRAM Controller (either all 0s or all 1s)

...

18.9 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 Intel® microarchitecture code name Ivy Bridge. The performance monitoring facilities in the processor core generally are the same as those described in Section 18.8 through Section 18.8.5. The non-architectural performance monitoring events supported by the processor core are listed in Table 19-7.

18.10 NEXT GENERATION INTEL® CORE™ PROCESSOR PERFORMANCE MONITORING FACILITY

The Next Generation Intel® Core™ processor is based on Intel® microarchitecture code name Haswell. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 18.2.2.2) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring events and non-architectural monitoring events are programmed using fixed counters and programmable counters/event select MSRS as described in Section 18.2.2.2.

The core PMU's capability is similar to those described in Section 18.8, with some differences and enhancements summarized in Table 18-31.

Table 18-31 Core PMU Comparison

Box	Haswell	Sandy Bridge	Comment
# of Fixed counters per thread	3	3	
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48 , W: 32/48	R:48 , W: 32/48	See Section 18.2.2.3.
# 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 enumerate # of counters.
Precise Event Based Sampling (PEBS) Events	See Table 18-21	See Table 18-21	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 18.8.4.2;	See Section 18.8.4.2;	
PEBS-Precise Store	No, replaced by Data Address profiling	Section 18.8.4.3	
PEBS-PDIR	yes (using precise INST_RETIRED.ALL)	yes (using precise INST_RETIRED.ALL)	
PEBS-EventingIP	yes	no	
Data Address Profiling	yes	no	
LBR Profiling	yes	yes	
Call Stack Profiling	yes, see Section 17.8	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 18.10.5;	no	

18.10.1 Precise Event Based Sampling (PEBS) Facility

The PEBS facility in the Next Generation Intel Core processor is similar to those in processors based on Intel microarchitecture code name Sandy Bridge, with several enhanced features. The key components and differences of PEBS facility relative to Intel microarchitecture code name Sandy Bridge is summarized in Table 18-32.

Table 18-32 PEBS Facility Comparison

Box	Haswell	Sandy Bridge	Comment
Valid IA32_PMCx	PMCO-PMC3	PMCO-PMC3	No PEBS on PMC4-PMC7
PEBS Buffer Programming	Section 18.6.1.1	Section 18.6.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 18-29	Figure 18-15	
PEBS record layout	Table 18-33, Enhanced fields at offsets 98H, A0H, A8H, B0H	Table 18-12, Enhanced fields at offsets 98H, A0H, A8H	
PEBS Events	See Table 18-21	See Table 18-21	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 18-22	Table 18-22	
PEBS-Precise Store	no, replaced by data address profiling	yes; see Section 18.8.4.3	
PEBS-PDIR	yes	yes	IA32_PMC1 only
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.

18.10.2 PEBS Data Format

The PEBS record format for the Next Generation Intel Core processor is shown in Table 18-33. The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 18-33 PEBS Record Format for Next Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
0x0	R/EFLAGS	0x60	R10
0x8	R/EIP	0x68	R11
0x10	R/EAX	0x70	R12
0x18	R/EBX	0x78	R13
0x20	R/ECX	0x80	R14
0x28	R/EDX	0x88	R15
0x30	R/ESI	0x90	IA32_PERF_GLOBAL_STATUS
0x38	R/EDI	0x98	Data Linear Address
0x40	R/EBP	0xA0	Data Source Encoding
0x48	R/ESP	0xA8	Latency value (core cycles)

Table 18-33 PEBS Record Format for Next Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
0x50	R8	0xB0	EventingIP
0x58	R9		

The layout of PEBS records are almost identical to those shown in Table 18-12. Offset 0xB0 is a new field that records the eventing IP address of the retired instruction that triggered the PEBS assist.

The PEBS records at offsets 0x98, 0xA0, and 0xAB record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 18.8.4.2), PDIR (Section 18.8.4.4), and precise store (Section 18.8.4.3).

In the core PMU of the next generation 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 0x98, 0xA0, and 0xAB.

18.10.3 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 next generation processor supports the following events configured to use PEBS:

Table 18-34 Precise Events That Supports Data Linear Address Profiling

Event Name	Event Name
MEM_UOPS_RETIREDD.STLB_MISS_LOADS	MEM_UOPS_RETIREDD.STLB_MISS_STORES
MEM_UOPS_RETIREDD.LOCK_LOADS	MEM_UOPS_RETIREDD.LOCK_STORES
MEM_UOPS_RETIREDD.SPLIT_LOADS	MEM_UOPS_RETIREDD.SPLIT_STORES
MEM_UOPS_RETIREDD.ALL_LOADS	MEM_UOPS_RETIREDD.ALL_STORES
MEM_LOAD_UOPS_RETIREDD.L1_HIT	MEM_LOAD_UOPS_RETIREDD.L2_HIT
MEM_LOAD_UOPS_RETIREDD.LLC_HIT	MEM_LOAD_UOPS_RETIREDD.L1_MISS
MEM_LOAD_UOPS_RETIREDD.L2_MISS	MEM_LOAD_UOPS_RETIREDD.LLC_MISS
MEM_LOAD_UOPS_RETIREDD.HIT_LFB	MEM_LOAD_UOPS_LLC_HIT_RETIREDD.XSNP_MISS
MEM_LOAD_UOPS_LLC_HIT_RETIREDD.XSNP_HIT	MEM_LOAD_UOPS_LLC_HIT_RETIREDD.XSNP_HITM
UOPS_RETIREDD.ALL (if load or store is tagged)	MEM_LOAD_UOPS_MISC_RETIREDD.UC
MEM_LOAD_UOPS_LLC_HIT_RETIREDD.XSNP_NONE	MEM_LOAD_UOPS_LLC_MISS_RETIREDD.LOCAL_DRAM
MEM_LOAD_UOPS_LLC_MISS_RETIREDD.LOCAL_DRAM_SNP_HIT	MEM_LOAD_UOPS_LLC_MISS_RETIREDD.REMOTE_DRAM
MEM_LOAD_UOPS_LLC_MISS_RETIREDD.REMOTE_DRAM_SNP_HIT	MEM_LOAD_UOPS_LLC_MISS_RETIREDD.REMOTE_HITM
MEM_LOAD_UOPS_LLC_MISS_RETIREDD.REMOTE_FWD	MEM_LOAD_UOPS_MISC_RETIREDD.NON_DRAM
MEM_LOAD_UOPS_MISC_RETIREDD.LLC_MISS	

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 the an event listed in Table 18-34 using any one of IA32_PERFEVTSELO-IA32_PERFEVTSEL3.
- Set the corresponding IA32_PEBS_ENABLE.PEBS_EN_CTRx bit and IA32_PEBS_ENABLE[63]. This enables the corresponding IA32_PMCx as a PEBS counter and enables the DataLA facility, respectively.

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 18-35.

Table 18-35 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	<ul style="list-style-type: none"> ▪ 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.LOCK_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 18-34.
Reserved	A8H	Always zero

18.10.3.1 EventingIP Record

The PEBS record layout for processors based on Intel microarchitecture code name Haswell adds a new field at offset OBOH. 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.

18.10.4 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 18.8.5. The event codes are listed in Table 18-24. 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 18-36.
- Supplier information (bits 30:16): see Table 18-26.
- Snoop response information (bits 37:31): see Table 18-27.

Table 18-36 MSR_OFFCORE_RSP_x Request_Type Definition (Haswell)

Bit Name	Offset	Description
DMND_DATA_RD	0	(R/W). 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	(R/W). 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	(R/W). Counts the number of demand and DCU prefetch instruction cacheline reads. Does not count L2 code read prefetches.
Reserved	3	Reserved
PF_DATA_RD	4	(R/W). Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	(R/W). Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	(R/W). Counts the number of code reads generated by L2 prefetchers.
Reserved	7-14	Reserved
OTHER	15	(R/W). Any other request that crosses IDI, including I/O.

18.10.5 Performance Monitoring and Intel® TSX

Intel TSX allows multi-threaded program to make forward progress with less synchronization overhead. If a target workload for performance monitoring contains instruction streams using Intel TSX, the transaction code regions in the workload may encounter the following scenarios: (a) The transactional code on some logical processors may execute speculatively and commit results with synchronization overhead elided, or (b) the speculatively executed transaction code aborts and the transactional code will restart normal execution experiencing the cost of the synchronization primitive. For details of transactional code behavior of Intel TSX, see Chapter 8 of *Intel® Architecture Instruction Set Extensions Programming Reference*.

If a processor supports Intel TSX, the core PMU enhances its IA32_PERFEVTSELx MSR with two additional bit fields for event filtering. Support for Intel TSX is indicated by either (a) CPUID.(EAX=7, ECX=0):RTM[bit 11]=1, or (b) if CPUID.07H.EBX.HLE [bit 4] = 1. The TSX-enhanced layout of IA32_PERFEVTSELx is shown in Figure 18-34. The two additional bit fields are:

- **IN_TX** (bit 32): When set, the counter will only include counts that occurred inside a transactional region, regardless of whether that region was aborted or committed. This bit may only be set if the processor supports HLE or RTM.
- **IN_TXCP** (bit 33): When set, the counter will not include counts that occurred inside of an aborted transactional region. This bit may only be set if the processor supports HLE or RTM. This bit may only be set for IA32_PERFEVTSEL2.

When the IA32_PERFEVTSELx MSR is programmed with both IN_TX=0 and IN_TXCP=0 on a processor that supports Intel TSX, the result in a counter may include detectable conditions associated with a transaction code region for its aborted execution (if any) and completed execution.

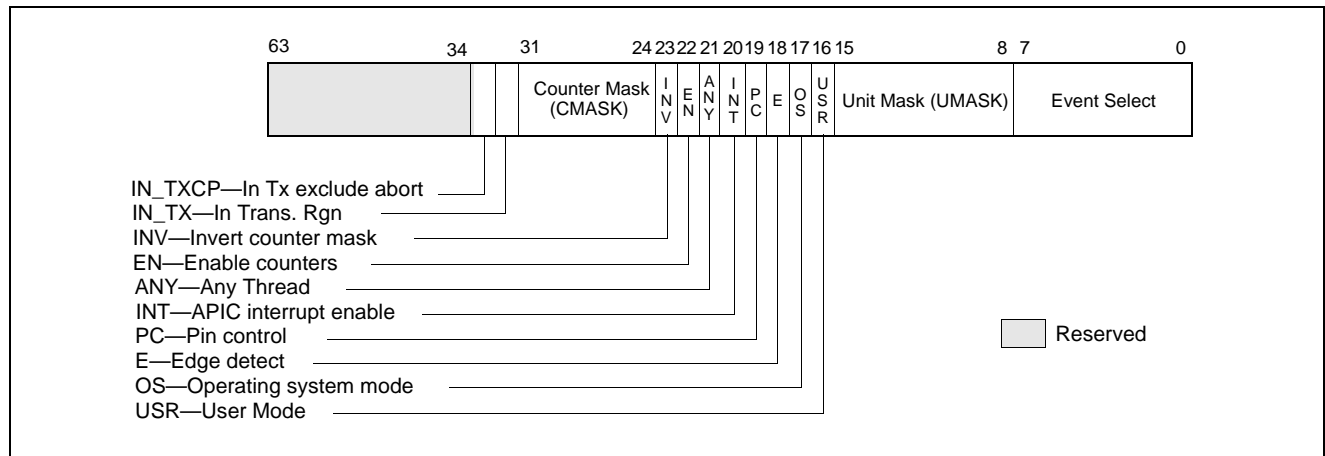


Figure 18-34 Layout of IA32_PERFEVTSELx MSRs Supporting Intel TSX

A common usage of setting IN_TXCP=1 is to capture the number of events that were discarded due to a transactional abort. With IA32_PMC2 configured to count in such a manner, then when a TX 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 code region are discarded. The following example illustrates using three counters to drill down cycles spent inside and outside of transactional regions:

- Program IA32_PERFEVTSEL2 to count Unhalted_Core_Cycles with (IN_TXCP=1, IN_TX=0), such that IA32_PMC2 will count cycles spent due to aborted TSX transactions;
- Program IA32_PERFEVTSEL0 to count Unhalted_Core_Cycles with (IN_TXCP=0, IN_TX=1), such that IA32_PMC0 will count cycles spent by the transactional code regions;
- Program IA32_PERFEVTSEL1 to count Unhalted_Core_Cycles with (IN_TXCP=0, IN_TX=0), such that IA32_PMC1 will count total cycles spent by the non-transactional code and transactional code regions.

Additionally, a number of performance events are solely focused on characterizing the execution of Intel TSX transactional code, they are listed in Table 19-3.

18.10.5.1 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 in Table 19-3 also support using PEBS facility to capture additional information. They are:

- HLE_RETIREDA.BORT ED (encoding 0xc8 mask 0x4),
- RTM_RETIREDA.BORT ED (encoding 0xc9 mask 0x4).

A transactional abort (HLE_RETIREDA.BORT ED,RTM_RETIREDA.BORT ED) 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 pending 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 0xB8 bits[33:32] to indicate an HLE abort or an RTM abort):

- Offset 0x98 Data Linear Address (if the uop that triggered PEBS was a load or a store),
- Offset 0xB0 EventingIP,
- Offset 0xB8 TX Abort Information

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 18-37.

Table 18-37 TX Abort Information Field Definition

Bit Name	Offset	Description
Cycles_Last_Block	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 transactional abort was associated with the instruction corresponding to the eventing IP
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. This value matches the RTM Abort Status Information in EAX bit[1]
Memory_Data_Conflict	37	If set, another logical processor conflicted with a memory address that was part of the transactional region that aborted. Matches RTM Abort Encoding EAX bit[2]
Capacity	38	Matches RTM Abort Encoding EAX bit[3]
Reserved	63:39	Reserved

18.10.6 Uncore Performance Monitoring Facilities in Next Generation Intel® Core™ Processors

The uncore sub-system in the Next 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 18.8.6.

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 18-32.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 18-33 shows the layout of the uncore domain global control.

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 18-28 summarizes the number MSRs for uncore PMU for each box.

Table 18-37 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 35-12 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 are listed in Table 19-4.

...

18. Updates to Chapter 19, Volume 3B

Change bars show changes to Chapter 19 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

...

This chapter lists the performance-monitoring events that can be monitored with the Intel 64 or IA-32 processors. The ability to monitor performance events and the events that can be monitored in these processors are mostly model-specific, except for architectural performance events, described in Section 19.1.

Non-architectural performance events (i.e. model-specific events) are listed for each generation of microarchitecture:

- Section 19.2 - Next Generation Intel® Core™ Processors
- Section 19.3 - Processors based on Intel® microarchitecture code name Ivy Bridge
- Section 19.4 - Processors based on Intel® microarchitecture code name Sandy Bridge
- Section 19.5 - Processors based on Intel® microarchitecture code name Nehalem
- Section 19.6 - Processors based on Intel® microarchitecture code name Westmere
- Section 19.7 - Processors based on Enhanced Intel® Core™ microarchitecture
- Section 19.8 - Processors based on Intel® Core™ microarchitecture
- Section 19.9 - Processors based on Intel® Atom™ microarchitecture
- Section 19.10 - Intel® Core™ Solo and Intel® Core™ Duo processors
- Section 19.11 - Processors based on Intel NetBurst® microarchitecture
- Section 19.12 - Pentium® M family processors
- Section 19.13 - P6 family processors
- Section 19.14 - Pentium® processors

NOTE

These performance-monitoring events are intended to be used as guides for performance tuning. The counter values reported by the performance-monitoring events are approximate and believed to be useful as relative guides for tuning software. Known discrepancies are documented where applicable.

...

19.2 PERFORMANCE MONITORING EVENTS FOR NEXT GENERATION INTEL® CORE™ PROCESSORS

The Next Generation Intel® Core™ Processors are based on the Intel microarchitecture code name Haswell. They support the architectural performance-monitoring events listed in Table 19-1. Non-architectural performance-monitoring events in the processor core are listed in Table 19-5. The events in Table 19-5 apply to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_3CH and 06_45H.

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	LD_BLOCKS.STORE_FORWARD	loads blocked by overlapping with store buffer that cannot be forwarded .	
05H	01H	MISALIGN_MEM_REF.LOADS	Speculative cache-line split load uops dispatched to L1D.	
05H	02H	MISALIGN_MEM_REF.STORES	Speculative cache-line split Store-address uops dispatched to L1D.	
07H	01H	LD_BLOCKS_PARTIAL.ADDRESS_ALIAS	False dependencies in MOB due to partial compare on address.	
08H	01H	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Misses in all TLB levels that cause a page walk of any page size.	
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED_4K	Completed page walks due to demand load misses that caused 4K page walks in any TLB levels.	
08H	04H	DTLB_LOAD_MISSES.WALK_COMPLETED_2M_4M	Completed page walks due to demand load misses that caused 2M/4M page walks in any TLB levels.	
08H	0EH	DTLB_LOAD_MISSES.WALK_COMPLETED	Completed page walks in any TLB of any page size due to demand load misses	
08H	10H	DTLB_LOAD_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
08H	20H	DTLB_LOAD_MISSES.STLB_HIT_4K	Load misses that missed DTLB but hit STLB (4K).	
08H	40H	DTLB_LOAD_MISSES.STLB_HIT_2M	Load misses that missed DTLB but hit STLB (2M).	
08H	60H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.	
08H	80H	DTLB_LOAD_MISSES.PDE_CACHE_MISS	DTLB demand load misses with low part of linear-to-physical address translation missed	
0DH	03H	INT_MISC.RECOVERY_CYCLES	Cycles waiting to recover after Machine Clears except JEClear. Set Cmask= 1.	Set Edge to count occurrences
0EH	01H	UOPS_ISSUED.ANY	Increments each cycle the # of Uops issued by the RAT to RS. Set Cmask = 1, Inv = 1, Any= 1 to count stalled cycles of this core.	Set Cmask = 1, Inv = 1 to count stalled cycles
0EH	10H	UOPS_ISSUED.FLAGS_MERGE	Number of flags-merge uops allocated. Such uops adds delay.	

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
0EH	20H	UOPS_ISSUED.SLOW_LEA	Number of slow LEA or similar uops allocated. Such uop has 3 sources (e.g. 2 sources + immediate) regardless if as a result of LEA instruction or not.	
0EH	40H	UOPS_ISSUED.SINGLE_MUL	Number of multiply packed/scalar single precision uops allocated.	
24H	21H	L2_RQSTS.DEMAND_DATA_RD_MISS	Demand Data Read requests that missed L2, no rejects.	
24H	41H	L2_RQSTS.DEMAND_DATA_RD_HIT	Demand Data Read requests that hit L2 cache.	
24H	E1H	L2_RQSTS.ALL_DEMAND_DATA_RD	Counts any demand and L1 HW prefetch data load requests to L2.	
24H	42H	L2_RQSTS.RFO_HIT	Counts the number of store RFO requests that hit the L2 cache.	
24H	22H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache.	
24H	E2H	L2_RQSTS.ALL_RFO	Counts all L2 store RFO requests.	
24H	44H	L2_RQSTS.CODE_RD_HIT	Number of instruction fetches that hit the L2 cache.	
24H	24H	L2_RQSTS.CODE_RD_MISS	Number of instruction fetches that missed the L2 cache.	
24H	27H	L2_RQSTS.ALL_DEMAND_MISS	Demand requests that miss L2 cache.	
24H	E7H	L2_RQSTS.ALL_DEMAND_REFERENCES	Demand requests to L2 cache.	
24H	E4H	L2_RQSTS.ALL_CODE_RD	Counts all L2 code requests.	
24H	50H	L2_RQSTS.L2_PF_HIT	Counts all L2 HW prefetcher requests that hit L2.	
24H	30H	L2_RQSTS.L2_PF_MISS	Counts all L2 HW prefetcher requests that missed L2.	
24H	F8H	L2_RQSTS.ALL_PF	Counts all L2 HW prefetcher requests.	
24H	3FH	L2_RQSTS.MISS	All requests that missed L2.	
24H	FFH	L2_RQSTS.REFERENCES	All requests to L2 cache.	
27H	50H	L2_DEMAND_RQSTS.WB_HIT	Not rejected writebacks that hit L2 cache	
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache.	see Table 19-1
2EH	41H	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache.	see Table 19-1
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Counts the number of thread cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. The core frequency may change from time to time due to power or thermal throttling.	see Table 19-1
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK	Increments at the frequency of XCLK (100 MHz) when not halted.	see Table 19-1

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle. Set Cmaks = 1 and Edge =1 to count occurrences.	Counter 2 only; Set Cmask = 1 to count cycles.
49H	01H	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Miss in all TLB levels causes an page walk of any page size (4K/2M/4M/1G).	
49H	02H	DTLB_STORE_MISSES.WALK_COMPLETED_4K	Completed page walks due to store misses in one or more TLB levels of 4K page structure.	
49H	04H	DTLB_STORE_MISSES.WALK_COMPLETED_2M_4M	Completed page walks due to store misses in one or more TLB levels of 2M/4M page structure.	
49H	0EH	DTLB_STORE_MISSES.WALK_COMPLETED	Completed page walks due to store miss in any TLB levels of any page size (4K/2M/4M/1G).	
49H	10H	DTLB_STORE_MISSES.WALK_DURATION	Cycles PMH is busy with this walk.	
49H	20H	DTLB_STORE_MISSES.STLB_HIT_4K	Store misses that missed DTLB but hit STLB (4K).	
49H	40H	DTLB_STORE_MISSES.STLB_HIT_2M	Store misses that missed DTLB but hit STLB (2M).	
49H	60H	DTLB_STORE_MISSES.STLB_HIT	Store operations that miss the first TLB level but hit the second and do not cause page walks.	
49H	80H	DTLB_STORE_MISSES.PDE_CACHE_MISS	DTLB store misses with low part of linear-to-physical address translation missed.	
4CH	01H	LOAD_HIT_PRE.SW_PF	Non-SW-prefetch load dispatches that hit fill buffer allocated for S/W prefetch.	
4CH	02H	LOAD_HIT_PRE.HW_PF	Non-SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.	
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
58H	04H	MOVE_ELIMINATION.INT_NOT_ELIMINATED	Number of integer Move Elimination candidate uops that were not eliminated.	
58H	08H	MOVE_ELIMINATION.SIMD_NOT_ELIMINATED	Number of SIMD Move Elimination candidate uops that were not eliminated.	
58H	01H	MOVE_ELIMINATION.INT_ELIMINATED	Number of integer Move Elimination candidate uops that were eliminated.	
58H	02H	MOVE_ELIMINATION.SIMD_ELIMINATED	Number of SIMD Move Elimination candidate uops that were eliminated.	
5CH	01H	CPL_CYCLES.RING0	Unhalted core cycles when the thread is in ring 0.	Use Edge to count transition
5CH	02H	CPL_CYCLES.RING123	Unhalted core cycles when the thread is not in ring 0.	
5EH	01H	RS_EVENTS.EMPTY_CYCLES	Cycles the RS is empty for the thread.	
63H	01H	LOCK_CYCLES.SPLIT_LOCK_UC_LOCK_DURATION	Cycles in which the L1D and L2 are locked, due to a UC lock or split lock.	

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
63H	02H	LOCK_CYCLES.CACHE_LOCK_DURATION	Cycles in which the L1D is locked.	
79H	02H	IDQ.EMPTY	Counts cycles the IDQ is empty.	
79H	04H	IDQ.MITE_UOPS	Increment each cycle # of uops delivered to IDQ from MITE path. Set Cmask = 1 to count cycles.	Can combine Umask 04H and 20H
79H	08H	IDQ.DSB_UOPS	Increment each cycle. # of uops delivered to IDQ from DSB path. Set Cmask = 1 to count cycles.	Can combine Umask 08H and 10H
79H	10H	IDQ.MS_DSB_UOPS	Increment each cycle # of uops delivered to IDQ when MS_busy by DSB. Set Cmask = 1 to count cycles. Add Edge=1 to count # of delivery.	Can combine Umask 04H, 08H
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ when MS_busy by MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H, 08H
79H	30H	IDQ.MS_UOPS	Increment each cycle # of uops delivered to IDQ from MS by either DSB or MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H, 08H
79H	18H	IDQ.ALL_DSB_CYCLES_ANY_UOPS	Counts cycles DSB is delivered at least one uops. Set Cmask = 1.	
79H	18H	IDQ.ALL_DSB_CYCLES_4_UOPS	Counts cycles DSB is delivered four uops. Set Cmask = 4.	
79H	24H	IDQ.ALL_MITE_CYCLES_ANY_UOPS	Counts cycles MITE is delivered at least one uops. Set Cmask = 1.	
79H	24H	IDQ.ALL_MITE_CYCLES_4_UOPS	Counts cycles MITE is delivered four uops. Set Cmask = 4.	
79H	3CH	IDQ.MITE_ALL_UOPS	# of uops delivered to IDQ from any path.	
80H	02H	ICACHE.MISSES	Number of Instruction Cache, Streaming Buffer and Victim Cache Misses. Includes UC accesses.	
85H	01H	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses in ITLB that causes a page walk of any page size.	
85H	02H	ITLB_MISSES.WALK_COMPLETED_4K	Completed page walks due to misses in ITLB 4K page entries.	
85H	04H	ITLB_MISSES.WALK_COMPLETED_2M_4M	Completed page walks due to misses in ITLB 2M/4M page entries.	
85H	0EH	ITLB_MISSES.WALK_COMPLETED	Completed page walks in ITLB of any page size.	
85H	10H	ITLB_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
85H	20H	ITLB_MISSES.STLB_HIT_4K	ITLB misses that hit STLB (4K).	
85H	40H	ITLB_MISSES.STLB_HIT_2M	ITLB misses that hit STLB (2M).	
85H	60H	ITLB_MISSES.STLB_HIT	ITLB misses that hit STLB. No page walk.	

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
87H	01H	ILD_STALL.LCP	Stalls caused by changing prefix length of the instruction.	
87H	04H	ILD_STALL.IQ_FULL	Stall cycles due to IQ is full.	
88H	01H	BR_INST_EXEC.COND	Qualify conditional near branch instructions executed, but not necessarily retired.	Must combine with umask 40H, 80H
88H	02H	BR_INST_EXEC.DIRECT_JMP	Qualify all unconditional near branch instructions excluding calls and indirect branches.	Must combine with umask 80H
88H	04H	BR_INST_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify executed indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
88H	08H	BR_INST_EXEC.RETURN_NEAR	Qualify indirect near branches that have a return mnemonic.	Must combine with umask 80H
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Qualify unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Qualify indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
88H	40H	BR_INST_EXEC.NONTAKEN	Qualify non-taken near branches executed.	Applicable to umask 01H only
88H	80H	BR_INST_EXEC.TAKEN	Qualify taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
88H	FFH	BR_INST_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	
89H	01H	BR_MISP_EXEC.COND	Qualify conditional near branch instructions mispredicted.	Must combine with umask 40H, 80H
89H	04H	BR_MISP_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify mispredicted indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Qualify mispredicted indirect near branches that have a return mnemonic.	Must combine with umask 80H
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Qualify mispredicted unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Qualify mispredicted indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
89H	40H	BR_MISP_EXEC.NONTAKEN	Qualify mispredicted non-taken near branches executed.	Applicable to umask 01H only
89H	80H	BR_MISP_EXEC.TAKEN	Qualify mispredicted taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
89H	FFH	BR_MISP_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CORE	Count number of non-delivered uops to RAT per thread.	Use Cmask to qualify uop b/w
A1H	01H	UOPS_EXECUTED_PORT.PORT_0	Cycles which a Uop is dispatched on port 0 in this thread.	Set AnyThread to count per core

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A1H	02H	UOPS_EXECUTED_PORT.PORT_1	Cycles which a Uop is dispatched on port 1 in this thread.	Set AnyThread to count per core
A1H	04H	UOPS_EXECUTED_PORT.PORT_2	Cycles which a uop is dispatched on port 2 in this thread.	Set AnyThread to count per core
A1H	08H	UOPS_EXECUTED_PORT.PORT_3	Cycles which a uop is dispatched on port 3 in this thread.	Set AnyThread to count per core
A1H	10H	UOPS_EXECUTED_PORT.PORT_4	Cycles which a uop is dispatched on port 4 in this thread.	Set AnyThread to count per core
A1H	20H	UOPS_EXECUTED_PORT.PORT_5	Cycles which a uop is dispatched on port 5 in this thread.	Set AnyThread to count per core
A1H	40H	UOPS_EXECUTED_PORT.PORT_6	Cycles which a Uop is dispatched on port 6 in this thread.	Set AnyThread to count per core
A1H	80H	UOPS_EXECUTED_PORT.PORT_7	Cycles which a Uop is dispatched on port 7 in this thread	Set AnyThread to count per core
A2H	01H	RESOURCE_STALLS.ANY	Cycles Allocation is stalled due to Resource Related reason.	
A2H	04H	RESOURCE_STALLS.RS	Cycles stalled due to no eligible RS entry available.	
A2H	08H	RESOURCE_STALLS.SB	Cycles stalled due to no store buffers available (not including draining form sync).	
A2H	10H	RESOURCE_STALLS.ROB	Cycles stalled due to re-order buffer full.	
A3H	02H	CYCLE_ACTIVITY.CYCLES_LDM_PENDING	Cycles with pending memory loads. Set Cmask=2 to count cycle.	
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_PENDING	Cycles with pending L1 cache miss loads. Set Cmask=8 to count cycle.	PMC2 only
AEH	01H	ITLB.ITLB_FLUSH	Counts the number of ITLB flushes, includes 4k/2M/4M pages.	
BOH	02H	OFFCORE_REQUESTS.DEMAND_CODE_RD	Demand code read requests sent to uncore.	
BOH	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ltoM.	
BOH	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	
B1H	02H	UOPS_EXECUTED.CORE	Counts total number of uops to be executed per-core each cycle.	Do not need to set ANY
B7H	01H	OFF_CORE_RESPONSE_0	see Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A7H
BCH	11H	PAGE_WALKER_LOADS.DTLB_L1	Number of DTLB page walker loads that hit in the L1+FB.	
BCH	21H	PAGE_WALKER_LOADS.ITLB_L1	Number of ITLB page walker loads that hit in the L1+FB.	

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
BCH	12H	PAGE_WALKER_LOADS.DTLB_L2	Number of DTLB page walker loads that hit in the L2.	
BCH	22H	PAGE_WALKER_LOADS.ITLB_L2	Number of ITLB page walker loads that hit in the L2.	
BCH	14H	PAGE_WALKER_LOADS.DTLB_L3	Number of DTLB page walker loads that hit in the L3.	
BCH	24H	PAGE_WALKER_LOADS.ITLB_L3	Number of ITLB page walker loads that hit in the L3.	
BCH	18H	PAGE_WALKER_LOADS.DTLB_MEMORY	Number of DTLB page walker loads from memory.	
BCH	28H	PAGE_WALKER_LOADS.ITLB_MEMORY	Number of ITLB page walker loads from memory.	
BDH	01H	TLB_FLUSH.DTLB_THREAD	DTLB flush attempts of the thread-specific entries.	
BDH	20H	TLB_FLUSH.STLB_ANY	Count number of STLB flush attempts.	
COH	00H	INST_RETIRED.ANY_P	Number of instructions at retirement.	See Table 19-1
COH	01H	INST_RETIRED.ALL	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution.	PMC1 only; Must quiesce other PMCs.
C1H	08H	OTHER_ASSISTS.AVX_TO_SSE	Number of transitions from AVX-256 to legacy SSE when penalty applicable.	
C1H	10H	OTHER_ASSISTS.SSE_TO_AVX	Number of transitions from SSE to AVX-256 when penalty applicable.	
C1H	40H	OTHER_ASSISTS.ANY_WB_ASSIST	Number of microcode assists invoked by HW upon uop writeback.	
C2H	01H	UOPS_RETIRED.ALL	Counts the number of micro-ops retired, Use cmask=1 and invert to count active cycles or stalled cycles.	Supports PEBS, use Any=1 for core granular.
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEARS.SMC	Number of self-modifying-code machine clears detected.	
C3H	20H	MACHINE_CLEARS.MASKMOV	Counts the number of executed AVX masked load operations that refer to an illegal address range with the mask bits set to 0.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions at retirement.	See Table 19-1
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	Supports PEBS
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Direct and indirect near call instructions retired.	

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	
C4H	10H	BR_INST_RETIRED.NOT_TAKEN	Counts the number of not taken branch instructions retired.	
C4H	20H	BR_INST_RETIRED.NEAR_TAKEN	Number of near taken branches retired.	
C4H	40H	BR_INST_RETIRED.FAR_BRANCH	Number of far branches retired.	
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement	See Table 19-1
C5H	01H	BR_MISP_RETIRED.CONDITIONAL	Mispredicted conditional branch instructions retired.	Supports PEBS
C5H	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	
CAH	02H	FP_ASSIST.X87_OUTPUT	Number of X87 FP assists due to Output values.	
CAH	04H	FP_ASSIST.X87_INPUT	Number of X87 FP assists due to input values.	
CAH	08H	FP_ASSIST.SIMD_OUTPUT	Number of SIMD FP assists due to Output values.	
CAH	10H	FP_ASSIST.SIMD_INPUT	Number of SIMD FP assists due to input values.	
CAH	1EH	FP_ASSIST.ANY	Cycles with any input/output SSE* or FP assists.	
CCH	20H	ROB_MISC_EVENTS.LBR_INSERTS	Count cases of saving new LBR records by hardware.	
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY	Sample loads with specified latency threshold. PMC3 only.	Specify threshold in MSR 0x3F6
DOH	01H	MEM_UOP_RETIRED.LOADS	Qualify retired memory uops that are loads. Combine with umask 10H, 20H, 40H, 80H.	Supports PEBS and DataLA
DOH	02H	MEM_UOP_RETIRED.STORES	Qualify retired memory uops that are stores. Combine with umask 10H, 20H, 40H, 80H.	Supports PEBS and DataLA
DOH	10H	MEM_UOP_RETIRED.STLB_MISSES	Qualify retired memory uops with STLB miss. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS and DataLA
DOH	20H	MEM_UOP_RETIRED.LOCK	Qualify retired memory uops with lock. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS and DataLA
DOH	40H	MEM_UOP_RETIRED.SPLIT	Qualify retired memory uops with line split. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS and DataLA
DOH	80H	MEM_UOP_RETIRED.ALL	Qualify any retired memory uops. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS and DataLA
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS and DataLA
D1H	02H	MEM_LOAD_UOPS_RETIRED.L2_HIT	Retired load uops with L2 cache hits as data sources.	Supports PEBS and DataLA

Table 19-2 Non-Architectural Performance Events In the Processor Core of Next Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D1H	04H	MEM_LOAD_UOPS_RETIRED.LLC_HIT	Retired load uops with LLC cache hits as data sources.	Supports PEBS and DataLA
D1H	10H	MEM_LOAD_UOPS_RETIRED.L2_MISS	Retired load uops missed L2. Unknown data source excluded.	Supports PEBS and DataLA
D1H	40H	MEM_LOAD_UOPS_RETIRED.HIT_LFB	Retired load uops which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.	
D2H	01H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS	Retired load uops which data sources were LLC hit and cross-core snoop missed in on-pkg core cache.	Supports PEBS and DataLA
D2H	02H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT	Retired load uops which data sources were LLC and cross-core snoop hits in on-pkg core cache.	Supports PEBS and DataLA
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Retired load uops which data sources were HitM responses from shared LLC.	Supports PEBS and DataLA
D2H	08H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE	Retired load uops which data sources were hits in LLC without snoops required.	Supports PEBS and DataLA
D3H	01H	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM	Retired load uops which data sources missed LLC but serviced from local dram.	Supports PEBS and DataLA.
E6H	1FH	BACLEARS.ANY	Number of front end re-steers due to BPU misprediction.	
F0H	01H	L2_TRANS.DEMAND_DATA_RD	Demand Data Read requests that access L2 cache.	
F0H	02H	L2_TRANS.RFO	RFO requests that access L2 cache.	
F0H	04H	L2_TRANS.CODE_RD	L2 cache accesses when fetching instructions.	
F0H	08H	L2_TRANS.ALL_PF	Any MLC or LLC HW prefetch accessing L2, including rejects.	
F0H	10H	L2_TRANS.L1D_WB	L1D writebacks that access L2 cache.	
F0H	20H	L2_TRANS.L2_FILL	L2 fill requests that access L2 cache.	
F0H	40H	L2_TRANS.L2_WB	L2 writebacks that access L2 cache.	
F0H	80H	L2_TRANS.ALL_REQUESTS	Transactions accessing L2 pipe.	
F1H	01H	L2_LINES_IN.I	L2 cache lines in I state filling L2.	Counting does not cover rejects.
F1H	02H	L2_LINES_IN.S	L2 cache lines in S state filling L2.	Counting does not cover rejects.
F1H	04H	L2_LINES_IN.E	L2 cache lines in E state filling L2.	Counting does not cover rejects.
F1H	07H	L2_LINES_IN.ALL	L2 cache lines filling L2.	Counting does not cover rejects.
F2H	05H	L2_LINES_OUT.DEMAND_CLEAN	Clean L2 cache lines evicted by demand.	
F2H	06H	L2_LINES_OUT.DEMAND_DIRTY	Dirty L2 cache lines evicted by demand.	

Table 19-3 Intel TSX Performance Events

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
54H	01H	TX_MEM.ABORT_CONFLICT	Number of times a transactional abort was signaled due to a data conflict on a transactionally accessed address	
	02H	TX_MEM.ABORT_CAPACITY	Number of times a transactional abort was signaled due to a data capacity limitation	
	04H	TX_MEM.ABORT_HLE_STORE_TO_ELIDED_LOCK	Number of times a HLE transactional region aborted due to a non XRELEASE prefixed instruction writing to an elided lock in the elision buffer	
	08H	TX_MEM.ABORT_HLE_ELISION_BUFFER_NOT_EMPTY	Number of times an HLE transactional execution aborted due to NoAllocatedElisionBuffer being non-zero.	
	10H	TX_MEM.ABORT_HLE_ELISION_BUFFER_MISMATCH	Number of times an HLE transactional execution aborted due to XRELEASE lock not satisfying the address and value requirements in the elision buffer.	
	20H	TX_MEM.ABORT_HLE_ELISION_BUFFER_UNSUPPORTED_ALIGNMENT	Number of times an HLE transactional execution aborted due to an unsupported read alignment from the elision buffer.	
	40H	TX_MEM.ABORT_HLE_ELISION_BUFFER_FULL	Number of times HLE lock could not be elided due to ElisionBufferAvailable being zero.	
5DH	01H	TX_EXEC.MISC1	Counts the number of times a class of instructions that may cause a transactional abort was executed. Since this is the count of execution, it may not always cause a transactional abort.	
	02H	TX_EXEC.MISC2	Counts the number of times a class of instructions that may cause a transactional abort was executed inside a transactional region	
	04H	TX_EXEC.MISC3	Counts the number of times an instruction execution caused the nest count supported to be exceeded	
	08H	TX_EXEC.MISC4	Counts the number of times an HLE XACQUIRE instruction was executed inside an RTM transactional region	

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C8H	01H	HLE_RETIREDD.START	Number of times an HLE execution started.	IF HLE is supported
	02H	HLE_RETIREDD.COMMIT	Number of times an HLE execution successfully committed	
	04H	HLE_RETIREDD.ABORTED	Number of times an HLE execution aborted due to any reasons (multiple categories may count as one)	
	08H	HLE_RETIREDD.ABORTED_MISC 1	Number of times an HLE execution aborted due to various memory events	
	10H	HLE_RETIREDD.ABORTED_MISC 2	Number of times an HLE execution aborted due to uncommon conditions	
	20H	HLE_RETIREDD.ABORTED_MISC 3	Number of times an HLE execution aborted due to HLE-unfriendly instructions	
	40H	HLE_RETIREDD.ABORTED_MISC 4	Number of times an HLE execution aborted due to incompatible memory type	
	80H	HLE_RETIREDD.ABORTED_MISC 5	Number of times an HLE execution aborted due to none of the previous 4 categories (e.g. interrupt)	
C9H	01H	RTM_RETIREDD.START	Number of times an RTM execution started.	IF RTM is supported
	02H	RTM_RETIREDD.COMMIT	Number of times an RTM execution successfully committed	
	04H	RTM_RETIREDD.ABORTED	Number of times an RTM execution aborted due to any reasons (multiple categories may count as one)	
	08H	RTM_RETIREDD.ABORTED_MISC 1	Number of times an RTM execution aborted due to various memory events	IF RTM is supported
	10H	RTM_RETIREDD.ABORTED_MISC 2	Number of times an RTM execution aborted due to uncommon conditions	
	20H	RTM_RETIREDD.ABORTED_MISC 3	Number of times an RTM execution aborted due to HLE-unfriendly instructions	
	40H	RTM_RETIREDD.ABORTED_MISC 4	Number of times an RTM execution aborted due to incompatible memory type	
	80H	RTM_RETIREDD.ABORTED_MISC 5	Number of times an RTM execution aborted due to none of the previous 4 categories (e.g. interrupt)	

Non-architectural performance monitoring events that are located in the uncore sub-system are implementation specific between different platforms using processors based on Intel microarchitecture Sandy Bridge. Processors with CPUID signature of DisplayFamily_DisplayModel 06_3CH and 06_45H support performance events listed in Table 19-4.

Table 19-4 Non-Architectural Uncore Performance Events In the Next Generation Intel® Core™ Processors

Event Num. ¹	Umask Value	Event Mask Mnemonic	Description	Comment
22H	01H	UNC_CBO_XSNP_RESPONSE.MISS	A snoop misses in some processor core.	Must combine with one of the umask values of 20H, 40H, 80H
22H	02H	UNC_CBO_XSNP_RESPONSE.INVAL	A snoop invalidates a non-modified line in some processor core.	
22H	04H	UNC_CBO_XSNP_RESPONSE.HIT	A snoop hits a non-modified line in some processor core.	
22H	08H	UNC_CBO_XSNP_RESPONSE.HITM	A snoop hits a modified line in some processor core.	
22H	10H	UNC_CBO_XSNP_RESPONSE.INVAL_M	A snoop invalidates a modified line in some processor core.	
22H	20H	UNC_CBO_XSNP_RESPONSE.EXTERNAL_FILTER	Filter on cross-core snoops initiated by this Cbox due to external snoop request.	Must combine with at least one of 01H, 02H, 04H, 08H, 10H
22H	40H	UNC_CBO_XSNP_RESPONSE.CORE_FILTER	Filter on cross-core snoops initiated by this Cbox due to processor core memory request.	
22H	80H	UNC_CBO_XSNP_RESPONSE.EVICTION_FILTER	Filter on cross-core snoops initiated by this Cbox due to LLC eviction.	
34H	01H	UNC_CBO_CACHE_LOOKUP.M	LLC lookup request that access cache and found line in M-state.	Must combine with one of the umask values of 10H, 20H, 40H, 80H
34H	02H	UNC_CBO_CACHE_LOOKUP.E	LLC lookup request that access cache and found line in E-state.	
34H	04H	UNC_CBO_CACHE_LOOKUP.S	LLC lookup request that access cache and found line in S-state.	
34H	08H	UNC_CBO_CACHE_LOOKUP.I	LLC lookup request that access cache and found line in I-state.	
34H	10H	UNC_CBO_CACHE_LOOKUP.READ_FILTER	Filter on processor core initiated cacheable read requests. Must combine with at least one of 01H, 02H, 04H, 08H.	
34H	20H	UNC_CBO_CACHE_LOOKUP.WRITE_FILTER	Filter on processor core initiated cacheable write requests. Must combine with at least one of 01H, 02H, 04H, 08H.	
34H	40H	UNC_CBO_CACHE_LOOKUP.EXTSNP_FILTER	Filter on external snoop requests. Must combine with at least one of 01H, 02H, 04H, 08H.	
34H	80H	UNC_CBO_CACHE_LOOKUP.ANY_REQUEST_FILTER	Filter on any IRQ or IPQ initiated requests including uncacheable, non-coherent requests. Must combine with at least one of 01H, 02H, 04H, 08H.	
80H	01H	UNC_ARB_TRK_OCCUPANCY.ALL	Counts cycles weighted by the number of requests waiting for data returning from the memory controller. Accounts for coherent and non-coherent requests initiated by IA cores, processor graphic units, or LLC.	Counter 0 only
81H	01H	UNC_ARB_TRK_REQUEST.ALL	Counts the number of coherent and in-coherent requests initiated by IA cores, processor graphic units, or LLC.	

Table 19-4 Non-Architectural Uncore Performance Events In the Next Generation Intel® Core™ Processors

Event Num. ¹	Umask Value	Event Mask Mnemonic	Description	Comment
81H	20H	UNC_ARB_TRK_REQUEST.WRITES	Counts the number of allocated write entries, include full, partial, and LLC evictions.	
81H	80H	UNC_ARB_TRK_REQUEST.EVICTIONS	Counts the number of LLC evictions allocated.	
83H	01H	UNC_ARB_COH_TRK_OCCUPANCY.ALL	Cycles weighted by number of requests pending in Coherency Tracker.	Counter 0 only
84H	01H	UNC_ARB_COH_TRK_REQUEST.ALL	Number of requests allocated in Coherency Tracker.	

NOTES:

1. The uncore events must be programmed using MSRs located in specific performance monitoring units in the uncore. UNC_CBO* events are supported using MSR_UNC_CBO* MSRs; UNC_ARB* events are supported using MSR_UNC_ARB*MSRs.

19.3 PERFORMANCE MONITORING EVENTS FOR 3RD GENERATION INTEL® CORE™ PROCESSORS

3rd Generation Intel® Core™ Processors are based on the Intel microarchitecture code name Ivy Bridge. They support architectural performance-monitoring events listed in Table 19-1. Non-architectural performance-monitoring events in the processor core are listed in Table 19-5. The events in Table 19-5 apply to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_3AH.

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	LD_BLOCKS.STORE_FORWARD	loads blocked by overlapping with store buffer that cannot be forwarded .	
05H	01H	MISALIGN_MEM_REF.LOADS	Speculative cache-line split load uops dispatched to L1D.	
05H	02H	MISALIGN_MEM_REF.STORES	Speculative cache-line split Store-address uops dispatched to L1D.	
07H	01H	LD_BLOCKS_PARTIAL.ADDRESS_ALIAS	False dependencies in MOB due to partial compare on address.	
08H	81H	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Misses in all TLB levels that cause a page walk of any page size from demand loads.	
08H	82H	DTLB_LOAD_MISSES.WALK_COMPLETED	Misses in all TLB levels that caused page walk completed of any size by demand loads.	
08H	84H	DTLB_LOAD_MISSES.WALK_DURATION	Cycle PMH is busy with a walk due to demand loads.	
0EH	01H	UOPS_ISSUED.ANY	Increments each cycle the # of Uops issued by the RAT to RS. Set Cmask = 1, Inv = 1, Any= 1 to count stalled cycles of this core.	Set Cmask = 1, Inv = 1 to count stalled cycles

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
0EH	10H	UOPS_ISSUED.FLAGS_MERGE	Number of flags-merge uops allocated. Such uops adds delay.	
0EH	20H	UOPS_ISSUED.SLOW_LEA	Number of slow LEA or similar uops allocated. Such uop has 3 sources (e.g. 2 sources + immediate) regardless if as a result of LEA instruction or not.	
0EH	40H	UOPS_ISSUED.SINGLE_MUL	Number of multiply packed/scalar single precision uops allocated.	
14H	01H	ARITH.FPU_DIV_ACTIVE	Cycles that the divider is active, includes INT and FP. Set 'edge =1, cmask=1' to count the number of divides.	
24H	01H	L2_RQSTS.DEMAND_DATA_RD_HIT	Demand Data Read requests that hit L2 cache	
24H	03H	L2_RQSTS.ALL_DEMAND_DATA_RD	Counts any demand and L1 HW prefetch data load requests to L2.	
24H	04H	L2_RQSTS.RFO_HITS	Counts the number of store RFO requests that hit the L2 cache.	
24H	08H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache.	
24H	0CH	L2_RQSTS.ALL_RFO	Counts all L2 store RFO requests.	
24H	10H	L2_RQSTS.CODE_RD_HIT	Number of instruction fetches that hit the L2 cache.	
24H	20H	L2_RQSTS.CODE_RD_MISS	Number of instruction fetches that missed the L2 cache.	
24H	30H	L2_RQSTS.ALL_CODE_RD	Counts all L2 code requests.	
24H	40H	L2_RQSTS.PF_HIT	Counts all L2 HW prefetcher requests that hit L2.	
24H	80H	L2_RQSTS.PF_MISS	Counts all L2 HW prefetcher requests that missed L2.	
24H	C0H	L2_RQSTS.ALL_PF	Counts all L2 HW prefetcher requests.	
27H	01H	L2_STORE_LOCK_RQSTS.MISS	RFOs that miss cache lines	
27H	08H	L2_STORE_LOCK_RQSTS.HIT_M	RFOs that hit cache lines in M state	
27H	0FH	L2_STORE_LOCK_RQSTS.ALL	RFOs that access cache lines in any state	
28H	01H	L2_L1D_WB_RQSTS.MISS	Not rejected writebacks that missed LLC.	
28H	04H	L2_L1D_WB_RQSTS.HIT_E	Not rejected writebacks from L1D to L2 cache lines in E state.	
28H	08H	L2_L1D_WB_RQSTS.HIT_M	Not rejected writebacks from L1D to L2 cache lines in M state.	
28H	0FH	L2_L1D_WB_RQSTS.ALL	Not rejected writebacks from L1D to L2 cache lines in any state.	
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache.	see Table 19-1

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
2EH	41H	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache.	see Table 19-1
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Counts the number of thread cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. The core frequency may change from time to time due to power or thermal throttling.	see Table 19-1
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK	Increments at the frequency of XCLK (100 MHz) when not halted.	see Table 19-1
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle. Set Cmaks = 1 and Edge = 1 to count occurrences.	PMC2 only; Set Cmask = 1 to count cycles.
49H	01H	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Miss in all TLB levels causes a page walk of any page size (4K/2M/4M/1G).	
49H	02H	DTLB_STORE_MISSES.WALK_COMPLETED	Miss in all TLB levels causes a page walk that completes of any page size (4K/2M/4M/1G).	
49H	04H	DTLB_STORE_MISSES.WALK_DURATION	Cycles PMH is busy with this walk.	
49H	10H	DTLB_STORE_MISSES.STLB_HIT	Store operations that miss the first TLB level but hit the second and do not cause page walks	
4CH	01H	LOAD_HIT_PRE.SW_PF	Non-SW-prefetch load dispatches that hit fill buffer allocated for S/W prefetch.	
4CH	02H	LOAD_HIT_PRE.HW_PF	Non-SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.	
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
58H	04H	MOVE_ELIMINATION.INT_NOT_ELIMINATED	Number of integer Move Elimination candidate uops that were not eliminated.	
58H	08H	MOVE_ELIMINATION.SIMD_NOT_ELIMINATED	Number of SIMD Move Elimination candidate uops that were not eliminated.	
58H	01H	MOVE_ELIMINATION.INT_ELIMINATED	Number of integer Move Elimination candidate uops that were eliminated.	
58H	02H	MOVE_ELIMINATION.SIMD_ELIMINATED	Number of SIMD Move Elimination candidate uops that were eliminated.	
5CH	01H	CPL_CYCLES.RING0	Unhalted core cycles when the thread is in ring 0.	Use Edge to count transition
5CH	02H	CPL_CYCLES.RING123	Unhalted core cycles when the thread is not in ring 0.	
5EH	01H	RS_EVENTS.EMPTY_CYCLES	Cycles the RS is empty for the thread.	
5FH	01H	DTLB_LOAD_MISSES.STLB_HIT	Counts load operations that missed 1st level DTLB but hit the 2nd level.	

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Offcore outstanding Demand Data Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD	Offcore outstanding Demand Code Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Offcore outstanding RFO store transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Offcore outstanding cacheable data read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
63H	01H	LOCK_CYCLES.SPLIT_LOCK_UC_LOCK_DURATION	Cycles in which the L1D and L2 are locked, due to a UC lock or split lock.	
63H	02H	LOCK_CYCLES.CACHE_LOCK_DURATION	Cycles in which the L1D is locked.	
79H	02H	IDQ.EMPTY	Counts cycles the IDQ is empty.	
79H	04H	IDQ.MITE_UOPS	Increment each cycle # of uops delivered to IDQ from MITE path. Set Cmask = 1 to count cycles.	Can combine Umask 04H and 20H
79H	08H	IDQ.DSB_UOPS	Increment each cycle. # of uops delivered to IDQ from DSB path. Set Cmask = 1 to count cycles.	Can combine Umask 08H and 10H
79H	10H	IDQ.MS_DSB_UOPS	Increment each cycle # of uops delivered to IDQ when MS_busy by DSB. Set Cmask = 1 to count cycles. Add Edge=1 to count # of delivery.	Can combine Umask 04H, 08H
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ when MS_busy by MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H, 08H
79H	30H	IDQ.MS_UOPS	Increment each cycle # of uops delivered to IDQ from MS by either DSB or MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H, 08H
79H	18H	IDQ.ALL_DSB_CYCLES_ANY_UOPS	Counts cycles DSB is delivered at least one uops. Set Cmask = 1.	
79H	18H	IDQ.ALL_DSB_CYCLES_4_UOPS	Counts cycles DSB is delivered four uops. Set Cmask = 4.	
79H	24H	IDQ.ALL_MITE_CYCLES_ANY_UOPS	Counts cycles MITE is delivered at least one uops. Set Cmask = 1.	
79H	24H	IDQ.ALL_MITE_CYCLES_4_UOPS	Counts cycles MITE is delivered four uops. Set Cmask = 4.	
79H	3CH	IDQ.MITE_ALL_UOPS	# of uops delivered to IDQ from any path.	
80H	02H	ICACHE.MISSES	Number of Instruction Cache, Streaming Buffer and Victim Cache Misses. Includes UC accesses.	

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
85H	01H	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses in all ITLB levels that cause page walks	
85H	02H	ITLB_MISSES.WALK_COMPLETED	Misses in all ITLB levels that cause completed page walks	
85H	04H	ITLB_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
85H	10H	ITLB_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.	
87H	01H	ILD_STALL.LCP	Stalls caused by changing prefix length of the instruction.	
87H	04H	ILD_STALL.IQ_FULL	Stall cycles due to IQ is full.	
88H	01H	BR_INST_EXEC.COND	Qualify conditional near branch instructions executed, but not necessarily retired.	Must combine with umask 40H, 80H
88H	02H	BR_INST_EXEC.DIRECT_JMP	Qualify all unconditional near branch instructions excluding calls and indirect branches.	Must combine with umask 80H
88H	04H	BR_INST_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify executed indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
88H	08H	BR_INST_EXEC.RETURN_NEAR	Qualify indirect near branches that have a return mnemonic.	Must combine with umask 80H
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Qualify unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Qualify indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
88H	40H	BR_INST_EXEC.NONTAKEN	Qualify non-taken near branches executed.	Applicable to umask 01H only
88H	80H	BR_INST_EXEC.TAKEN	Qualify taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
88H	FFH	BR_INST_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	
89H	01H	BR_MISP_EXEC.COND	Qualify conditional near branch instructions mispredicted.	Must combine with umask 40H, 80H
89H	04H	BR_MISP_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify mispredicted indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Qualify mispredicted indirect near branches that have a return mnemonic.	Must combine with umask 80H
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Qualify mispredicted unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Qualify mispredicted indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
89H	40H	BR_MISP_EXEC.NONTAKEN	Qualify mispredicted non-taken near branches executed.	Applicable to umask 01H only
89H	80H	BR_MISP_EXEC.TAKEN	Qualify mispredicted taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
89H	FFH	BR_MISP_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CORE	Count number of non-delivered uops to RAT per thread.	Use Cmask to qualify uop b/w
A1H	01H	UOPS_DISPATCHED_PORT.PORT_0	Cycles which a Uop is dispatched on port 0.	
A1H	02H	UOPS_DISPATCHED_PORT.PORT_1	Cycles which a Uop is dispatched on port 1.	
A1H	04H	UOPS_DISPATCHED_PORT.PORT_2_LD	Cycles which a load uop is dispatched on port 2.	
A1H	08H	UOPS_DISPATCHED_PORT.PORT_2_STA	Cycles which a store address uop is dispatched on port 2.	
A1H	0CH	UOPS_DISPATCHED_PORT.PORT_2	Cycles which a Uop is dispatched on port 2.	
A1H	10H	UOPS_DISPATCHED_PORT.PORT_3_LD	Cycles which a load uop is dispatched on port 3.	
A1H	20H	UOPS_DISPATCHED_PORT.PORT_3_STA	Cycles which a store address uop is dispatched on port 3.	
A1H	30H	UOPS_DISPATCHED_PORT.PORT_3	Cycles which a Uop is dispatched on port 3.	
A1H	40H	UOPS_DISPATCHED_PORT.PORT_4	Cycles which a Uop is dispatched on port 4.	
A1H	80H	UOPS_DISPATCHED_PORT.PORT_5	Cycles which a Uop is dispatched on port 5.	
A2H	01H	RESOURCE_STALLS.ANY	Cycles Allocation is stalled due to Resource Related reason.	
A2H	04H	RESOURCE_STALLS.RS	Cycles stalled due to no eligible RS entry available.	
A2H	08H	RESOURCE_STALLS.SB	Cycles stalled due to no store buffers available (not including draining form sync).	
A2H	10H	RESOURCE_STALLS.ROB	Cycles stalled due to re-order buffer full.	
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_PENDING	Cycles with pending L2 miss loads. Set AnyThread to count per core.	
A3H	02H	CYCLE_ACTIVITY.CYCLES_LDM_PENDING	Cycles with pending memory loads. Set AnyThread to count per core.	PMC0-3 only.
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_PENDING	Cycles with pending L1 cache miss loads. Set AnyThread to count per core.	PMC2 only
A3H	04H	CYCLE_ACTIVITY.CYCLES_NO_EXECUTE	Cycles of dispatch stalls. Set AnyThread to count per core.	
ABH	01H	DSB2MITE_SWITCHES.COUNT	Number of DSB to MITE switches.	
ABH	02H	DSB2MITE_SWITCHES.PENALTY_CYCLES	Cycles DSB to MITE switches caused delay.	
ACH	08H	DSB_FILL.EXCEED_DSB_LINES	DSB Fill encountered > 3 DSB lines.	

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
AEH	01H	ITLB.ITLB_FLUSH	Counts the number of ITLB flushes, includes 4k/2M/4M pages.	
BOH	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data read requests sent to uncore.	
BOH	02H	OFFCORE_REQUESTS.DEMAND_CODE_RD	Demand code read requests sent to uncore.	
BOH	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ItoM	
BOH	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	
B1H	01H	UOPS_EXECUTED.THREAD	Counts total number of uops to be executed per-thread each cycle. Set Cmask = 1, INV =1 to count stall cycles.	
B1H	02H	UOPS_EXECUTED.CORE	Counts total number of uops to be executed per-core each cycle.	Do not need to set ANY
B7H	01H	OFFCORE_RESPONSE_0	see Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H
BBH	01H	OFFCORE_RESPONSE_1	See Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A7H
BDH	01H	TLB_FLUSH.DTLB_THREAD	DTLB flush attempts of the thread-specific entries.	
BDH	20H	TLB_FLUSH.STLB_ANY	Count number of STLB flush attempts.	
COH	00H	INST_RETIRED.ANY_P	Number of instructions at retirement.	See Table 19-1
COH	01H	INST_RETIRED.ALL	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution.	PMC1 only
C1H	08H	OTHER_ASSISTS.AVX_STORE	Number of assists associated with 256-bit AVX store operations.	
C1H	10H	OTHER_ASSISTS.AVX_TO_SSE	Number of transitions from AVX-256 to legacy SSE when penalty applicable.	
C1H	20H	OTHER_ASSISTS.SSE_TO_AVX	Number of transitions from SSE to AVX-256 when penalty applicable.	
C2H	01H	UOPS_RETIRED.ALL	Counts the number of micro-ops retired, Use cmask=1 and invert to count active cycles or stalled cycles.	Supports PEBS, use Any=1 for core granular.
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	
C3H	02H	MACHINE_CLEAR.MEMORY_ORDERING	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEAR.SMC	Number of self-modifying-code machine clears detected.	
C3H	20H	MACHINE_CLEAR.MASKMOV	Counts the number of executed AVX masked load operations that refer to an illegal address range with the mask bits set to 0.	

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions at retirement.	See Table 19-1
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	Supports PEBS
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Direct and indirect near call instructions retired.	
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	
C4H	10H	BR_INST_RETIRED.NOT_TAKEN	Counts the number of not taken branch instructions retired.	
C4H	20H	BR_INST_RETIRED.NEAR_TAKEN	Number of near taken branches retired.	
C4H	40H	BR_INST_RETIRED.FAR_BRANCH	Number of far branches retired.	
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement.	See Table 19-1
C5H	01H	BR_MISP_RETIRED.CONDITIONAL	Mispredicted conditional branch instructions retired.	Supports PEBS
C5H	02H	BR_MISP_RETIRED.NEAR_CALL	Direct and indirect mispredicted near call instructions retired.	
C5H	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	
C5H	10H	BR_MISP_RETIRED.NOT_TAKEN	Mispredicted not taken branch instructions retired.	
C5H	20H	BR_MISP_RETIRED.TAKEN	Mispredicted taken branch instructions retired.	
CAH	02H	FP_ASSIST.X87_OUTPUT	Number of X87 FP assists due to Output values.	
CAH	04H	FP_ASSIST.X87_INPUT	Number of X87 FP assists due to input values.	
CAH	08H	FP_ASSIST.SIMD_OUTPUT	Number of SIMD FP assists due to Output values.	
CAH	10H	FP_ASSIST.SIMD_INPUT	Number of SIMD FP assists due to input values.	
CAH	1EH	FP_ASSIST.ANY	Cycles with any input/output SSE* or FP assists.	
CCH	20H	ROB_MISC_EVENTS.LBR_INSERTS	Count cases of saving new LBR records by hardware.	
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY	Sample loads with specified latency threshold. PMC3 only.	Specify threshold in MSR 0x3F6
CDH	02H	MEM_TRANS_RETIRED.PRECISE_STORE	Sample stores and collect precise store operation via PEBS record. PMC3 only.	See Section 18.8.4.3
DOH	01H	MEM_UOPS_RETIRED.LOADS	Qualify retired memory uops that are loads. Combine with umask 10H, 20H, 40H, 80H.	Supports PEBS

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D0H	02H	MEM_UOPS_RETIREDD.STORES	Qualify retired memory uops that are stores. Combine with umask 10H, 20H, 40H, 80H.	
D0H	10H	MEM_UOPS_RETIREDD.STLB_MISS	Qualify retired memory uops with STLB miss. Must combine with umask 01H, 02H, to produce counts.	
D0H	20H	MEM_UOPS_RETIREDD.LOCK	Qualify retired memory uops with lock. Must combine with umask 01H, 02H, to produce counts.	
D0H	40H	MEM_UOPS_RETIREDD.SPLIT	Qualify retired memory uops with line split. Must combine with umask 01H, 02H, to produce counts.	
D0H	80H	MEM_UOPS_RETIREDD.ALL	Qualify any retired memory uops. Must combine with umask 01H, 02H, to produce counts.	
D1H	01H	MEM_LOAD_UOPS_RETIREDD.L1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS
D1H	02H	MEM_LOAD_UOPS_RETIREDD.L2_HIT	Retired load uops with L2 cache hits as data sources.	
D1H	04H	MEM_LOAD_UOPS_RETIREDD.LLC_HIT	Retired load uops with LLC cache hits as data sources.	
D1H	20H	MEM_LOAD_UOPS_RETIREDD.LLC_MISS	Retired load uops which data sources were data missed LLC (excluding unknown data source).	
D1H	40H	MEM_LOAD_UOPS_RETIREDD.HIT_LFB	Retired load uops which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.	
D2H	01H	MEM_LOAD_UOPS_LLCHIT_RETIREDD.XSNP_MISS	Retired load uops which data sources were LLC hit and cross-core snoop missed in on-pkg core cache.	Supports PEBS
D2H	02H	MEM_LOAD_UOPS_LLCHIT_RETIREDD.XSNP_HIT	Retired load uops which data sources were LLC and cross-core snoop hits in on-pkg core cache.	Supports PEBS
D2H	04H	MEM_LOAD_UOPS_LLCHIT_RETIREDD.XSNP_HITM	Retired load uops which data sources were HitM responses from shared LLC.	
D2H	08H	MEM_LOAD_UOPS_LLCHIT_RETIREDD.XSNP_NONE	Retired load uops which data sources were hits in LLC without snoops required.	
D3H	01H	MEM_LOAD_UOPS_LLCHIT_RETIREDD.LOCAL_DRAM	Retired load uops which data sources missed LLC but serviced from local dram.	Supports PEBS.
E6H	1FH	BACLEARSS.ANY	Number of front end re-steers due to BPU misprediction.	
FOH	01H	L2_TRANS.DEMAND_DATA_READ	Demand Data Read requests that access L2 cache.	
FOH	02H	L2_TRANS.RFO	RFO requests that access L2 cache.	
FOH	04H	L2_TRANS.CODE_RD	L2 cache accesses when fetching instructions.	
FOH	08H	L2_TRANS.ALL_PF	Any MLC or LLC HW prefetch accessing L2, including rejects.	
FOH	10H	L2_TRANS.L1D_WB	L1D writebacks that access L2 cache.	
FOH	20H	L2_TRANS.L2_FILL	L2 fill requests that access L2 cache.	

Table 19-5 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
F0H	40H	L2_TRANS.L2_WB	L2 writebacks that access L2 cache.	
F0H	80H	L2_TRANS.ALL_REQUESTS	Transactions accessing L2 pipe.	
F1H	01H	L2_LINES_IN.I	L2 cache lines in I state filling L2.	Counting does not cover rejects.
F1H	02H	L2_LINES_IN.S	L2 cache lines in S state filling L2.	Counting does not cover rejects.
F1H	04H	L2_LINES_IN.E	L2 cache lines in E state filling L2.	Counting does not cover rejects.
F1H	07H	L2_LINES_IN.ALL	L2 cache lines filling L2.	Counting does not cover rejects.
F2H	01H	L2_LINES_OUT.DEMAND_CLEAN	Clean L2 cache lines evicted by demand.	
F2H	02H	L2_LINES_OUT.DEMAND_DIRTY	Dirty L2 cache lines evicted by demand.	
F2H	04H	L2_LINES_OUT.PF_CLEAN	Clean L2 cache lines evicted by the MLC prefetcher.	
F2H	08H	L2_LINES_OUT.PF_DIRTY	Dirty L2 cache lines evicted by the MLC prefetcher.	
F2H	0AH	L2_LINES_OUT.DIRTY_ALL	Dirty L2 cache lines filling the L2.	Counting does not cover rejects.

...

Table 19-6 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E5 Family (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
...				
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle. Set Cmaks = 1 and Edge =1 to count occurrences.	PMC2 only; Set Cmask = 1 to count cycles.
...				
5BH	0CH	RESOURCE_STALLS2.ALL_FL_EMPTY	Cycles stalled due to free list empty.	PMC0-3 only regardless HTT
...				
A2H	01H	RESOURCE_STALLS.ANY	Cycles Allocation is stalled due to Resource Related reason.	
...				
A3H	02H	CYCLE_ACTIVITY.CYCLES_L1D_PENDING	Cycles with pending L1 cache miss loads.Set AnyThread to count per core.	PMC2 only
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_PENDING	Cycles with pending L2 miss loads. Set AnyThread to count per core.	

A3H	04H	CYCLE_ACTIVITY.CYCLES_NO_DISPATCH	Cycles of dispatch stalls. Set AnyThread to count per core.	PMCO-3 only
...				
B1H	01H	UOPS_DISPATCHED.THREAD	Counts total number of uops to be dispatched per-thread each cycle. Set Cmask = 1, INV =1 to count stall cycles.	PMCO-3 only regardless HTT
...				
B7H	01H	OFF_CORE_RESPONSE_0	see Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A7H
...				
D0H	01H	MEM_UOP_RETIRED.LOADS	Qualify retired memory uops that are loads. Combine with umask 10H, 20H, 40H, 80H.	Supports PEBS. PMCO-3 only regardless HTT.
...				
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS. PMCO-3 only regardless HTT
...				
D4H	02H	MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS	Retired load uops with unknown information as data source in cache serviced the load.	Supports PEBS. PMCO-3 only regardless HTT
...				

...

Table 19-7 Non-Architectural Performance Events applicable only to the Processor core for 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D2H	01H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS	Retired load uops which data sources were LLC hit and cross-core snoop missed in on-pkg core cache.	Supports PEBS. PMCO-3 only regardless HTT
D2H	02H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT	Retired load uops which data sources were LLC and cross-core snoop hits in on-pkg core cache.	
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Retired load uops which data sources were HitM responses from shared LLC.	
D2H	08H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE	Retired load uops which data sources were hits in LLC without snoops required.	
...				

...

19. Updates to Chapter 24, Volume 3C

Change bars show changes to Chapter 24 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

...

24.4.1 Guest Register State

The following fields in the guest-state area correspond to processor registers:

- Control registers CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Debug register DR7 (64 bits; 32 bits on processors that do not support Intel 64 architecture).
- RSP, RIP, and RFLAGS (64 bits each; 32 bits on processors that do not support Intel 64 architecture).¹
- The following fields for each of the registers CS, SS, DS, ES, FS, GS, LDTR, and TR:
 - Selector (16 bits).
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture). The base-address fields for CS, SS, DS, and ES have only 32 architecturally-defined bits; nevertheless, the corresponding VMCS fields have 64 bits on processors that support Intel 64 architecture.
 - Segment limit (32 bits). The limit field is always a measure in bytes.
 - Access rights (32 bits). The format of this field is given in Table 24-2 and detailed as follows:
 - The low 16 bits correspond to bits 23:8 of the upper 32 bits of a 64-bit segment descriptor. While bits 19:16 of code-segment and data-segment descriptors correspond to the upper 4 bits of the segment limit, the corresponding bits (bits 11:8) are reserved in this VMCS field.
 - Bit 16 indicates an **unusable segment**. Attempts to use such a segment fault except in 64-bit mode. In general, a segment register is unusable if it has been loaded with a null selector.²
 - Bits 31:17 are reserved.

Table 24-2 Format of Access Rights

Bit Position(s)	Field
3:0	Segment type
4	S — Descriptor type (0 = system; 1 = code or data)
6:5	DPL — Descriptor privilege level
7	P — Segment present
11:8	Reserved
12	AVL — Available for use by system software

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.
2. There are a few exceptions to this statement. For example, a segment with a non-null selector may be unusable following a task switch that fails after its commit point; see "Interrupt 10—Invalid TSS Exception (#TS)" in Section 6.14, "Exception and Interrupt Handling in 64-bit Mode," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. In contrast, the TR register is usable after processor reset despite having a null selector; see Table 10-1 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Table 24-2 Format of Access Rights (Contd.)

Bit Position(s)	Field
13	Reserved (except for CS) L – 64-bit mode active (for CS only)
14	D/B – Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
15	G – Granularity
16	Segment unusable (0 = usable; 1 = unusable)
31:17	Reserved

The base address, segment limit, and access rights compose the “hidden” part (or “descriptor cache”) of each segment register. These data are included in the VMCS because it is possible for a segment register’s descriptor cache to be inconsistent with the segment descriptor in memory (in the GDT or the LDT) referenced by the segment register’s selector.

The value of the DPL field for SS is always equal to the logical processor’s current privilege level (CPL).¹

- 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.
- The register SMBASE (32 bits). This register contains the base address of the logical processor’s SMRAM image.

24.4.2 Guest Non-Register State

In addition to the register state described in Section 24.4.1, the guest-state area includes the following fields that characterize guest state but which do not correspond to processor registers:

- **Activity state** (32 bits). This field identifies the logical processor’s activity state. When a logical processor is executing instructions normally, it is in the **active state**. Execution of certain instructions and the occurrence of certain events may cause a logical processor to transition to an **inactive state** in which it ceases to execute instructions.

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.

The following activity states are defined:¹

- 0: **Active**. The logical processor is executing instructions normally.
- 1: **HLT**. The logical processor is inactive because it executed the HLT instruction.
- 2: **Shutdown**. The logical processor is inactive because it incurred a **triple fault**² or some other serious error.
- 3: **Wait-for-SIPI**. The logical processor is inactive because it is waiting for a startup-IPI (SIPI).

Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.

- **Interruptibility state** (32 bits). The IA-32 architecture includes features that permit certain events to be blocked for a period of time. This field contains information about such blocking. Details and the format of this field are given in Table 24-3.

Table 24-3 Format of Interruptibility State

Bit Position(s)	Bit Name	Notes
0	Blocking by STI	See the “STI—Set Interrupt Flag” section in Chapter 4 of the <i>Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B</i> . Execution of STI with RFLAGS.IF = 0 blocks interrupts (and, optionally, other events) for one instruction after its execution. Setting this bit indicates that this blocking is in effect.
1	Blocking by MOV SS	See the “MOV—Move a Value from the Stack” and “POP—Pop a Value from the Stack” sections in Chapter 4 of the <i>Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B</i> , and Section 6.8.3 in the <i>Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A</i> . Execution of a MOV to SS or a POP to SS blocks interrupts for one instruction after its execution. In addition, certain debug exceptions are inhibited between a MOV to SS or a POP to SS and a subsequent instruction. Setting this bit indicates that the blocking of all these events is in effect. 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 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 6.7.1 in the <i>Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A</i> and Section 34.8. Delivery of a non-maskable interrupt (NMI) or a system-management interrupt (SMI) blocks subsequent NMIs until the next execution of IRET. See Section 25.3 for how this behavior of IRET may change in VMX non-root operation. Setting this bit indicates that blocking of NMIs is in effect. Clearing this bit does not imply that NMIs are not (temporarily) blocked for other reasons. If the “virtual NMIs” VM-execution control (see Section 24.6.1) is 1, this bit does not control the blocking of NMIs. Instead, it refers to “virtual-NMI blocking” (the fact that guest software is not ready for an NMI).
31:4	Reserved	VM entry will fail if these bits are not 0. See Section 26.3.1.5.

1. Execution of the MWAIT instruction may put a logical processor into an inactive state. However, this VMCS field never reflects this state. See Section 27.1.
2. A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

- **Pending debug exceptions** (64 bits; 32 bits on processors that do not support Intel 64 architecture). IA-32 processors may recognize one or more debug exceptions without immediately delivering them.¹ This field contains information about such exceptions. This field is described in Table 24-4.

Table 24-4 Format of Pending-Debug-Exceptions

Bit Position(s)	Bit Name	Notes
3:0	B3 - B0	When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if the corresponding enabling bit in DR7 is not set.
11:4	Reserved	VM entry fails if these bits are not 0. See Section 26.3.1.5.
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7.
13	Reserved	VM entry fails if this bit is not 0. See Section 26.3.1.5.
14	BS	When set, this bit indicates that a debug exception would have been triggered by single-step execution mode.
63:15	Reserved	VM entry fails if these bits are not 0. See Section 26.3.1.5. Bits 63:32 exist only on processors that support Intel 64 architecture.

- **VMCS link pointer** (64 bits). This field is included for future expansion. Software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 26.3.1.5).
- **VMX-preemption timer value** (32 bits). This field is supported only on processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control. This field contains the value that the VMX-preemption timer will use following the next VM entry with that setting. See Section 25.5.1 and Section 26.6.4.
- **Page-directory-pointer-table entries** (PDPTs; 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 PDPTs referenced by CR3 when PAE paging is in use (see Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). They are used only if the “enable EPT” VM-execution control is 1.
- **Guest interrupt status** (16 bits). This field is supported only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control. It characterizes part of the guest’s virtual-APIC state and does not correspond to any processor or APIC registers. It comprises two 8-bit subfields:
 - **Requesting virtual interrupt (RVI)**. This is the low byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is requesting service. (The value 0 implies that there is no such interrupt.)
 - **Servicing virtual interrupt (SVI)**. This is the high byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is in service. (The value 0 implies that there is no such interrupt.)

See Chapter 29 for more information on the use of this field.

1. For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 6.8.3 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. In addition, certain events incident to an instruction (for example, an INIT signal) may take priority over debug traps generated by that instruction. See Table 6-2 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

24.5 HOST-STATE AREA

This section describes fields contained in the host-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM exit (see Section 27.5).

All fields in the host-state area correspond to processor registers:

- CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- RSP and RIP (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Selector fields (16 bits each) for the segment registers CS, SS, DS, ES, FS, GS, and TR. There is no field in the host-state area for the LDTR selector.
- Base-address fields for FS, GS, TR, GDTR, and IDTR (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- The following MSRs:
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
 - IA32_PAT (64 bits). This field is supported only on processors that support either the 1-setting of the “load 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-exit control.

In addition to the state identified here, some processor state components are loaded with fixed values on every VM exit; there are no fields corresponding to these components in the host-state area. See Section 27.5 for details of how state is loaded on VM exits.

...

24.6.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts).¹ Table 24-5 lists the controls. See Chapter 25 for how these controls affect processor behavior in VMX non-root operation.

Table 24-5 Definitions of Pin-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	External-interrupt exiting	If this control is 1, external interrupts cause VM exits. Otherwise, they are delivered normally through the guest interrupt-descriptor table (IDT). If this control is 1, the value of RFLAGS.IF does not affect interrupt blocking.
3	NMI exiting	If this control is 1, non-maskable interrupts (NMIs) cause VM exits. Otherwise, they are delivered normally using descriptor 2 of the IDT. This control also determines interactions between IRET and blocking by NMI (see Section 25.3).

1. Some asynchronous events cause VM exits regardless of the settings of the pin-based VM-execution controls (see Section 25.2).

Table 24-5 Definitions of Pin-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
5	Virtual NMIs	If this control is 1, NMIs are never blocked and the “blocking by NMI” bit (bit 3) in the interruptibility-state field indicates “virtual-NMI blocking” (see Table 24-3). This control also interacts with the “NMI-window exiting” VM-execution control (see Section 24.6.2).
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 25.5.1. A VM exit occurs when the timer counts down to zero; see Section 25.2.
7	Process posted interrupts	If this control is 1, the processor treats interrupts with the posted-interrupt notification vector (see Section 24.6.8) specially, updating the virtual-APIC page with posted-interrupt requests (see Section 29.6).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PINBASED_CTLs and IA32_VMX_TRUE_PINBASED_CTLs (see Appendix A.3.1) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 26.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 2, and 4. The VMX capability MSR IA32_VMX_PINBASED_CTLs will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PINBASED_CTLs MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

24.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls constitute two 32-bit vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions.¹ These are the **primary processor-based VM-execution controls** and the **secondary processor-based VM-execution controls**.

Table 24-6 lists the primary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-6 Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there are no other blocking of interrupts (see Section 24.4.2).
3	Use TSC offsetting	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC offset field (see Section 24.6.5 and Section 25.3).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC cause VM exits.
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP cause VM exits.

1. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 25.1.2), as do task switches (see Section 25.2).

Table 24-6 Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
15	CR3-load exiting	In conjunction with the CR3-target controls (see Section 24.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 25.1.3. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
16	CR3-store exiting	This control determines whether executions of MOV from CR3 cause VM exits. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits.
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits.
21	Use TPR shadow	Setting this control to 1 enables TPR virtualization and other APIC-virtualization features. See Chapter 29.
22	NMI-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if there is no virtual-NMI blocking (see Section 24.4.2).
23	MOV-DR exiting	This control determines whether executions of MOV DR cause VM exits.
24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits.
25	Use I/O bitmaps	This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 24.6.4 and Section 25.1.3). For this control, “0” means “do not use I/O bitmaps” and “1” means “use I/O bitmaps.” If the I/O bitmaps are used, the setting of the “unconditional I/O exiting” control is ignored.
27	Monitor trap flag	If this control is 1, the monitor trap flag debugging feature is enabled. See Section 25.5.2.
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 24.6.9 and Section 25.1.3). For this control, “0” means “do not use MSR bitmaps” and “1” means “use MSR bitmaps.” If the MSR bitmaps are not used, all executions of the RDMSR and WRMSR instructions cause VM exits.
29	MONITOR exiting	This control determines whether executions of MONITOR cause VM exits.
30	PAUSE exiting	This control determines whether executions of PAUSE cause VM exits.
31	Activate secondary controls	This control determines whether the secondary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the secondary processor-based VM-execution controls were also 0.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PROCBASED_CTLs and IA32_VMX_TRUE_PROCBASED_CTLs (see Appendix A.3.2) to determine how

to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 26.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 4–6, 8, 13–16, and 26. The VMX capability MSR IA32_VMX_PROCBASED_CTLs will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PROCBASED_CTLs MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-execution controls determines whether the secondary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the

secondary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 31 of the primary processor-based VM-execution controls do not support the secondary processor-based VM-execution controls.

Table 24-7 lists the secondary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-7 Definitions of Secondary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	Virtualize APIC accesses	If this control is 1, the logical processor treats specially accesses to the page with the APIC-access address. See Section 29.4.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Section 28.2.
2	Descriptor-table exiting	This control determines whether executions of LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, and STR cause VM exits.
3	Enable RDTSCP	If this control is 0, any execution of RDTSCP causes an invalid-opcode exception (#UD).
4	Virtualize x2APIC mode	If this control is 1, the logical processor treats specially RDMSR and WRMSR to APIC MSRs (in the range 800H-8FFH). See Section 29.5.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 28.1.
6	WBINVD exiting	This control determines whether executions of WBINVD cause VM exits.
7	Unrestricted guest	This control determines whether guest software may run in unpagged protected mode or in real-address mode.
8	APIC-register virtualization	If this control is 1, the logical processor virtualizes certain APIC accesses. See Section 29.4 and Section 29.5.
9	Virtual-interrupt delivery	This control enables the evaluation and delivery of pending virtual interrupts as well as the emulation of writes to the APIC registers that control interrupt prioritization.
10	PAUSE-loop exiting	This control determines whether a series of executions of PAUSE can cause a VM exit (see Section 24.6.13 and Section 25.1.3).
11	RDRAND exiting	This control determines whether executions of RDRAND cause VM exits.
12	Enable INVPCID	If this control is 0, any execution of INVPCID causes an invalid-opcode exception (#UD).
13	Enable VM functions	Setting this control to 1 enables use of the VMFUNC instruction in VMX non-root operation. See Section 25.5.5.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTL2 (see Appendix A.3.3) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 26.2.1.1).

...

24.6.8 Controls for APIC Virtualization

There are three mechanisms by which software accesses registers of the logical processor's local APIC:

- If the local APIC is in xAPIC mode, it can perform memory-mapped accesses to addresses in the 4-KByte page referenced by the physical address in the IA32_APIC_BASE MSR (see Section 10.4.4, "Local APIC Status and Location" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A and Intel® 64 Architecture Processor Topology Enumeration*).¹

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

- If the local APIC is in x2APIC mode, it can access the local APIC's registers using the RDMSR and WRMSR instructions (see *Intel® 64 Architecture Processor Topology Enumeration*).
- In 64-bit mode, it can access the local APIC's task-priority register (TPR) using the MOV CR8 instruction.

There are five processor-based VM-execution controls (see Section 24.6.2) that control such accesses. There are “use TPR shadow”, “virtualize APIC accesses”, “virtualize x2APIC mode”, “virtual-interrupt delivery”, and “APIC-register virtualization”. These controls interact with the following fields:

- **APIC-access address** (64 bits). This field contains the physical address of the 4-KByte **APIC-access page**. If the “virtualize APIC accesses” VM-execution control is 1, access to this page may cause VM exits or be virtualized by the processor. See Section 29.4.

The APIC-access address exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.

- **Virtual-APIC address** (64 bits). This field contains the physical address of the 4-KByte **virtual-APIC page**. The processor uses the virtual-APIC page to virtualize certain accesses to APIC registers and to manage virtual interrupts; see Chapter 29.

Depending on the setting of the controls indicated earlier, the virtual-APIC page may be accessed by the following operations:

- The MOV CR8 instructions (see Section 29.3).
- Accesses to the APIC-access page if, in addition, the “virtualize APIC accesses” VM-execution control is 1 (see Section 29.4).
- The RDMSR and WRMSR instructions if, in addition, the value of ECX is in the range 800H–8FFH (indicating an APIC MSR) and the “virtualize x2APIC mode” VM-execution control is 1 (see Section 29.5).

If the “use TPR shadow” VM-execution control is 1, VM entry ensures that the virtual-APIC address is 4-KByte aligned. The virtual-APIC address exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

- **TPR threshold** (32 bits). Bits 3:0 of this field determine the threshold below which bits 7:4 of VTPR (see Section 29.1.1) cannot fall. If the “virtual-interrupt delivery” VM-execution control is 0, a VM exit occurs after an operation (e.g., an execution of MOV to CR8) that reduces the value of those bits below the TPR threshold. See Section 29.1.2.

The TPR threshold exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

- **EOI-exit bitmap** (4 fields; 64 bits each). These fields are supported only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control. They are used to determine which virtualized writes to the APIC's EOI register cause VM exits:

- EOI_EXIT0 contains bits for vectors from 0 (bit 0) to 63 (bit 63).
- EOI_EXIT1 contains bits for vectors from 64 (bit 0) to 127 (bit 63).
- EOI_EXIT2 contains bits for vectors from 128 (bit 0) to 191 (bit 63).
- EOI_EXIT3 contains bits for vectors from 192 (bit 0) to 255 (bit 63).

See Section 29.1.4 for more information on the use of this field.

- **Posted-interrupt notification vector** (16 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. Its low 8 bits contain the interrupt vector that is used to notify a logical processor that virtual interrupts have been posted. See Section 29.6 for more information on the use of this field.
- **Posted-interrupt descriptor address** (64 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. It is the physical address of a 64-byte aligned posted interrupt descriptor. See Section 29.6 for more information on the use of this field.

...

20. Updates to Chapter 25, Volume 3C

Change bars show changes to Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

...

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 25.1, "Instructions That Cause VM Exits"
- Section 25.2, "Other Causes of VM Exits"
- Section 25.3, "Changes to Instruction Behavior in VMX Non-Root Operation"
- Section 25.4, "Other Changes in VMX Non-Root Operation"
- Section 25.5, "Features Specific to VMX Non-Root Operation"

Chapter 24, "Virtual-Machine Control Structures," describes the data control structures that govern VMX non-root operation. Chapter 26, "VM Entries," describes the operation of VM entries by which the processor transitions from VMX root operation to VMX non-root operation. Chapter 27, "VM Exits," describes the operation of VM exits by which the processor transitions from VMX non-root operation to VMX root operation.

Chapter 28, "VMX Support for Address Translation," describes two features that support address translation in VMX non-root operation. Chapter 29, "APIC Virtualization and Virtual Interrupts," describes features that support virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC) in VMX non-root operation.

...

25.1.2 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, which include: INVEPT, INVVPID, VMCALL,² VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.

25.1.3 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:

- **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.
- **HLT.** The HLT instruction causes a VM exit if the "HLT exiting" 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.

- **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 24.6.4). 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).

See Section 25.1.1 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 1.¹
- **LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR.** These instructions cause VM exits if the “descriptor-table exiting” VM-execution control is 1.²
- **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/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/mask and the values of the corresponding bits in the source operand and the CR0 read shadow differ.
- **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. If the CR3-target count in 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.

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.

-
1. “Enable INVPCID” 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 INVPCID” VM-execution control were 0. See Section 24.6.2.
 2. “Descriptor-table exiting” 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 “descriptor-table exiting” VM-execution control were 0. See Section 24.6.2.

- **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 25.1.1 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 25.3).
- **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:¹

— 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.

- **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 range 00000000H – 00001FFFH or 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 24.6.9 for details regarding how these bitmaps are identified.

- **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 1.²

1. “PAUSE-loop exiting” 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 “PAUSE-loop exiting” VM-execution control were 0. See Section 24.6.2.

- **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 1.¹
- **RSM.** The RSM instruction causes a VM exit if executed in system-management mode (SMM).²
- **WBINVD.** The WBINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.³
- **WRMSR.** The WRMSR 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 range 00000000H – 00001FFFH or 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 24.6.9 for details regarding how these bitmaps are identified.

25.2 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 24.6.3). 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 through the guest IDT. This use of the exception bitmap applies also to exceptions generated by the instructions INT3, INTO, BOUND, and UD2.

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.

2. “RDRAND exiting” 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 “RDRAND exiting” VM-execution control were 0. See Section 24.6.2.

1. “Enable RDTSCP” 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 RDTSCP” VM-execution control were 0. See Section 24.6.2.

2. 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.

3. “WBINVD exiting” 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 “WBINVD exiting” VM-execution control were 0. See Section 24.6.2.

- **External interrupts.** An external interrupt causes a VM exit if the “external-interrupt exiting” VM-execution control is 1. Otherwise, the interrupt is delivered normally through the IDT. (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The interrupt is not delivered through the IDT 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 descriptor 2 of the IDT. (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered through the IDT 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 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 25.4.2.
- **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 25.5.1 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.

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 24-3). Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 26.6.5).

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 (see Table 24-3). (A logical processor may also prevent such a VM exit if there is blocking of events by STI.) Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 26.6.6).

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.

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.

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.

25.3 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:

- **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:
 - 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 23.8), 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.
- **INVPCID.** Behavior of the INVPCID instruction is determined first by the setting of the “enable INVPCID” VM-execution control:¹
 - If the “enable INVPCID” VM-execution control is 0, INVPCID causes an invalid-opcode exception (#UD).
 - 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 24-3) 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 26.2.1.1.)
 - 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 25.1.3) 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 23.8) 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.

1. “Enable INVPCID” 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 INVPCID” VM-execution control were 0. See Section 24.6.2.

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 25.1.3), the value loaded from CR3 is a guest-physical address; see Section 28.2.1.
- **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 25.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 29.3.
- **MOV to CR0.** An execution of MOV to CR0 that does not cause a VM exit (see Section 25.1.3) 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:¹
 - 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 23.8).
 - 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 25.1.3), the value loaded into CR3 is treated as a guest-physical address; see Section 28.2.1.
 - 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.²
 - 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 through EPT.
- **MOV to CR4.** An execution of MOV to CR4 that does not cause a VM exit (see Section 25.1.3) 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 23.8).
- **MOV to CR8.** If the MOV to CR8 instruction does not cause a VM exit (see Section 25.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 29.3.
- **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.

1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “unrestricted guest” VM-execution control were 0. See Section 24.6.2.

2. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

- If the “MWAIT exiting” VM-execution control is 0, MWAIT does not cause the processor to enter an implementation-dependent optimized state if (1) ECX[0] = 1; and (2) either (a) the “interrupt-window exiting” VM-execution control is 0; or (b) the logical processor has recognized a pending virtual interrupt (see Section 29.2.1). Instead, control passes to the instruction following the MWAIT instruction.
- **RDMSR.** Section 25.1.3 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 as well as the TSC offset:
 - If the control is 0, the instruction operates normally, loading EAX:EDX with the value of the IA32_TIME_STAMP_COUNTER MSR.
 - If the control is 1, the instruction loads EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset (interpreted as a signed value).

The 1-setting of the “use TSC-offsetting” VM-execution control does not effect 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 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 29.5.¹
- **RDTSC.** Behavior of the RDTSC instruction is determined by the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls as well as the TSC offset:
 - If both controls are 0, RDTSC operates normally.
 - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, RDTSC loads EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset (interpreted as a signed value).
 - If the “RDTSC exiting” VM-execution control is 1, RDTSC causes a VM exit.
- **RDTSCP.** Behavior of the RDTSCP instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:²
 - If the “enable RDTSCP” VM-execution control is 0, RDTSCP causes an invalid-opcode exception (#UD).
 - If the “enable RDTSCP” VM-execution control is 1, treatment is based on the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls as well as the TSC offset:
 - If both controls are 0, RDTSCP operates normally.
 - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, RDTSCP loads EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset (interpreted as a signed value); it also loads ECX with the value of bits 31:0 of the IA32_TSC_AUX MSR.
 - If the “RDTSC 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

1. “Virtualize x2APIC mode” 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 “virtualize x2APIC mode” VM-execution control were 0. See Section 24.6.2.

2. “Enable RDTSCP” 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 RDTSCP” VM-execution control were 0. See Section 24.6.2.

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.

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.

- **WRMSR.** Section 25.1.3 identifies when executions of the WRMSR instruction cause VM exits. If such an execution 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 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.
 - If ECX contains 808H (indicating the TPR MSR), 80BH (the EOI MSR), or 83FH (self-IPI MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 29.5.¹

25.4 OTHER CHANGES IN VMX NON-ROOT OPERATION

Treatments of event blocking and of task switches differ in VMX non-root operation as described in the following sections.

...

25.4.2 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*, 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.

1. “Virtualize x2APIC mode” 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 “virtualize x2APIC mode” VM-execution control were 0. See Section 24.6.2.

- 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 IDT-vectoring information fields and information about the exception that caused the VM exit is recorded in the VM-exit interruption-information fields. See Section 27.2. 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 IDT-vectoring fields of the VMCS. Since the cause of such a VM exit is a task switch and not an interruption, the valid bit for the VM-exit interruption information field is 0. See Section 27.2.

...

25.5.4 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 29.

...

25.5.5.3 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 28.2 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 24.6.14; because this structure contains 512 8-Byte entries, VMFUNC causes a VM exit if $ECX \geq 512$). If the selected entry is a valid EPTP value (it would not cause VM entry to fail; see Section 26.2.1.1), it 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)
      THEN VMexit;
    ELSE
      write tent_EPTP to the EPTP field in the current VMCS;
      start using tent_EPTP as the new EPTP value for address translation;
    FI;
FI;

```

Execution of the EPTP-switching VM function does not modify the state of any registers; no flags are modified.

As noted in Section 25.5.5.2, 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).¹ If the “enable VPID” VM-execution control is 0, an EPTP-switching VMFUNC invalidates combined mappings associated with VPID 0000H (for all PCIDs and for all EP4TA values, where EP4TA 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 IA-32e paging 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.

...

21. Updates to Chapter 26, Volume 3C

Change bars show changes to Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C: System Programming Guide, Part 3*.

1. If the “enable VPID” VM-execution control is 0, the current VPID is 0000H; if CR4.PCIDE = 0, the current PCID is 000H.

...

26.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:¹

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSR 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 MSR 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 MSR 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.

- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC to determine the number of values supported (see Appendix A.6).
- If the “use I/O bitmaps” VM-execution control is 1, bits 11:0 of each I/O-bitmap address must be 0. Neither address should set any bits beyond the processor’s physical-address width.^{2,3}
- If the “use MSR bitmaps” VM-execution control is 1, bits 11:0 of the MSR-bitmap address must be 0. The address should not set any bits beyond the processor’s physical-address width.⁴
- If the “use TPR shadow” VM-execution control is 1, the virtual-APIC address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address should not set any bits beyond the processor’s physical-address width.⁵

If all of the above checks are satisfied and the “use TPR shadow” VM-execution control is 1, bytes 3:1 of VTPR (see Section 29.1.1) may be cleared (behavior may be implementation-specific).

The clearing of these bytes may occur even if the VM entry fails. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it causes processor state to be loaded from the host-state area of the VMCS.

- If the “use TPR shadow” VM-execution control is 1 and the “virtual-interrupt delivery” VM-execution control is 0, bits 31:4 of the TPR threshold VM-execution control field must be 0.⁶
- The following check is performed if the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” and “virtual-interrupt delivery” VM-execution controls are both 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 of VTPR (see Section 29.1.1).

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.

2. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

3. If IA32_VMX_BASIC[48] is read as 1, these addresses must not set any bits in the range 63:32; see Appendix A.1.

4. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.

5. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.

6. “Virtual-interrupt delivery” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtual-interrupt delivery” VM-execution control were 0. See Section 24.6.2.

- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” VM-execution control must be 0.
- If the “virtual NMIs” VM-execution control is 0, the “NMI-window exiting” VM-execution control must be 0.
- If the “virtualize APIC-accesses” VM-execution control is 1, the APIC-access address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address should not set any bits beyond the processor’s physical-address width.¹
- If the “use TPR shadow” VM-execution control is 0, the following VM-execution controls must also be 0: “virtualize x2APIC mode”, “APIC-register virtualization”, and “virtual-interrupt delivery”.²
- If the “virtualize x2APIC mode” VM-execution control is 1, the “virtualize APIC accesses” VM-execution control must be 0.
- If the “virtual-interrupt delivery” VM-execution control is 1, the “external-interrupt exiting” VM-execution control must be 1.
- If the “process posted interrupts” VM-execution control is 1, the following must be true:³
 - The “virtual-interrupt delivery” VM-execution control is 1.
 - The “acknowledge interrupt on exit” VM-exit control is 1.
 - The posted-interrupt notification vector has a value in the range 0–255 (bits 15:8 are all 0).
 - Bits 5:0 of the posted-interrupt descriptor address are all 0.
 - The posted-interrupt descriptor address does not set any bits beyond the processor’s physical-address width.⁴
- If the “enable VPID” VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.⁵
- If the “enable EPT” VM-execution control is 1, the EPT VM-execution control field (see Table 24-8 in Section 24.6.11) must satisfy the following checks:⁶
 - The EPT memory type (bits 2:0) must be a value supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10).
 - Bits 5:3 (1 less than the EPT page-walk length) must be 3, indicating an EPT page-walk length of 4; see Section 28.2.2.
 - Bit 6 (enable bit for accessed and dirty flags for EPT) must be 0 if bit 21 of the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10) is read as 0, indicating that the processor does not support accessed and dirty flags for EPT.
 - Reserved bits 11:7 and 63:N (where N is the processor’s physical-address width) must all be 0.
 - If the “unrestricted guest” VM-execution control is 1, the “enable EPT” VM-execution control must also be 1.⁷

-
1. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
 2. “Virtualize x2APIC mode” and “APIC-register virtualization” are secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if these controls were 0. See Section 24.6.2.
 3. “Process posted interrupts” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “process posted interrupts” VM-execution control were 0. See Section 24.6.2.
 4. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
 5. “Enable VPID” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “enable VPID” VM-execution control were 0. See Section 24.6.2.
 6. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “enable EPT” VM-execution control were 0. See Section 24.6.2.

- If the “enable VM functions” processor-based VM-execution control is 1, reserved bits in the VM-function controls must be clear.¹ Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.11). In addition, the following check is performed based on the setting of bits in the VM-function controls (see Section 24.6.14):
 - If “EPTP switching” VM-function control is 1, the “enable EPT” VM-execution control must also be 1. In addition, the EPTP-list address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address must not set any bits beyond the processor’s physical-address width.
- If the “enable VM functions” processor-based VM-execution control is 0, no checks are performed on the VM-function controls.

...

26.3.2.5 Updating Non-Register State

Section 28.3 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM entries invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).
- VM entries are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

If the “virtual-interrupt delivery” VM-execution control is 1, VM entry loads the values of RVI and SVI from the guest interrupt-status field in the VMCS (see Section 24.4.2). After doing so, the logical processor first causes PPR virtualization (Section 29.1.3) and then evaluates pending virtual interrupts (Section 29.2.1).

If a virtual interrupt is recognized, it may be delivered in VMX non-root operation immediately after VM entry (including any specified event injection) completes; see Section 26.6.5. See Section 29.2.2 for details regarding the delivery of virtual interrupts.

...

26.5.1.2 VM Exits During Event Injection

An event being injected never causes a VM exit directly regardless of the settings of the VM-execution controls. For example, setting the “NMI exiting” VM-execution control to 1 does not cause a VM exit due to injection of an NMI.

However, the event-delivery process may lead to a VM exit:

- If the vector in the VM-entry interruption-information field identifies a task gate in the IDT, the attempted task switch may cause a VM exit just as it would had the injected event occurred during normal execution in VMX non-root operation (see Section 25.4.2).
- If event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap (see Section 25.2).
- If event delivery generates a double-fault exception (due to a nested exception); the logical processor encounters another nested exception while attempting to call the double-fault handler; and that exception

7. “Unrestricted guest” and “enable EPT” are both secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if both these controls were 0. See Section 24.6.2.

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, VM entry functions as if the “enable VM functions” VM-execution control were 0. See Section 24.6.2.

does not cause a VM exit due to the exception bitmap; then a VM exit occurs due to triple fault (see Section 25.2).

- If event delivery injects a double-fault exception and encounters a nested exception that does not cause a VM exit due to the exception bitmap, then a VM exit occurs due to triple fault (see Section 25.2).
- If the “virtualize APIC accesses” VM-execution control is 1 and event delivery generates an access to the APIC-access page, that access is treated as described in Section 29.4 and may cause a VM exit.¹

If the event-delivery process does cause a VM exit, the processor state before the VM exit is determined just as it would be had the injected event occurred during normal execution in VMX non-root operation. If the injected event directly accesses a task gate that cause a VM exit or if the first nested exception encountered causes a VM exit, information about the injected event is saved in the IDT-vectoring information field (see Section 27.2.3).

...

26.6.3 Delivery of Pending Debug Exceptions after VM Entry

The pending debug exceptions field in the guest-state area indicates whether there are debug exceptions that have not yet been delivered (see Section 24.4.2). This section describes how these are treated on VM entry.

There are no pending debug exceptions after VM entry if any of the following are true:

- The VM entry is vectoring with one of the following interruption types: external interrupt, non-maskable interrupt (NMI), hardware exception, or privileged software exception.
- The interruptibility-state field does not indicate blocking by MOV SS and the VM entry is vectoring with either of the following interruption type: software interrupt or software exception.
- The VM entry is not vectoring and the activity-state field indicates either shutdown or wait-for-SIPI.

If none of the above hold, the pending debug exceptions field specifies the debug exceptions that are pending for the guest. There are **valid pending debug exceptions** if either the BS bit (bit 14) or the enable-breakpoint bit (bit 12) is 1. If there are valid pending debug exceptions, they are handled as follows:

- If the VM entry is not vectoring, the pending debug exceptions are treated as they would had they been encountered normally in guest execution:
 - If the logical processor is not blocking such exceptions (the interruptibility-state field indicates no blocking by MOV SS), a debug exception is delivered after VM entry (see below).
 - If the logical processor is blocking such exceptions (due to blocking by MOV SS), the pending debug exceptions are held pending or lost as would normally be the case.
- If the VM entry is vectoring (with interruption type software interrupt or software exception and with blocking by MOV SS), the following items apply:
 - For injection of a software interrupt or of a software exception with vector 3 (#BP) or vector 4 (#OF), the pending debug exceptions are treated as they would had they been encountered normally in guest execution if the corresponding instruction (INT3 or INTO) were executed after a MOV SS that encountered a debug trap.
 - For injection of a software exception with a vector other than 3 and 4, the pending debug exceptions may be lost or they may be delivered after injection (see below).

If there are no valid pending debug exceptions (as defined above), no pending debug exceptions are delivered after VM entry.

If a pending debug exception is delivered after VM entry, it has the priority of “traps on the previous instruction” (see Section 6.9 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). Thus, INIT

1. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 24.6.2.

signals and system-management interrupts (SMIs) take priority of such an exception, as do VM exits induced by the TPR threshold (see Section 26.6.7) and pending MTF VM exits (see Section 26.6.8). 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.

A pending debug exception delivered after VM entry causes a VM exit if the bit 1 (#DB) is 1 in the exception bitmap. If it does not cause a VM exit, it updates DR6 normally.

...

26.6.5 Interrupt-Window Exiting and Virtual-Interrupt Delivery

If “interrupt-window exiting” VM-execution control is 1, an open interrupt window may cause a VM exit immediately after VM entry (see Section 25.2 for details). If the “interrupt-window exiting” VM-execution control is 0 but the “virtual-interrupt delivery” VM-execution control is 1, a virtual interrupt may be delivered immediately after VM entry (see Section 26.3.2.5 and Section 29.2.1).

The following items detail the treatment of these events:

- These events occur after any event injection specified for VM entry.
- Non-maskable interrupts (NMIs) and higher priority events take priority over these events. These events take priority over external interrupts and lower priority events.
- These events wake the logical processor if it just entered the HLT state because of a VM entry (see Section 26.6.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

...

26.6.7 VM Exits Induced by the TPR Threshold

If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1 and the “virtual-interrupt delivery” VM-execution control is 0, a VM exit occurs immediately after VM entry if the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 of VTPR (see Section 29.1.1).¹

The following items detail the treatment of these VM exits:

- The VM exits are not blocked if RFLAGS.IF = 0 or by the setting of bits in the interruptibility-state field in guest-state area.
- The VM exits follow event injection if such injection is specified for VM entry.
- VM exits caused by this control take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. They thus have priority over the VM exits described in Section 26.6.5, Section 26.6.6, and Section 26.6.8, as well as any interrupts or debug exceptions that may be pending at the time of VM entry.
- These VM exits wake the logical processor if it just entered the HLT state as part of a VM entry (see Section 26.6.2). 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.”

...

1. “Virtualize APIC accesses” and “virtual-interrupt delivery” are secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if these controls were 0. See Section 24.6.2.

22. Updates to Chapter 27, Volume 3C

Change bars show changes to Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

...

VM exits occur in response to certain instructions and events in VMX non-root operation as detailed in Section 25.1 through Section 25.2. VM exits perform the following operations:

1. Information about the cause of the VM exit is recorded in the VM-exit information fields and VM-entry control fields are modified as described in Section 27.2.
2. Processor state is saved in the guest-state area (Section 27.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 27.4). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.
4. The following may be performed in parallel and in any order (Section 27.5):
 - Processor state is loaded based in part on the host-state area and some VM-exit controls. This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM. See Section 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 27.6). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.

VM exits are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

Section 27.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 27.2 through Section 27.6.

Section 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.

27.1 ARCHITECTURAL STATE BEFORE A VM EXIT

This section describes the architectural state that exists before a VM exit, especially for VM exits caused by events that would normally be delivered through the IDT. Note the following:

- An exception causes a VM exit **directly** if the bit corresponding to that exception is set in the exception bitmap. A non-maskable interrupt (NMI) causes a VM exit directly if the “NMI exiting” VM-execution control is 1. An external interrupt causes a VM exit directly if the “external-interrupt exiting” VM-execution control is 1. A start-up IPI (SIPI) that arrives while a logical processor is in the wait-for-SIPI activity state causes a VM exit directly. INIT signals that arrive while the processor is not in the wait-for-SIPI activity state cause VM exits directly.
- An exception, NMI, external interrupt, or software interrupt causes a VM exit **indirectly** if it does not do so directly but delivery of the event causes a nested exception, double fault, task switch, APIC access (see Section 29.4), EPT violation, or EPT misconfiguration 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.GD, or IA32_DEBUGCTL.LBR. (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 LO – 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 24.4.2) and not executing instructions, some events may be blocked but others may return the logical processor to the active state. Unblocked events may cause VM exits.² If an unblocked event causes a VM exit directly, a return to the active state occurs only after the VM exit completes.³ The VM exit generates any special bus cycle that is normally generated when the active state is entered from that activity state.

MTF VM exits (see Section 25.5.2 and Section 26.6.8) 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.

-
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.

- Processor state that is normally updated as part of delivery through the IDT (CS, RIP, SS, RSP, RFLAGS) is not modified. However, the incomplete delivery of the event may write to the stack.
- The treatment of last-exception records is implementation dependent:
 - Some processors make a last-exception record when beginning the delivery of an event through the IDT (before it can encounter a nested exception). Such processors perform this update even if the event encounters a nested exception that causes a VM exit (including the case where nested exceptions lead to a triple fault).
 - Other processors delay making a last-exception record until event delivery has reached some event handler successfully (perhaps after one or more nested exceptions). Such processors do not update the last-exception record if a VM exit or triple fault occurs before an event handler is reached.
- 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, 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, or EPT misconfiguration 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 VM-exit interruption-information field; see Section 27.2.2.
- If a VM exit results from a fault, EPT violation, or EPT misconfiguration 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 virtual-NMI blocking may be recorded in the VM-exit interruption-information field; see Section 27.2.2.
- Suppose that a VM exit is caused directly by an x87 FPU Floating-Point Error (#MF) or by any of the following events if the event was unblocked due to (and given priority over) an x87 FPU Floating-Point Error: an INIT signal, an external interrupt, an NMI, an SMI; or a machine-check exception. In these cases, there is no blocking by STI or by MOV SS when the VM exit commences.
- Normally, a last-branch record may be made when an event is delivered through the IDT. However, if such an event results in a VM exit before delivery is complete, no last-branch record is made.
- If machine-check exception results in a VM exit, processor state is suspect and may result in suspect state being saved to the guest-state area. A VM monitor should consult the RIPV and EIPV bits in the IA32_MCG_STATUS MSR before resuming a guest that caused a VM exit resulting from a machine-check exception.
- If a VM exit results from a fault, APIC access (see Section 29.4), EPT violation, or EPT misconfiguration 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 (see Section 27.3.4).
- The following VM exits are considered to happen after an instruction is executed:
 - VM exits resulting from debug traps (single-step, I/O breakpoints, and data breakpoints).
 - VM exits resulting from debug exceptions whose recognition was delayed by blocking by MOV SS.
 - VM exits resulting from some machine-check exceptions.
 - Trap-like VM exits due to execution of MOV to CR8 when the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1 (see Section 29.3). (Such VM exits can occur only from 64-bit mode and thus only on processors that support Intel 64 architecture.)
 - Trap-like VM exits due to execution of WRMSR when the “use MSR bitmaps” VM-execution control is 1; the value of ECX is in the range 800H–8FFH; and the bit corresponding to the ECX value in write bitmap for low MSRs is 0; and the “virtualize x2APIC mode” VM-execution control is 1. See Section 29.5.
 - VM exits caused by APIC-write emulation (see Section 29.4.3.2) that result from APIC accesses as part of instruction execution.

For these VM exits, the instruction's modifications to architectural state complete before the VM exit occurs. Such modifications include those to the logical processor's interruptibility state (see Table 24-3). If there had been blocking by MOV SS, POP SS, or STI before the instruction executed, such blocking is no longer in effect.

27.2.1 Basic VM-Exit Information

Section 24.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix C lists the numbers used and their meaning.
 - The remainder of the field (bits 31:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 34.15.2.3).¹
- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); system-management interrupts (SMIs) that arrive immediately after the retirement of I/O instructions; task switches; INVEPT; INVLPG; INVPCID; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 29.4); EPT violations; EOI virtualization (Section 29.1.4); and APIC-write emulation (see Section 29.4.3.3). For all other VM exits, this field is cleared. The following items provide details:

For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 24-4.

...

- For an APIC-access VM exit resulting from a linear access or a guest-physical access to the APIC-access page (see Section 29.4), the exit qualification contains information about the access and has the format given in Table 27-6.²

Table 27-6 Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	<ul style="list-style-type: none"> ▪ 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	<p>Access type:</p> <ul style="list-style-type: none"> 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 10 = guest-physical access during event delivery 15 = guest-physical access for an instruction fetch or during instruction execution <p>Other values not used</p>
63:16	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

1. Bit 13 of this field is set on certain VM-entry failures; see Section 26.7.
2. The exit qualification is undefined if the access was part of the logging of a branch record or a precise-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.

Such a VM exit that set bits 15:12 of the exit qualification to 0000b (data read during instruction execution) or 0001b (data write during instruction execution) set bit 12—which distinguishes data read from data write—to that which would have been stored in bit 1—W/R—of the page-fault error code had the access caused a page fault instead of an APIC-access VM exit. This implies the following:

- For an APIC-access VM exit caused by the CLFLUSH instruction, 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.”
- 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.”

Such a VM exit stores 1 for bit 31 for IDT-vectoring information field (see Section 27.2.3) if and only if it sets bits 15:12 of the exit qualification to 0011b (linear access during event delivery) or 1010b (guest-physical access during event delivery).

See Section 29.4.4 for further discussion of these instructions and APIC-access VM exits.

For APIC-access VM exits resulting from physical accesses, the APIC-access page (see Section 29.4.6), the exit qualification is undefined.

...

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.

Bit 12 is undefined in any of the following cases:

- If the “NMI exiting” VM-execution control is 1 and the “virtual NMIs” VM-execution control is 0.
- If the VM exit sets the valid bit in the IDT-vectoring information field (see Section 27.2.3).

Otherwise, bit 12 is defined as follows:

- If the “virtual NMIs” VM-execution control is 0, the EPT violation was caused by a memory access as part of execution of the IRET instruction, and blocking by NMI (see Table 24-3) was in effect before execution of IRET, bit 12 is set to 1.
 - If the “virtual NMIs” VM-execution control is 1, the EPT violation was caused by a memory access as part of execution of the IRET instruction, and virtual-NMI blocking was in effect before execution of IRET, bit 12 is set to 1.
 - For all other relevant VM exits, bit 12 is cleared to 0.
- For VM exits caused as part of EOI virtualization (Section 29.1.4), bits 7:0 of the exit qualification are set to vector of the virtual interrupt that was dismissed by the EOI virtualization. Bits above bit 7 are cleared.
 - For APIC-write VM exits (Section 29.4.3.3), bits 11:0 of the exit qualification are set to the page offset of the write access that caused the VM exit.¹ Bits above bit 11 are cleared.

...

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.

27.2.3 Information for VM Exits During Event Delivery

Section 24.9.3 defined fields containing information for VM exits that occur while delivering an event through the IDT and as a result of any of the following cases:¹

- A fault occurs during event delivery and causes a VM exit (because the bit associated with the fault is set to 1 in the exception bitmap).
- A task switch is invoked through a task gate in the IDT. The VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 25.4.2).
- Event delivery causes an APIC-access VM exit (see Section 29.4).
- An EPT violation or EPT misconfiguration that occurs during event delivery.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 26.5.1.2).

...

27.2.4 Information for VM Exits Due to Instruction Execution

Section 24.9.4 defined fields containing information for VM exits that occur due to instruction execution. (The VM-exit instruction length is also used for VM exits that occur during the delivery of a software interrupt or software exception.) The following items detail their use.

- **VM-exit instruction length.** This field is used in the following cases:
 - For fault-like VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 25.1.2) or based on the settings of VM-execution controls (see Section 25.1.3): CLTS, CPUID, GETSEC, HLT, IN, INS, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, RDMSR, RDPIC, RDRAND, RDTSC, RDTSCP, RSM, SGDT, SIDT, SLDT, STR, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WRMSR, and XSETBV.²
 - For VM exits due to software exceptions (those generated by executions of INT3 or INTO).
 - For VM exits due to faults encountered during delivery of a software interrupt, privileged software exception, or software exception.
 - For VM exits due to attempts to effect a task switch via instruction execution. These are VM exits that produce an exit reason indicating task switch and either of the following:
 - An exit qualification indicating execution of CALL, IRET, or JMP instruction.
 - An exit qualification indicating a task gate in the IDT and an IDT-vectoring information field indicating that the task gate was encountered during delivery of a software interrupt, privileged software exception, or software exception.
 - For APIC-access VM exits resulting from accesses (see Section 29.4) during delivery of a software interrupt, privileged software exception, or software exception.³

...

1. This includes the case in which a VM exit occurs while delivering a software interrupt (INT *n*) through the 16-bit IVT (interrupt vector table) that is used in virtual-8086 mode with virtual-machine extensions (if RFLAGS.VM = CR4.VME = 1).
2. 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.
3. The VM-exit instruction-length field is not defined following APIC-access VM exits resulting from physical accesses (see Section 29.4.6) even if encountered during delivery of a software interrupt, privileged software exception, or software exception.

27.3.3 Saving RIP, RSP, and RFLAGS

The contents of the RIP, RSP, and RFLAGS 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 occurs due to by an attempt to execute an instruction that causes VM exits unconditionally or that has been configured to cause a VM exit via the VM-execution controls, the value saved references that instruction.
 - If the VM exit is caused by an occurrence of an INIT signal, a start-up IPI (SIPI), or system-management interrupt (SMI), the value saved is that which was in RIP before the event occurred.
 - If the VM exit occurs due to the 1-setting of either the “interrupt-window exiting” VM-execution control or the “NMI-window exiting” VM-execution control, the value saved is that which would be in the register had the VM exit not occurred.
 - If the VM exit is due to an external interrupt, non-maskable interrupt (NMI), or hardware exception (as defined in Section 27.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,¹ or into the old task-state segment had the event been delivered through a task gate).
 - If the VM exit is due to a triple fault, the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate, or into the old task-state segment had the event been delivered through a task gate) had delivery of the double fault not encountered the nested exception that caused the triple fault.
 - If the VM exit is due to a software exception (due to an execution of INT3 or INTO), the value saved references the INT3 or INTO 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*) or software exception (due to execution of INT3 or INTO) 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, or INTO).
 - 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 29.1.1) below that of TPR threshold VM-execution control field (see Section 29.1.2), the value saved references the instruction following the MOV to CR8 or WRMSR.
 - If the VM exit was caused by APIC-write emulation (see Section 29.4.3.2) that results from an APIC access as part of instruction execution, the value saved references the instruction following the one whose execution caused the APIC-write emulation.
- The contents of the RSP register are saved into the RSP field.
- With the exception of the resume flag (RF; bit 16), the contents of the RFLAGS register is saved into the RFLAGS field. RFLAGS.RF is saved as follows:
 - If the VM exit is caused directly by an event that would normally be delivered through the IDT, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate² or into the old task-state segment

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.

2. 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.

had the event been delivered through a task gate) had the event been delivered through the IDT. See below for VM exits due to task switches caused by task gates in the IDT.

- If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
- If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.
- If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.¹
- For APIC-access VM exits and for VM exits caused by EPT violations and EPT misconfigurations, the value saved depends on whether the VM exit occurred during delivery of an event through the IDT:
 - If the VM exit stored 0 for bit 31 for IDT-vectoring information field (because the VM exit did not occur during delivery of an event through the IDT; see Section 27.2.3), the value saved is 1.
 - If the VM exit stored 1 for bit 31 for IDT-vectoring information field (because the VM exit did occur during delivery of an event through the IDT), the value saved is the value that would have appeared in the saved RFLAGS image had the event been delivered through the IDT (see above).
- For all other VM exits, the value saved is the value RFLAGS.RF had before the VM exit occurred.

...

27.3.4 Saving Non-Register State

Information corresponding to guest non-register state is saved as follows:

- The activity-state field is saved with the logical processor's activity state before the VM exit.² See Section 27.1 for details of how events leading to a VM exit may affect the activity state.
- The interruptibility-state field is saved to reflect the logical processor's interruptibility before the VM exit. See Section 27.1 for details of how events leading to a VM exit may affect this state. 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.
- 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",³ "virtualized EOI", "APIC write", or "monitor trap flag."
 - 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:

1. 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.
2. 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.
3. This item includes VM exits that occur as a result of certain VM entries (Section 26.6.7).

- Each of bits 3:0 may be set if it corresponds to a matched breakpoint. This may be true even if the corresponding breakpoint is not enabled in DR7.
- Suppose that a VM exit is due to an INIT signal, a machine-check exception, or an SMI; or that a VM exit has basic exit reason “TPR below threshold” or “monitor trap flag.” In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit.

If the VM exit occurs immediately after VM entry, the value saved may match that which was loaded on VM entry (see Section 26.6.3). Otherwise, the following items apply:

- Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 26.6.3) and the VM exit occurred before those exceptions were either delivered or lost. 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.
- Suppose that a VM exit is due to another reason (but not a debug exception) and occurs while there is MOV-SS blocking of debug exceptions. In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit. If the VM exit occurs immediately after VM entry (no instructions were executed in VMX non-root operation), the value saved may match that which was loaded on VM entry (see Section 26.6.3). Otherwise, the following items apply:
 - Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 26.6.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
 - The setting of bit 14 (BS) is implementation-specific. However, it is not set if RFLAGS.TF = 0 or IA32_DEBUGCTL.BTF = 1.
- The reserved bits in the field are cleared.
- If the “save VMX-preemption timer value” VM-exit control is 1, the value of timer is saved into the VMX-preemption timer-value field. This is the value loaded from this field on VM entry as subsequently decremented (see Section 25.5.1). VM exits due to timer expiration save the value 0. Other VM exits may also save the value 0 if the timer expired during VM exit. (If the “save VMX-preemption timer value” VM-exit control is 0, VM exit does not modify the value of the VMX-preemption timer-value field.)
- If the logical processor supports the 1-setting of the “enable EPT” VM-execution control, values are saved into the four (4) PDPTE fields as follows:
 - If the “enable EPT” VM-execution control is 1 and the logical processor was using PAE paging at the time of the VM exit, the PDPTE values currently in use are saved:¹
 - The values saved into bits 11:9 of each of the fields is undefined.
 - If the value saved into one of the fields has bit 0 (present) clear, the value saved into bits 63:1 of that field is undefined. That value need not correspond to the value that was loaded by VM entry or to any value that might have been loaded in VMX non-root operation.
 - If the value saved into one of the fields has bit 0 (present) set, the value saved into bits 63:12 of the field is a guest-physical address.

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM exit functions as if the “enable EPT” VM-execution control were 0. See Section 24.6.2.

- 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.

...

23. Updates to Chapter 28, Volume 3C

Change bars show changes to Chapter 28 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C: System Programming Guide, Part 3*.

...

28.3.3.3 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 (e.g., see Section 32.3, “Memory Virtualization”).

If EPT is not in use, it is likely that the VMM is virtualizing the guest paging structures. Such a VMM may configure the VMCS so that all or some of the operations that invalidate entries the TLBs and the paging-structure caches (e.g., the INVLPG instruction) cause VM exits. If VMM software is emulating these operations, it may be necessary to use the INVVPID instruction to ensure that the logical processor’s TLBs and the paging-structure caches are appropriately invalidated.

Requirements of when software should use the INVVPID instruction depend on the specific algorithm being used for page-table virtualization. The following items provide guidelines for software developers:

- Emulation of the INVLPG instruction may require execution of the INVVPID instruction as follows:
 - The INVVPID type is individual-address (0).
 - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
 - The linear address in the INVVPID descriptor is that of the operand of the INVLPG instruction being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—except for global translations. An example is the MOV to CR3 instruction. (See Section 4.10, “Caching Translation Information” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A* for details regarding global translations.) Emulation of such an instruction may require execution of the INVVPID instruction as follows:
 - The INVVPID type is single-context-retaining-globals (3).
 - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—including for global translations. An example is the MOV to CR4 instruction if the value of value of bit 4 (page global enable—PGE) is changing. Emulation of such an instruction may require execution of the INVVPID instruction as follows:
 - The INVVPID type is single-context (1).
 - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.

If EPT is not in use, the logical processor associates all mappings it creates with the current VPID, and it will use such mappings to translate linear addresses. For that reason, a VMM should not use the same VPID for different non-EPT guests that use different page tables. Doing so may result in one guest using translations that pertain to the other.

If EPT is in use, the instructions enumerated above might not be configured to cause VM exits and the VMM might not be emulating them. In that case, executions of the instructions by guest software properly invalidate the required entries in the TLBs and paging-structure caches (see Section 28.3.3.1); execution of the INVVPID instruction is not required.

If EPT is in use, the logical processor associates all mappings it creates with the value of bits 51:12 of current EPTP. If a VMM uses different EPTP values for different guests, it may use the same VPID for those guests. Doing so cannot result in one guest using translations that pertain to the other.

The following guidelines apply more generally and are appropriate even if EPT is in use:

- As detailed in Section 29.4.5, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the paging structures. If, at one time, the current VPID on a logical processor was a non-zero value X, it is recommended that software use the INVVPID instruction with the “single-context” INVVPID type and with VPID X in the INVVPID descriptor before a VM entry on the same logical processor that establishes VPID X and either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.
- Software can use the INVVPID instruction with the “all-context” INVVPID type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from paging structures between separate uses of VMX operation.

28.3.3.4 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.
 - Changing the physical address in bits 51:12.
 - Clearing bit 8 (the accessed flag) if accessed and dirty flags for EPT will be enabled.
 - For an EPT PDPTE or an EPT PDE, changing bit 7 (which determines whether the entry maps a page).
 - For the **last** EPT paging-structure entry used to translate a guest-physical address (an EPT PDPTE with bit 7 set to 1, an EPT PDE with bit 7 set to 1, or an EPT PTE), changing either bits 5:3 or bit 6. (These bits determine the effective memory type of accesses using that EPT paging-structure entry; see Section 28.2.5.)
 - For the **last** EPT paging-structure entry used to translate a guest-physical address (an EPT PDPTE with bit 7 set to 1, an EPT PDE with bit 7 set to 1, or an EPT PTE), clearing bit 9 (the dirty flag) if accessed and dirty flags for EPT will be enabled.
- Software should use the INVEPT instruction with the “single-context” INVEPT type before a VM entry with an EPTP value X such that $X[6] = 1$ (accessed and dirty flags for EPT are enabled) if the logical processor had earlier been in VMX non-root operation with an EPTP value Y such that $Y[6] = 0$ (accessed and dirty flags for EPT are not enabled) and $Y[51:12] = X[51:12]$.
- Software may use the INVEPT instruction after modifying a present EPT paging-structure entry to change any of the privilege bits 2:0 from 0 to 1. Failure to do so may cause an EPT violation that would not otherwise occur. Because an EPT violation invalidates any mappings that would be used by the access that caused the EPT violation (see Section 28.3.3.1), an EPT violation will not recur if the original access is performed again, even if the INVEPT instruction is not executed.

- Because a logical processor does not cache any information derived from EPT paging-structure entries that are not present or misconfigured (see Section 28.2.3.1), it is not necessary to execute INVEPT following modification of an EPT paging-structure entry that had been not present or misconfigured.
- As detailed in Section 29.4.5, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the EPT paging structures. If EPT was in use on a logical processor at one time with EPTP X, it is recommended that software use the INVEPT instruction with the “single-context” INVEPT type and with EPTP X in the INVEPT descriptor before a VM entry on the same logical processor that enables EPT with EPTP X and either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.
- Software can use the INVEPT instruction with the “all-context” INVEPT type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from EPT paging structures between separate uses of VMX operation.

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 4.10.5, “Propagation of Paging-Structure Changes to Multiple Processors,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

...

24. Updates to Chapter 29, Volume 3C

Chapter 29 is a new chapter of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C: System Programming Guide, Part 3*.

...

CHAPTER 29 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 with out 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 29.1 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 24.6 for information about the locations of these controls):

- **Virtual-interrupt delivery.** This controls enables the evaluation and delivery of pending virtual interrupts (Section 29.2). 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 29.3) and, if enabled, via the memory-mapped or MSR-based interfaces.

1. In most cases, it is not necessary for a virtual-machine monitor (VMM) to inject virtual interrupts as part of VM entry.

- **Virtualize APIC accesses.** This control enables virtualization of memory-mapped accesses to the APIC (Section 29.4) 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 29.5).
- **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 29.6).

“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. See Section 24.6.2.

29.1 VIRTUAL APIC STATE

The **virtual-APIC page** is a 4-KByte region of memory that the processor uses the virtual-APIC page 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 24.6.8).

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 29.1.1 identifies and defines these fields. Section 29.1.2, Section 29.1.3, Section 29.1.4, and Section 29.1.5 detail the actions taken to virtualize updates to some of these fields.

29.1.1 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.

29.1.2 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 29.3, Section 29.4.3, and Section 29.5 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 24.6.8)
            THEN cause VM exit due to TPR below threshold;
        FI;
    ELSE
        perform PPR virtualization (see Section 29.1.3);
        evaluate pending virtual interrupts (see Section 29.2.1);
    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).

29.1.3 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 26.3.2.5, Section 29.1.2, and Section 29.1.4 for details of when PPR virtualization is performed.

PPR virtualization uses the guest interrupt status (specifically, SVI; see Section 24.4.2) 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 & F0H;
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 29.2.2). No other operations modify VPPR, even if they modify SVI, VISR, or VTPR.

29.1.4 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 29.4.3 and Section 29.5 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 24.4.2). The following pseudocode details the behavior of EOI virtualization:

```
Vector ← SVI;
VISR[Vector] ← 0; (see Section 29.1.1 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 29.1.3);
  IF EOI_exit_bitmap[Vector] = 1 (see Section 24.6.8 for definition of EOI_exit_bitmap)
    THEN cause EOI-induced VM exit with Vector as exit qualification;
    ELSE evaluate pending virtual interrupts; (see Section 29.2.1)
  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).

29.1.5 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 29.4.3 and Section 29.5 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 29.4.3 and Section 29.5). Self-IPI virtualization updates the guest interrupt status (specifically, RVI; see Section 24.4.2). The following pseudocode details the behavior of self-IPI virtualization:

```

  VIRR[Vector] ← 1; (see Section 29.1.1 for definition of VIRR)
  RVI ← max{RVI, Vector};
  evaluate pending virtual interrupts; (see Section 29.2.1)

```

29.2 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 29.2.1. 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 29.2.2.

29.2.1 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 26.3.2.5, Section 29.1.2, Section 29.1.4, Section 29.1.5, and Section 29.6 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.

Evaluation of pending virtual interrupts uses the guest interrupt status (specifically, RVI; see Section 24.4.2). The following pseudocode details the evaluation of pending virtual interrupts:

```

  IF “interrupt-window exiting” is 0 AND
  RVI[7:4] > VPPR[7:4] (see Section 29.1.1 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 29.2.2.

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.

29.2.2 Virtual-Interrupt Delivery

If a virtual interrupt has been recognized (see Section 29.2.1), it will be 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 24.4.2) 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 29.1.1 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;
deliver interrupt with Vector through IDT;
cease recognition of any pending virtual interrupt;
```

29.3 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 10.8.6, “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 25.1.3). Alternatively, there are methods for virtualizing some CR8-based APIC accesses without VM exits.

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.

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 29.1.1). 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 29.1.2).

29.4 VIRTUALIZING MEMORY-MAPPED APIC ACCESSES

When the local APIC is in xAPIC mode, software accesses the local APIC's control registers using a memory-mapped interface. Specifically, software uses linear addresses that translate to physical addresses on page frame indicated by the base address in the IA32_APIC_BASE MSR (see Section 10.4.4, "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 (The APIC-access page is identified by the **APIC-access address**, a field in the VMCS; see Section 24.6.8.)

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 29.4.1 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 29.4.6 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 29.4.2 discusses when and how the processor may virtualize read accesses from the APIC-access page. Section 29.4.3 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.

1. Even when addresses are translated using EPT (see Section 28.2), 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.

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.

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 29.4.6.

29.4.1 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.

29.4.2 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” VM-execution control:

- If “APIC-register virtualization” is 0, a read access is virtualized if its page offset is 080H (task priority); otherwise, the access causes an APIC-access VM exit.
- 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

29.4.3 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 29.4.3.1 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 29.4.3.2 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 29.4.3.3 discusses APIC-write VM exits.

29.4.3.1 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” and “virtual-interrupt delivery” VM-execution controls:

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).

- If the “APIC-register virtualization” and “virtual-interrupt delivery” VM-execution controls are both 0, a write access is virtualized if its page offset is 080H; otherwise, the access causes an APIC-access VM exit.
- If the “APIC-register virtualization” VM-execution control is 0 and the “virtual-interrupt delivery” VM-execution control is 1, a write access is virtualized if its page offset is 080H (task priority), 0B0H (end of interrupt), and 300H (interrupt command — low); otherwise, the access causes an APIC-access VM exit.
- If “APIC-register virtualization” is 1, a write access is virtualized if it is entirely within one of 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);
 - 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 29.4.3.2.

29.4.3.2 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 29.1.2).
- 0B0H (end of interrupt). If the “virtual-interrupt delivery” VM-execution control is 1, the processor clears VEOI and then causes EOI virtualization (Section 29.1.4); otherwise, the processor causes an APIC-write VM exit (Section 29.4.3.3).
- 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).

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.
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.

- 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 29.1.5).

If the “virtual-interrupt delivery” VM-execution control is 0, or if any of the items above are false, the processor causes an APIC-write VM exit (Section 29.4.3.3).

- 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 (Section 29.4.3.3).

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. In this case, APIC-write emulation occurs after the fault is delivered and before the fault handler can execute. However, if the operation causes a VM exit (perhaps due to a fault), the APIC-write emulation does not occur.

29.4.3.3 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 29.4.3.2 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 29.5, 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 such an APIC-write VM exit is 3F0H.

29.4.4 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**. With regard to faulting, the processor operates as if CLFLUSH 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 MAKSMOVDQU**. 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 instructions that normally operate on floating-point, SSE, or AVX registers may cause APIC-access VM exit unconditionally regardless of the page offset they access on the APIC-access page.

29.4.5 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 29.4.6.1.)

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 30.3) 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 30.3) 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.)

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.

29.4.6 APIC Accesses Not Directly Resulting From Linear Addresses

Section 29.4 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.

- 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 29.4.6.1 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 29.4.6.2 discusses the treatment of physical accesses to the APIC-access page.

29.4.6.1 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 4.4).
- 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).

Every guest-physical access 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.

29.4.6.2 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 4.4).
 - 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).

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 24.6.2.

- Any of the following accesses made by the processor to support VMX non-root operation:
 - Accesses to the VMCS region.
 - Accesses to data structures referenced (directly or indirectly) by physical addresses in VM-execution control fields in the VMCS. These include the I/O bitmaps, the MSR bitmaps, and the virtual-APIC page.
- Accesses that effect transitions into and out of SMM.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.

29.5 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 10.12, “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 24.6.9) to ensure that the accesses cause VM exits (see Section 25.1.3). 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” VM-execution control.

1. Technically, these accesses do not occur in VMX non-root operation. They are included here for clarity.

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; and (3) ECX contains 83FH (indicating the self-IPI MSR) and the “virtual-interrupt delivery” VM-execution control is 1.

If special processing applies, no general-protection exception is produced due to the fact that the local APIC is in xAPIC mode. However, WRMSR does perform the normal reserved-bit checking:

- If ECX contains 808H or 83FH, a general-protection fault occurs if either EDX or EAX[31:8] is non-zero.
- If ECX contains 80BH, a general-protection fault occurs if either EDX or EAX is non-zero.

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 29.1.2).
- If ECX contains 80BH, the processor performs EOI virtualization (see Section 29.1.4).
- If ECX contains 83FH, the processor It 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 29.1.5).
 - 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 29.4.3.3).

If special processing does not apply, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a writeable 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 writeable APIC register, a general-protection fault occurs.

29.6 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**. The posted-interrupt notification vector and the address of the posted-interrupt descriptor are fields in the VMCS; see Section 24.6.8.

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. The posted-interrupt descriptor has the following format:

Table 0-1. Format of Posted-Interrupt Descriptor

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1
256	Outstanding notification	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0
511:257	Reserved for software and other agents	These bits may be used by software and by other agents in the system (e.g., chipset). The processor does not modify these bits.

The notation **PIR** (posted-interrupt requests) refers to the 256 posted-interrupt bits in the posted-interrupt descriptor.

Use of the posted-interrupt descriptor 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.

If the “external-interrupt exiting” VM-execution control is 1, any unmasked external interrupt causes a VM exit (see Section 25.2). 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 VM-exit interruption-information 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 29.2.1.

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. This 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).

...

25. Updates to Chapter 34, Volume 3C

Change bars show changes to Chapter 34 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

...

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 26.2.1.1.

Table 34-3 SMRAM State Save Map for Intel 64 Architecture

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FF8H	CRO	No
7FF0H	CR3	No
7FE8H	RFLAGS	Yes
7FE0H	IA32_EFER	Yes
7FD8H	RIP	Yes
7FD0H	DR6	No
7FC8H	DR7	No
7FC4H	TR SEL ¹	No
7FC0H	LDTR SEL ¹	No
7FBCH	GS SEL ¹	No
7FB8H	FS SEL ¹	No
7FB4H	DS SEL ¹	No
7FB0H	SS SEL ¹	No
7FACH	CS SEL ¹	No
7FA8H	ES SEL ¹	No
7FA4H	IO_MISC	No
7F9CH	IO_MEM_ADDR	No
7F94H	RDI	Yes
7F8CH	RSI	Yes
7F84H	RBP	Yes
7F7CH	RSP	Yes
7F74H	RBX	Yes
7F6CH	RDX	Yes
7F64H	RCX	Yes
7F5CH	RAX	Yes
7F54H	R8	Yes
7F4CH	R9	Yes
7F44H	R10	Yes
7F3CH	R11	Yes
7F34H	R12	Yes
7F2CH	R13	Yes
7F24H	R14	Yes
7F1CH	R15	Yes
7F1BH-7F04H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes

Table 34-3 SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7EE4H	Reserved	No
7EE0H	Setting of "enable EPT" VM-execution control	No
7ED8H	Value of EPTP VM-execution control field	No
7ED7H - 7EA0H	Reserved	No
7E9CH	LDT Base (lower 32 bits)	No
7E98H	Reserved	No
7E94H	IDT Base (lower 32 bits)	No
7E90H	Reserved	No
7E8CH	GDT Base (lower 32 bits)	No
7E8BH - 7E44H	Reserved	No
7E40H	CR4	No
7E3FH - 7DF0H	Reserved	No
7DE8H	IO_RIP	Yes
7DE7H - 7DDCH	Reserved	No
7DD8H	IDT Base (Upper 32 bits)	No
7DD4H	LDT Base (Upper 32 bits)	No
7DD0H	GDT Base (Upper 32 bits)	No
7DCFH - 7C00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

...

34.15.6.4 Saving MSRs

The VM-exit MSR-store area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are saved into that area.

...

26. Updates to Chapter 35, Volume 3C

Change bars show changes to Chapter 35 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

...

Table 35-1 CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_3CH, 06_45H	Next Generation Intel Core Processor
06_3EH	Next Generation Intel Xeon Processor E5 Family based on Intel microarchitecture Ivy Bridge
06_3AH	3rd Generation Intel Core Processor and Intel Xeon Processor E3-1200v2 Product Family based on Intel microarchitecture Ivy Bridge
06_2DH	Intel Xeon Processor E5 Family based on Intel microarchitecture Sandy Bridge
06_2FH	Intel Xeon Processor E7 Family
06_2AH	Intel Xeon Processor E3-1200 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_1CH, 06_26H, 06_27H	Intel Atom Processor Family
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

...

Table 35-2 IA-32 Architectural MSRs

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
...				
3BH	59	IA32_TSC_ADJUST	Per Logical Processor TSC Adjust (R/Write to clear)	If CPUID.(EAX=07H, ECX=0H): EBX[1] = 1
		63:0	THREAD_ADJUST: Local offset value of the IA32_TSC for a logical processor. Reset value is Zero. A write to IA32_TSC will modify the local offset in IA32_TSC_ADJUST and the content of IA32_TSC, but does not affect the internal invariant TSC hardware.	
...				
C000_0080H		IA32_EFER	Extended Feature Enables	If (CPUID.80000001.EDX.[bit 20] or CPUID.80000001.EDX.[bit 29])
		0	SYSCALL Enable (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.	
		7:1	Reserved.	
		8	IA-32e Mode Enable (R/W) Enables IA-32e mode operation.	
		9	Reserved.	
		10	IA-32e Mode Active (R) Indicates IA-32e mode is active when set.	
		11	Execute Disable Bit Enable (R/W)	
		63:12	Reserved.	
...				

...

35.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table 35-3 lists model-specific registers (MSRs) for Intel Core 2 processor family and for Intel Xeon processors based on Intel Core microarchitecture, architectural MSR addresses are also included in Table 35-3. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_0FH, see Table 35-1.

MSRs listed in Table 35-2 and Table 35-3 are also supported by processors based on the Enhanced Intel Core microarchitecture. Processors based on the Enhanced Intel Core microarchitecture have the CPUID signature DisplayFamily_DisplayModel of 06_17H.

The column “Shared/Unique” applies to multi-core processors based on Intel Core microarchitecture. “Unique” means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. “Shared” means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Unique	See Section 35.14, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Unique	See Section 35.14, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, “Monitor/Mwait Address Range Determination.” and Table 35-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.13, “Time-Stamp Counter,” and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 35-2.
17H	23	MSR_PLATFORM_ID	Shared	Model Specific Platform ID (R)
		7:0		Reserved.
		12:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		49:13		Reserved.
		52:50		See Table 35-2.
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, “Local APIC Status and Location.” and Table 35-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved.
		1		Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all 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.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		5		Reserved.
		6		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		Intel TXT Capable Chipset. (R/O) 1 = Present; 0 = Not Present
		12		BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled
		13		Reserved.
		14		1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved.
		17:16		APIC Cluster ID (R/O)
		18		N/2 Non-Integer Bus Ratio (R/O) 0 = Integer ratio; 1 = Non-integer ratio
		19		Reserved.
		21: 20		Symmetric Arbitration ID (R/O)
		26:22		Integer Bus Frequency Ratio (R/O)
3AH	58	IA32_FEATURE_CONTROL	Unique	Control Features in Intel 64Processor (R/W) See Table 35-2.
		3	Unique	SMRR Enable (R/WL) When this bit is set and the lock bit is set makes the SMRR_PHYS_BASE and SMRR_PHYS_MASK registers read visible and writeable while in SMM.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
40H	64	MSR_LASTBRANCH_0_FROM_IP	Unique	Last Branch Record 0 From IP (R/W) One of four pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction for one of the last four branches, exceptions, or interrupts taken by the processor. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.11, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	Last Branch Record 0 To IP (R/W) One of four pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last four branches, exceptions, or interrupts taken by the processor.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	BIOS Update Trigger Register (W) See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (RO) See Table 35-2.
A0H	160	MSR_SMRR_PHYSBASE	Unique	System Management Mode Base Address register (W0 in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		11:0		Reserved.
		31:12		PhysBase. SMRR physical Base Address.
		63:32		Reserved.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
A1H	161	MSR_SMRR_PHYSMASK	Unique	System Management Mode Physical Address Mask register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM..
		10:0		Reserved.
		11		Valid. Physical address base and range mask are valid.
		31:12		PhysMask. SMRR physical address range mask.
		63:32		Reserved.
C1H	193	IA32_PMC0	Unique	Performance Counter Register See Table 35-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 35-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed(RO) This field indicates the intended scaleable bus clock speed for processors based on Intel Core microarchitecture: <ul style="list-style-type: none"> ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.
		2:0		
		63:3		Reserved.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed(RO) This field indicates the intended scaleable bus clock speed for processors based on Enhanced Intel Core microarchitecture:
		2:0		

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
				133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
				266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (RW) See Table 35-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (RW) See Table 35-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 35-2.
		11	Unique	SMRR Capability Using MSR 0A0H and 0A1H (R)
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		L2 Hardware Enabled (RO) 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		L2 Enabled (R/W) 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.
		23		L2 Not Present (RO) 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 35-2.
179H	377	IA32_MCG_CAP	Unique	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Unique	

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSELO	Unique	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 35-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 35-2.
198H	408	MSR_PERF_STATUS	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.
199H	409	IA32_PERF_CTL	Unique	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 35-2.
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 35-2.
19DH	413	MSR_THERM2_CTL	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.
1A0	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 35-2.
		2:1		Reserved.
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 35-2.
		6:4		Reserved.
		7	Shared	Performance Monitoring Available (R) See Table 35-2.
		8		Reserved.
		9		Hardware Prefetcher Disable (R/W) When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.
10	Shared	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.		

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		11	Shared	Branch Trace Storage Unavailable (RO) See Table 35-2.
		12	Shared	Precise Event Based Sampling Unavailable (RO) See Table 35-2.
		13	Shared	<p>TM2 Enable (R/W)</p> <p>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.</p> <p>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.</p> <p>The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location.</p> <p>The processor is operating out of specification if both this bit and the TM1 bit are set to 0.</p>
		15:14		Reserved.
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 35-2.
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 35-2.
		19	Shared	<p>Adjacent Cache Line Prefetch Disable (R/W)</p> <p>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).</p> <p>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.</p> <p>BIOS may contain a setup option that controls the setting of this bit.</p>

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		20	Shared	<p>Enhanced Intel SpeedStep Technology Select Lock (R/W0)</p> <p>When set, this bit causes the following bits to become read-only:</p> <ul style="list-style-type: none"> ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit), ▪ Enhanced Intel SpeedStep Technology Enable bit. <p>The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.</p>
		21		Reserved.
		22	Shared	<p>Limit CPUID Maxval (R/W)</p> <p>See Table 35-2.</p>
		23	Shared	<p>xTPR Message Disable (R/W)</p> <p>See Table 35-2.</p>
		33:24		Reserved.
		34	Unique	<p>XD Bit Disable (R/W)</p> <p>See Table 35-2.</p>
		36:35		Reserved.
		37	Unique	<p>DCU Prefetcher Disable (R/W)</p> <p>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.</p> <p>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.</p>
		38	Shared	<p>IDA Disable (R/W)</p> <p>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 clear (CPUID.06H: EAX[1]=0).</p> <p>When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of IDA is enabled.</p> <p>Note: the power-on default value is used by BIOS to detect hardware support of IDA. If power-on default value is 1, IDA is available in the processor. If power-on default value is 0, IDA is not available.</p>

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		39	Unique	IP Prefetcher Disable (R/W) When set to 1, The IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The IP prefetcher is an L1 data cache prefetcher. The IP prefetcher looks for sequential load history to determine whether to prefetch the next expected data into the L1 cache from memory or L2.
		63:40		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Unique	Last Branch Record Stack TOS (R) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) See Table 35-2
1DDH	477	MSR_LER_FROM_LIP	Unique	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Unique	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Unique	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Unique	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Unique	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Unique	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Unique	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Unique	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Unique	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Unique	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Unique	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Unique	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Unique	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Unique	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Unique	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Unique	See Table 35-2.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
20FH	527	IA32_MTRR_PHYSMASK7	Unique	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_00000	Unique	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_80000	Unique	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_A0000	Unique	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Unique	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Unique	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Unique	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Unique	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Unique	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Unique	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Unique	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Unique	See Table 35-2.
277H	631	IA32_PAT	Unique	See Table 35-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	Default Memory Types (R/W) See Table 35-2.
309H	777	IA32_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W) See Table 35-2.
309H	777	MSR_PERF_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W)
30AH	778	IA32_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W) See Table 35-2.
30AH	778	MSR_PERF_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W)
30BH	779	IA32_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W) See Table 35-2.
30BH	779	MSR_PERF_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W)
345H	837	IA32_PERF_CAPABILITIES	Unique	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
345H	837	MSR_PERF_CAPABILITIES	Unique	RO. This applies to processors that do not support architectural perfmon version 2.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		5:0		LBR Format. See Table 35-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 35-2.
		63:8		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W) See Table 35-2.
38DH	909	MSR_PERF_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W)
38EH	910	IA32_PERF_GLOBAL_STAUS	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STAUS	Unique	See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	MSR_PERF_GLOBAL_CTRL	Unique	See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Unique	See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBES_ENABLE	Unique	See Table 35-2. See Section 18.4.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MCO_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	Unique	See Section 15.3.2.3, "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.
404H	1028	IA32_MC1_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the 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.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
408H	1032	IA32_MC2_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the 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.
40CH	1036	MSR_MC4_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC4_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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.
410H	1040	MSR_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC3_STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the 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.
413H	1043	MSR_MC3_MISC	Unique	
414H	1044	MSR_MC5_CTL	Unique	
415H	1045	MSR_MC5_STATUS	Unique	
416H	1046	MSR_MC5_ADDR	Unique	
417H	1047	MSR_MC5_MISC	Unique	
419H	1045	MSR_MC6_STATUS	Unique	Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 15.3.2.2, "IA32_MCi_STATUS MSRS." and Chapter 23.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 35-2. See Appendix A.1, "Basic VMX Information."

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
481H	1153	IA32_VMX_PINBASED_CTL5	Unique	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL5	Unique	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL5	Unique	Capability Reporting Register of VM-exit Controls (R/O) See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL5	Unique	Capability Reporting Register of VM-entry Controls (R/O) See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Unique	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Unique	Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 35-2. See Section 18.11.4, “Debug Store (DS) Mechanism.”
107CC H		MSR_EMON_L3_CTR_CTL0	Unique	GBUSQ Event Control/Counter Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CD H		MSR_EMON_L3_CTR_CTL1	Unique	GBUSQ Event Control/Counter Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CE H		MSR_EMON_L3_CTR_CTL2	Unique	GSNPQ Event Control/Counter Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CF H		MSR_EMON_L3_CTR_CTL3	Unique	GSNPQ Event Control/Counter Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D0 H		MSR_EMON_L3_CTR_CTL4	Unique	FSB Event Control/Counter Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D1 H		MSR_EMON_L3_CTR_CTL5	Unique	FSB Event Control/Counter Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D2 H		MSR_EMON_L3_CTR_CTL6	Unique	FSB Event Control/Counter Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D3 H		MSR_EMON_L3_CTR_CTL7	Unique	FSB Event Control/Counter Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D8 H		MSR_EMON_L3_GL_CTL	Unique	L3/FSB Common Control Register (R/W) Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
C000_ 0080H		IA32_EFER	Unique	Extended Feature Enables See Table 35-2.
C000_ 0081H		IA32_STAR	Unique	System Call Target Address (R/W) See Table 35-2.
C000_ 0082H		IA32_LSTAR	Unique	IA-32e Mode System Call Target Address (R/W) See Table 35-2.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
C000_0084H		IA32_FMASK	Unique	System Call Flag Mask (R/W) See Table 35-2.
C000_0100H		IA32_FS_BASE	Unique	Map of BASE Address of FS (R/W) See Table 35-2.
C000_0101H		IA32_GS_BASE	Unique	Map of BASE Address of GS (R/W) See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Unique	Swap Target of BASE Address of GS (R/W) See Table 35-2.

...

35.3 MSRS IN THE INTEL® ATOM™ PROCESSOR FAMILY

Table 35-4 lists model-specific registers (MSRs) for Intel Atom processor family, architectural MSR addresses are also included in Table 35-4. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1CH, see Table 35-1.

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 35-4 MSRs in Intel® Atom™ Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 35.14, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Shared	See Section 35.14, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, “Monitor/Mwait Address Range Determination.” and Table 35-2
10H	16	IA32_TIME_STAMP_COUNTER	Shared	See Section 17.13, “Time-Stamp Counter,” and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 35-2.
17H	23	MSR_PLATFORM_ID	Shared	Model Specific Platform ID (R)
		7:0		Reserved.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		12:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		63:13		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved.
		1		Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		2		Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		3		AERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		4		BERR# Enable for initiator bus requests (R/W) 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved.
		6		Reserved.
		7		BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved.
		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.		

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		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 OOB.
		19: 18		Reserved.
		21: 20		Symmetric Arbitration ID (R/O) Always OOB.
		26:22		Integer Bus Frequency Ratio (R/O)
3AH	58	IA32_FEATURE_CONTROL	Unique	Control Features in Intel 64Processor (R/W) See Table 35-2.
40H	64	MSR_ LASTBRANCH_0_FROM_IP	Unique	Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction for one of the last eight branches, exceptions, or interrupts taken by the processor. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.11, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_ LASTBRANCH_1_FROM_IP	Unique	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_ LASTBRANCH_2_FROM_IP	Unique	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_ LASTBRANCH_3_FROM_IP	Unique	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_ LASTBRANCH_4_FROM_IP	Unique	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_ LASTBRANCH_5_FROM_IP	Unique	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_ LASTBRANCH_6_FROM_IP	Unique	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_ LASTBRANCH_7_FROM_IP	Unique	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_ LASTBRANCH_0_TO_IP	Unique	Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last eight branches, exceptions, or interrupts taken by the processor.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Unique	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Unique	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Unique	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
67H	103	MSR_LASTBRANCH_7_TO_IP	Unique	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	BIOS Update Trigger Register (W) See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (RO) See Table 35-2.
C1H	193	IA32_PMC0	Unique	Performance counter register See Table 35-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 35-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed(RO) This field indicates the intended scaleable bus clock speed for processors based on Intel Atom microarchitecture:
		2:0		<ul style="list-style-type: none"> ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (RW) See Table 35-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (RW) See Table 35-2.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
FEH	254	IA32_MTRRCAP	Shared	Memory Type Range Register (R) See Table 35-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		L2 Hardware Enabled (RO) 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		L2 Enabled. (R/W) 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.
		23		L2 Not Present (RO) 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 35-2.
17AH	378	IA32_MCG_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.
186H	390	IA32_PERFEVTSELO	Unique	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 35-2.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
198H	408	IA32_PERF_STATUS	Shared	See Table 35-2.
198H	408	MSR_PERF_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.
199H	409	IA32_PERF_CTL	Unique	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 35-2.
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 35-2.
19DH	413	MSR_THERM2_CTL	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.
1A0	416	IA32_MISC_ENABLE	Unique	Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0		Fast-Strings Enable See Table 35-2.
		2:1		Reserved.
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 35-2.
		6:4		Reserved.
		7	Shared	Performance Monitoring Available (R) See Table 35-2.
		8		Reserved.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		9		Reserved.
		10	Shared	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	Branch Trace Storage Unavailable (RO) See Table 35-2.
		12	Shared	Precise Event Based Sampling Unavailable (RO) See Table 35-2.
		13	Shared	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved.
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 35-2.
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 35-2.
		19		Reserved.
		20	Shared	Enhanced Intel SpeedStep Technology Select Lock (R/W0) When set, this bit causes the following bits to become read-only: <ul style="list-style-type: none"> ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit), ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved.
		22	Unique	Limit CPUID Maxval (R/W) See Table 35-2.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		23	Shared	xTPR Message Disable (R/W) See Table 35-2.
		33:24		Reserved.
		34	Unique	XD Bit Disable (R/W) See Table 35-2.
		63:35		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Unique	Last Branch Record Stack TOS (R) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) See Table 35-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Shared	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Shared	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Shared	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Shared	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Shared	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Shared	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Shared	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Shared	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Shared	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Shared	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Shared	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Shared	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Shared	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Shared	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Shared	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Shared	See Table 35-2.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
250H	592	IA32_MTRR_FIX64K_00000	Shared	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_80000	Shared	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_A0000	Shared	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Shared	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Shared	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Shared	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Shared	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Shared	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Shared	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Shared	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Shared	See Table 35-2.
277H	631	IA32_PAT	Unique	See Table 35-2.
309H	777	IA32_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W) See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W) See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W) See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Shared	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W) See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_STAUS	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Unique	See Table 35-2. See Section 18.4.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MCO_CTL	Shared	See Section 15.3.2.1, "IA32_MCI_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Shared	See Section 15.3.2.2, "IA32_MCI_STATUS MSRs."

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
402H	1026	IA32_MCO_ADDR	Shared	See Section 15.3.2.3, "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.
404H	1028	IA32_MC1_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
408H	1032	IA32_MC2_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the 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.
40CH	1036	MSR_MC3_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC3_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	MSR_MC3_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the 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.
410H	1040	MSR_MC4_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC4_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	MSR_MC4_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Unique	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 35-2. See Appendix A.3, "VM-Execution Controls."

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
482H	1154	IA32_VMX_PROCBASED_CTL5	Unique	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL5	Unique	Capability Reporting Register of VM-exit Controls (R/O) See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL5	Unique	Capability Reporting Register of VM-entry Controls (R/O) See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Unique	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Unique	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Unique	Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 35-2. See Section 18.11.4, "Debug Store (DS) Mechanism."
C000_0080H		IA32_EFER	Unique	Extended Feature Enables See Table 35-2.
C000_0081H		IA32_STAR	Unique	System Call Target Address (R/W) See Table 35-2.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
C000_0082H		IA32_LSTAR	Unique	IA-32e Mode System Call Target Address (R/W) See Table 35-2.
C000_0084H		IA32_FMASK	Unique	System Call Flag Mask (R/W) See Table 35-2.
C000_0100H		IA32_FS_BASE	Unique	Map of BASE Address of FS (R/W) See Table 35-2.
C000_0101H		IA32_GS_BASE	Unique	Map of BASE Address of GS (R/W) See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Unique	Swap Target of BASE Address of GS (R/W) See Table 35-2.

Table 35-5 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor with the CPUID signature with DisplayFamily_DisplayModel of 06_27H.

Table 35-5 MSRs Supported by Intel® Atom™ Processors with CPUID Signature 06_27H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3F8H	1016	MSR_PKG_C2_RESIDENCY	Package	Package C2 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States
		63:0	Package	Package C2 Residency Counter. (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.
3F9H	1017	MSR_PKG_C4_RESIDENCY	Package	Package C4 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States
		63:0	Package	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.
3FAH	1018	MSR_PKG_C4_RESIDENCY	Package	Package C6 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States
		63:0	Package	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.

...

35.4 MSRS IN THE INTEL® MICROARCHITECTURE CODE NAME NEHALEM

Table 35-6 lists model-specific registers (MSRs) that are common for Intel® microarchitecture code name Nehalem. These include Intel Core i7 and i5 processor family. Architectural MSR addresses are also included in Table 35-6. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1AH, 06_1EH, 06_1FH, 06_2EH, see Table 35-1. Additional MSRs specific to 06_1AH, 06_1EH, 06_1FH are listed in Table 35-7. 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 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 35.14, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Thread	See Section 35.14, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, “Monitor/Mwait Address Range Determination,” and Table 35-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.13, “Time-Stamp Counter,” and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 35-2.
17H	23	MSR_PLATFORM_ID	Package	Model Specific Platform ID (R)
		49:0		Reserved.
		52:50		See Table 35-2.
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, “Local APIC Status and Location,” and Table 35-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O) Running count of SMI events since last RESET.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64Processor (R/W) See Table 35-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 35-2.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
8BH	139	IA32_BIOS_SIGN_ID	Thread	BIOS Update Signature ID (R0) See Table 35-2.
C1H	193	IA32_PMC0	Thread	Performance Counter Register See Table 35-2.
C2H	194	IA32_PMC1	Thread	Performance Counter Register See Table 35-2.
C3H	195	IA32_PMC2	Thread	Performance Counter Register See Table 35-2.
C4H	196	IA32_PMC3	Thread	Performance Counter Register See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	see http://biosbits.org .
		7:0		Reserved.
		15:8	Package	Maximum Non-Turbo Ratio (R/O) The is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.
		27:16		Reserved.
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	Programmable TDC-TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDC/TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.
		39:30		Reserved.
		47:40	Package	Maximum Efficiency Ratio (R/O) The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 133.33MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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, lock 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.
		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.
		63:27		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (RW) See Table 35-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (RW) See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_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.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
186H	390	IA32_PERFEVTSELO	Thread	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 35-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 35-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 35-2.
198H	408	IA32_PERF_STATUS	Core	See Table 35-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
199H	409	IA32_PERF_CTL	Thread	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		0		Reserved.
		3:1		On demand Clock Modulation Duty Cycle (R/W)
		4		On demand Clock Modulation Enable (R/W)
		63:5		Reserved.
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 35-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 35-2.
1A0	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Thread	Fast-Strings Enable See Table 35-2.
		2:1		Reserved.
		3	Thread	Automatic Thermal Control Circuit Enable (R/W) See Table 35-2.
		6:4		Reserved.
		7	Thread	Performance Monitoring Available (R) See Table 35-2.
		10:8		Reserved.
		11	Thread	Branch Trace Storage Unavailable (RO) See Table 35-2.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		12	Thread	Precise Event Based Sampling Unavailable (RO) See Table 35-2.
		15:13		Reserved.
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 35-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 35-2.
		21:19		Reserved.
		22	Thread	Limit CPUID Maxval (R/W) See Table 35-2.
		23	Thread	xTPR Message Disable (R/W) See Table 35-2.
		33:24		Reserved.
		34	Thread	XD Bit Disable (R/W) See Table 35-2.
		37:35		Reserved.
		38	Package	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. Note: the power-on default value is used by BIOS to detect hardware support of turbo mode. If power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.
1A2H	418	MSR_TEMPERATURE_TARGET	Thread	
		15:0		Reserved.
		23:16		Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		See http://biosbits.org .

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0	Package	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of EIST request from processor cores; When 1, disables hardware coordination of EIST requests.
		1	Thread	Energy/Performance Bias Enable (R/W) This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.(EAX=06h):ECX[3].
		63:2		Reserved.
1ADH	428	MSR_TURBO_POWER_CURRENT_LIMIT		See http://biosbits.org .
		14:0	Package	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.
		15	Package	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active, and a value = 1 indicates active.
		30:16	Package	TDC Limit (R/W) TDC limit in 1/8 Amp granularity.
		31	Package	TDC Limit Override Enable (R/W) A value = 0 indicates override is not active, and a value = 1 indicates active.
		63:32		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 17.6.2, "Filtering of Last Branch Records."

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 35-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See http://biosbits.org .
		0		Reserved.
		1	Package	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		63:2		Reserved.
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 35-2.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 35-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 35-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 35-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 35-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 35-2.
277H	631	IA32_PAT	Thread	See Table 35-2.
280H	640	IA32_MC0_CTL2	Package	See Table 35-2.
281H	641	IA32_MC1_CTL2	Package	See Table 35-2.
282H	642	IA32_MC2_CTL2	Core	See Table 35-2.
283H	643	IA32_MC3_CTL2	Core	See Table 35-2.
284H	644	IA32_MC4_CTL2	Core	See Table 35-2.
285H	645	IA32_MC5_CTL2	Core	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	Default Memory Types (R/W) See Table 35-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format. See Table 35-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 35-2.
		11:8		PEBS_REC_FORMAT. See Table 35-2.
		12		SMM_FREEZE. See Table 35-2.
63:13		Reserved.		
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_STAUS	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STAUS	Thread	(RO)
		61		UNC_Ovf Uncore overflowed if 1.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Thread	(R/W)
		61		CLR_UNC_Ovf Set 1 to clear UNC_Ovf.
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.6.1.1, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		63:36		Reserved.
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 18.6.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
400H	1024	IA32_MCO_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	Package	See Section 15.3.2.3, "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.
403H	1027	MSR_MCO_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the 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.
407H	1031	MSR_MC1_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the 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.
40BH	1035	MSR_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	MSR_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
40FH	1039	MSR_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
410H	1040	MSR_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	MSR_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the 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.
413H	1043	MSR_MC4_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
414H	1044	MSR_MC5_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	MSR_MC5_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
416H	1046	MSR_MC5_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	MSR_MC5_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	MSR_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 16.
41AH	1050	MSR_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	MSR_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	MSR_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 16.
41EH	1054	MSR_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	MSR_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	MSR_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 16.
422H	1058	MSR_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	MSR_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
480H	1152	IA32_VMX_BASIC	Thread	Reporting Register of Basic VMX Capabilities (R/O) See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Thread	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 35-2. See Appendix A.3, "VM-Execution Controls."

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
482H	1154	IA32_VMX_PROCBASED_CTL5	Thread	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL5	Thread	Capability Reporting Register of VM-exit Controls (R/O) See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL5	Thread	Capability Reporting Register of VM-entry Controls (R/O) See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	Capability Reporting Register of VMCS Field Enumeration (R/O). See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Thread	Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 35-2. See Section 18.11.4, "Debug Store (DS) Mechanism."

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.6.1, "LBR Stack."
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID register (R/O) See x2APIC Specification.
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service register bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode register bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode register bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request register bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request register bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request register bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request register bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request register bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request register bits [255:224] (R/O)

Table 35-6 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI register (W/O)
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 35-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 35-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 35-2.
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 35-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 35-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 35-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature. (R/W) See Table 35-2 and Section 17.13.2, "IA32_TSC_AUX Register and RDTSCP Support."

...

35.7 MSRS IN INTEL® PROCESSOR FAMILY (BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE)

Table 35-11 lists model-specific registers (MSRs) that are common to Intel® processor family based on Intel® microarchitecture (Sandy Bridge). All architectural MSRs listed in Table 35-2 are supported. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2AH, 06_2DH, see Table 35-1. Additional MSRs specific to 06_2AH are listed in Table 35-12.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 35.14, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 35.14, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 35-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.13, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 35-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O) Count SMIs.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 35-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	BIOS Update Signature ID (RO) See Table 35-2.
C1H	193	IA32_PMC0	Thread	Performance Counter Register See Table 35-2.
C2H	194	IA32_PMC1	Thread	Performance Counter Register See Table 35-2.
C3H	195	IA32_PMC2	Thread	Performance Counter Register See Table 35-2.
C4H	196	IA32_PMC3	Thread	Performance Counter Register See Table 35-2.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C5H	197	IA32_PMC4	Core	Performance Counter Register See Table 35-2.
C6H	198	IA32_PMC5	Core	Performance Counter Register See Table 35-2.
C7H	199	IA32_PMC6	Core	Performance Counter Register See Table 35-2.
C8H	200	IA32_PMC7	Core	Performance Counter Register See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org .
		7:0		Reserved.
		15:8	Package	Maximum Non-Turbo Ratio (R/O) The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved.
		47:40	Package	Maximum Efficiency Ratio (R/O) The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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/W0) When set, lock bits 15:0 of this register until next reset.
		24:16		Reserved.
		25		C3 state auto demotion enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		C1 state auto demotion enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		Enable C3 undemotion (R/W) When set, enables undemotion from demoted C3.
		28		Enable C1 undemotion (R/W) When set, enables undemotion from demoted C1.
		63:29		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (RW) See Table 35-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (RW) See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_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.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
186H	390	IA32_PERFEVTSELO	Thread	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 35-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 35-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 35-2.
18AH	394	IA32_PERFEVTSEL4	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18BH	395	IA32_PERFEVTSEL5	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18CH	396	IA32_PERFEVTSEL6	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18DH	397	IA32_PERFEVTSEL7	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
198H	408	IA32_PERF_STATUS	Package	See Table 35-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
198H	408	MSR_PERF_STATUS	Package	
		47:32		Core Voltage (R/O) P-state core voltage can be computed by $MSR_PERF_STATUS[37:32] * (\text{float}) 1/(2^{13})$.
199H	409	IA32_PERF_CTL	Thread	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 35-2 IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		3:0		On demand Clock Modulation Duty Cycle (R/W) In 6.25% increment
		4		On demand Clock Modulation Enable (R/W)
		63:5		Reserved.
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 35-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 35-2.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A0	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Thread	Fast-Strings Enable See Table 35-2
		6:1		Reserved.
		7	Thread	Performance Monitoring Available (R) See Table 35-2.
		10:8		Reserved.
		11	Thread	Branch Trace Storage Unavailable (RO) See Table 35-2.
		12	Thread	Precise Event Based Sampling Unavailable (RO) See Table 35-2.
		15:13		Reserved.
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 35-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 35-2.
		21:19		Reserved.
		22	Thread	Limit CPUID Maxval (R/W) See Table 35-2.
		23	Thread	xTPR Message Disable (R/W) See Table 35-2.
		33:24		Reserved.
		34	Thread	XD Bit Disable (R/W) See Table 35-2.
		37:35		Reserved.
		38	Package	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. Note: the power-on default value is used by BIOS to detect hardware support of turbo mode. If power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
63:39		Reserved.		

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A2H	418	MSR_TEMPERATURE_TARGET	Unique	
		15:0		Reserved.
		23:16		Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1A7H	422	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		See http://biosbits.org .
1ADH	428	MSR_TURBO_PWR_CURRENT_LIMIT		See http://biosbits.org .
1BOH	432	IA32_ENERGY_PERF_BIAS	Package	See Table 35-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 35-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 35-2.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W) See Section 17.6.2, "Filtering of Last Branch Records."
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 35-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1FCH	508	MSR_POWER_CTL	Core	See http://biosbits.org .
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 35-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 35-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 35-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 35-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 35-2.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 35-2.
277H	631	IA32_PAT	Thread	See Table 35-2.
280H	640	IA32_MC0_CTL2	Core	See Table 35-2.
281H	641	IA32_MC1_CTL2	Core	See Table 35-2.
282H	642	IA32_MC2_CTL2	Core	See Table 35-2.
283H	643	IA32_MC3_CTL2	Core	See Table 35-2.
284H	644	MSR_MC4_CTL2	Package	Always 0 (CMCI not supported).
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	Default Memory Types (R/W) See Table 35-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format. See Table 35-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 35-2.
		11:8		PEBS_REC_FORMAT. See Table 35-2.
		12		SMM_FREEZE. See Table 35-2.
63:13		Reserved.		
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_STAUS	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.6.1.1, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		63:36		Reserved.
3F6H	1014	MSR_PEBS_LD_LAT	Thread	see See Section 18.6.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
3FEH	1022	MSR_CORE_C7_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C7 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
402H	1026	IA32_MC0_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
403H	1027	IA32_MC0_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
406H	1030	IA32_MC1_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
407H	1031	IA32_MC1_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40BH	1035	IA32_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
410H	1040	MSR_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
		0		PCU Hardware Error (R/W) When set, enables signaling of PCU hardware detected errors.
		1		PCU Controller Error (R/W) When set, enables signaling of PCU controller detected errors
		2		PCU Firmware Error (R/W) When set, enables signaling of PCU firmware detected errors
		63:2		Reserved.
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
480H	1152	IA32_VMX_BASIC	Thread	Reporting Register of Basic VMX Capabilities (R/O) See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Thread	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Thread	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Thread	Capability Reporting Register of VM-exit Controls (R/O) See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Thread	Capability Reporting Register of VM-entry Controls (R/O) See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL2	Thread	Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
4C1H	1217	IA32_A_PMC0	Thread	See Table 35-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 35-2.
4C3H	1219	IA32_A_PMC2	Thread	See Table 35-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 35-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 35-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 35-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 35-2.
C8H	200	IA32_A_PMC7	Core	See Table 35-2.
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 35-2. See Section 18.11.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.7.1, "RAPL Interfaces."
60AH	1546	MSR_PKGC3_IRTL	Package	Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt response time limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved.
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
60BH	1547	MSR_PKGC6_IRTL	Package	<p>Package C6 Interrupt Response Limit (R/W)</p> <p>This MSR defines the budget allocated for the package to exit from C6 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.</p> <p>Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.</p>
		9:0		<p>Interrupt response time limit (R/W)</p> <p>Specifies the limit that should be used to decide if the package should be put into a package C6 state.</p>
		12:10		<p>Time Unit (R/W)</p> <p>Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported:</p> <p>000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns</p>
		14:13		Reserved.
		15		<p>Valid (R/W)</p> <p>Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.</p>
		63:16		Reserved.
60CH	1548	MSR_PKGC7_IRTL	Package	<p>Package C7 Interrupt Response Limit (R/W)</p> <p>This MSR defines the budget allocated for the package to exit from 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.</p> <p>Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.</p>
		9:0		<p>Interrupt response time limit (R/W)</p> <p>Specifies the limit that should be used to decide if the package should be put into a package C7 state.</p>

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C2 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_RAPL_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.7.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERY_STATUS	Package	PKG Energy Status (R/O) See Section 14.7.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W) See Section 14.7.3, "Package RAPL Domain."
638H	1592	MSR_PPO_POWER_LIMIT	Package	PPO RAPL Power Limit Control (R/W) See Section 14.7.4, "PPO/PP1 RAPL Domains."
639H	1593	MSR_PPO_ENERY_STATUS	Package	PPO Energy Status (R/O) See Section 14.7.4, "PPO/PP1 RAPL Domains."
63AH	1594	MSR_PPO_POLICY	Package	PPO Balance Policy (R/W) See Section 14.7.4, "PPO/PP1 RAPL Domains."
63BH	1595	MSR_PPO_PERF_STATUS	Package	PPO Performance Throttling Status (R/O) See Section 14.7.4, "PPO/PP1 RAPL Domains."

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.6.1, "LBR Stack."
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

**Table 35-11 MSRs Supported by Intel® Processors
Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6E0H	1760	IA32_TSC_DEADLINE	Thread	See Table 35-2.
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 35-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 35-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 35-2.
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 35-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 35-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 35-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 35-2 and Section 17.13.2, "IA32_TSC_AUX Register and RDTSCP Support."

...

35.7.2 MSRs In Intel® Xeon® Processor E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 35-13 lists selected model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family (based on Intel® microarchitecture code name Sandy Bridge). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH, see Table 35-1.

Table 35-13 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		0		Reserved
		1		MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
414H	1044	MSR_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	MSR_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
416H	1046	MSR_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	MSR_MC5_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	MSR_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41AH	1050	MSR_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	MSR_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	MSR_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41EH	1054	MSR_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	MSR_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."

Table 35-13 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	MSR_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
422H	1058	MSR_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	MSR_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
424H	1060	MSR_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	MSR_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
426H	1062	MSR_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	MSR_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	MSR_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	MSR_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42AH	1066	MSR_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	MSR_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	MSR_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	MSR_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42EH	1070	MSR_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	MSR_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	MSR_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	MSR_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
432H	1074	MSR_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	MSR_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	MSR_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	MSR_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
436H	1078	MSR_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	MSR_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	MSR_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	MSR_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43AH	1082	MSR_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	MSR_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	MSR_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	MSR_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43EH	1086	MSR_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	MSR_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	MSR_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 35-13 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
441H	1089	MSR_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
442H	1090	MSR_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRS."
443H	1091	MSR_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRS."
444H	1092	MSR_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCI_CTL MSRS."
445H	1093	MSR_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
446H	1094	MSR_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRS."
447H	1095	MSR_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRS."
448H	1096	MSR_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCI_CTL MSRS."
449H	1097	MSR_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
44AH	1098	MSR_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRS."
44BH	1099	MSR_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRS."
44CH	1100	MSR_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCI_CTL MSRS."
44DH	1101	MSR_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
44EH	1102	MSR_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRS."
44FH	1103	MSR_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRS."
613H	1555	MSR_RAPL_PERF_STATUS	Package	RAPL Perf Status (R/O)
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.7.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.7.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.7.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.7.5, "DRAM RAPL Domain."

...

35.9 MSRS IN THE NEXT GENERATION INTEL® CORE™ PROCESSORS (BASED ON INTEL® MICROARCHITECTURE CODE NAME HASWELL)

The Next Generation Intel® Core™ Processor Family (based on Intel® microarchitecture code name Haswell) supports the MSR interfaces listed in Table 35-11, Table 35-12, Table 35-14, and Table 35-15.

Table 35-15 Additional MSRs Supported by Next Generation Intel® Core™ Processors (Based on Intel® Microarchitecture Code Name Haswell)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3BH	59	IA32_TSC_ADJUST	THREAD	Per-Logical-Processor TSC ADJUST (R/W) See Table 35-2.

35.10 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 35-15 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 35-1.

- 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 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
0H	0	IA32_P5_MC_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 35.14, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3, 4, 6	Shared	See Section 35.14, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_LINE_SIZE	3, 4, 6	Shared	See Section 8.10.5, “Monitor/Mwait Address Range Determination.”
10H	16	IA32_TIME_STAMP_COUNTER	0, 1, 2, 3, 4, 6	Unique	Time Stamp Counter See Table 35-2.
					On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.
17H	23	IA32_PLATFORM_ID	0, 1, 2, 3, 4, 6	Shared	Platform ID (R) See Table 35-2. The operating system can use this MSR to determine “slot” information for the processor and the proper microcode update to load.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
1BH	27	IA32_APIC_BASE	0, 1, 2, 3, 4, 6	Unique	APIC Location and Status (R/W) See Table 35-2. See Section 10.4.4, "Local APIC Status and Location."
2AH	42	MSR_EBC_HARD_POWERON	0, 1, 2, 3, 4, 6	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0			Output Tri-state Enabled (R) Indicates whether tri-state output is enabled (1) or disabled (0) as set by the strapping of SMI#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		1			Execute BIST (R) Indicates whether the execution of the BIST is enabled (1) or disabled (0) as set by the strapping of INIT#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		2			In Order Queue Depth (R) Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		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 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		7			Bus Park Disable (R) Indicates whether bus park is enabled (0) or disabled (1) as set by the strapping of A15#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		11:8			Reserved.
		13:12			Agent ID (R) Contains the logical agent ID value as set by the strapping of BR[3:0]. The logical ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		63:14			Reserved.
2BH	43	MSR_EBC_SOFT_POWERON	0, 1, 2, 3, 4, 6	Shared	Processor Soft Power-On Configuration (R/W) Enables and disables processor features.
		0			RCNT/SCNT On Request Encoding Enable (R/W) Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).
		1			Data Error Checking Disable (R/W) Set to disable system data bus parity checking; clear to enable parity checking.
		2			Response Error Checking Disable (R/W) Set to disable (default); clear to enable.
		3			Address/Request Error Checking Disable (R/W) Set to disable (default); clear to enable.
		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.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
2CH	44	MSR_EBC_FREQUENCY_ID	2,3, 4, 6	Shared	<p>Processor Frequency Configuration</p> <p>The bit field layout of this MSR varies according to the MODEL value in the CPUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2.</p> <p>(R) The field Indicates the current processor frequency configuration.</p>
		15:0			Reserved.
		18:16			<p>Scalable Bus Speed (R/W)</p> <p>Indicates the intended scalable bus speed:</p> <p><u>Encoding Scalable Bus Speed</u></p> <p>000B 100 MHz (Model 2)</p> <p>000B 266 MHz (Model 3 or 4)</p> <p>001B 133 MHz</p> <p>010B 200 MHz</p> <p>011B 166 MHz</p> <p>100B 333 MHz (Model 6)</p> <p>133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.</p> <p>166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.</p>
					<p>266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4.</p> <p>333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6.</p> <p>All other values are reserved.</p>
		23:19			Reserved.
		31:24			<p>Core Clock Frequency to System Bus Frequency Ratio (R)</p> <p>The processor core clock frequency to system bus frequency ratio observed at the de-assertion of the reset pin.</p>
		63:25			Reserved.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
2CH	44	MSR_EBC_FREQUENCY_ID	0, 1	Shared	Processor Frequency Configuration (R) The bit field layout of this MSR varies according to the MODEL value of the CPUID version information. This bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding less than 2. Indicates current processor frequency configuration.
		20:0			Reserved.
		23:21			Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz All others values reserved.
		63:24			Reserved.
3AH	58	IA32_FEATURE_CONTROL	3, 4, 6	Unique	Control Features in IA-32 Processor (R/W) See Table 35-2 (If CPUID.01H:ECX.[bit 5])
79H	121	IA32_BIOS_UPDT_TRIG	0, 1, 2, 3, 4, 6	Shared	BIOS Update Trigger Register (W) See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3, 4, 6	Unique	BIOS Update Signature ID (R/W) See Table 35-2.
9BH	155	IA32_SMM_MONITOR_CTL	3, 4, 6	Unique	SMM Monitor Configuration (R/W) See Table 35-2.
FEH	254	IA32_MTRRCAP	0, 1, 2, 3, 4, 6	Unique	MTRR Information See Section 11.11.1, "MTRR Feature Identification."
174H	372	IA32_SYSENTER_CS	0, 1, 2, 3, 4, 6	Unique	CS register target for CPL 0 code (R/W) See Table 35-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
175H	373	IA32_SYSENTER_ESP	0, 1, 2, 3, 4, 6	Unique	Stack pointer for CPL 0 stack (R/W) See Table 35-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3, 4, 6	Unique	CPL 0 code entry point (R/W) See Table 35-2. See Section 5.8.7, “Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions.”
179H	377	IA32_MCG_CAP	0, 1, 2, 3, 4, 6	Unique	Machine Check Capabilities (R) See Table 35-2. See Section 15.3.1.1, “IA32_MCG_CAP MSR.”
17AH	378	IA32_MCG_STATUS	0, 1, 2, 3, 4, 6	Unique	Machine Check Status. (R) See Table 35-2. See Section 15.3.1.2, “IA32_MCG_STATUS MSR.”
17BH	379	IA32_MCG_CTL			Machine Check Feature Enable (R/W) See Table 35-2. See Section 15.3.1.3, “IA32_MCG_CTL MSR.”
180H	384	MSR_MCG_RAX	0, 1, 2, 3, 4, 6	Unique	Machine Check EAX/RAX Save State See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
181H	385	MSR_MCG_RBX	0, 1, 2, 3, 4, 6	Unique	Machine Check EBX/RBX Save State See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
182H	386	MSR_MCG_RCX	0, 1, 2, 3, 4, 6	Unique	Machine Check ECX/RCX Save State See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
183H	387	MSR_MCG_RDX	0, 1, 2, 3, 4, 6	Unique	Machine Check EDX/RDX Save State See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
184H	388	MSR_MCG_RSI	0, 1, 2, 3, 4, 6	Unique	Machine Check ESI/RSI Save State See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		63:0			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.
185H	389	MSR_MCG_RDI	0, 1, 2, 3, 4, 6	Unique	Machine Check EDI/RDI Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
186H	390	MSR_MCG_RBP	0, 1, 2, 3, 4, 6	Unique	Machine Check EBP/RBP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
187H	391	MSR_MCG_RSP	0, 1, 2, 3, 4, 6	Unique	Machine Check ESP/RSP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
188H	392	MSR_MCG_RFLAGS	0, 1, 2, 3, 4, 6	Unique	Machine Check EFLAGS/RFLAG Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
189H	393	MSR_MCG_RIP	0, 1, 2, 3, 4, 6	Unique	Machine Check EIP/RIP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
18AH	394	MSR_MCG_MISC	0, 1, 2, 3, 4, 6	Unique	Machine Check Miscellaneous See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		0			DS When set, the bit indicates that a page assist or page fault occurred during DS normal operation. The processors response is to shut down. The bit is used as an aid for debugging DS handling code. It is the responsibility of the user (BIOS or operating system) to clear this bit for normal operation.
		63:1			Reserved.
18BH - 18FH	395	MSR_MCG_RESERVED1 - MSR_MCG_RESERVED5			Reserved.
190H	400	MSR_MCG_R8	0, 1, 2, 3, 4, 6	Unique	Machine Check R8 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
191H	401	MSR_MCG_R9	0, 1, 2, 3, 4, 6	Unique	Machine Check R9D/R9 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
192H	402	MSR_MCG_R10	0, 1, 2, 3, 4, 6	Unique	Machine Check R10 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
193H	403	MSR_MCG_R11	0, 1, 2, 3, 4, 6	Unique	Machine Check R11 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
194H	404	MSR_MCG_R12	0, 1, 2, 3, 4, 6	Unique	Machine Check R12 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
195H	405	MSR_MCG_R13	0, 1, 2, 3, 4, 6	Unique	Machine Check R13 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
196H	406	MSR_MCG_R14	0, 1, 2, 3, 4, 6	Unique	Machine Check R14 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
197H	407	MSR_MCG_R15	0, 1, 2, 3, 4, 6	Unique	Machine Check R15 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
198H	408	IA32_PERF_STATUS	3, 4, 6	Unique	See Table 35-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."
199H	409	IA32_PERF_CTL	3, 4, 6	Unique	See Table 35-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."
19AH	410	IA32_CLOCK_MODULATION	0, 1, 2, 3, 4, 6	Unique	Thermal Monitor Control (R/W) See Table 35-2. See Section 14.5.3, "Software Controlled Clock Modulation."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
19BH	411	IA32_THERM_INTERRUPT	0, 1, 2, 3, 4, 6	Unique	Thermal Interrupt Control (R/W) See Section 14.5.2, "Thermal Monitor," and see Table 35-2.
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3, 4, 6	Shared	Thermal Monitor Status (R/W) See Section 14.5.2, "Thermal Monitor," and see Table 35-2.
19DH	413	MSR_THERM2_CTL			Thermal Monitor 2 Control.
			3,	Shared	For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.
			4, 6	Shared	For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.
1A0H	416	IA32_MISC_ENABLE	0, 1, 2, 3, 4, 6	Shared	Enable Miscellaneous Processor Features (R/W)
		0			Fast-Strings Enable. See Table 35-2.
		1			Reserved.
		2			x87 FPU Fopcode Compatibility Mode Enable
		3			Thermal Monitor 1 Enable See Section 14.5.2, "Thermal Monitor," and see Table 35-2.
		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.		

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
					Note that the bit controls only the third-level cache; and only if overall caching is enabled through the CD flag of control register CRO, the page-level cache controls, and/or the MTRRs. See Section 11.5.4, "Disabling and Enabling the L3 Cache."
		7			Performance Monitoring Available (R) See Table 35-2.
		8			Suppress Lock Enable When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.
		9			Prefetch Queue Disable When set, disables the prefetch queue. When clear (default), enables the prefetch queue.
		10			FERR# Interrupt Reporting Enable (R/W) When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled. 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 35-2. When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.
		12			PEBS_UNAVAILABLE: Precise Event Based Sampling Unavailable (R) See Table 35-2. When set, the processor does not support precise event-based sampling (PEBS); when clear, PEBS is supported.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
		13	3		<p>TM2 Enable (R/W)</p> <p>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.</p> <p>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.</p>
					If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		17:14			Reserved.
		18	3, 4, 6		<p>ENABLE MONITOR FSM (R/W)</p> <p>See Table 35-2.</p>
		19			<p>Adjacent Cache Line Prefetch Disable (R/W)</p> <p>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.</p>
					<p>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.</p> <p>BIOS may contain a setup option that controls the setting of this bit.</p>
		21:20			Reserved.
		22	3, 4, 6		<p>Limit CPUID MAXVAL (R/W)</p> <p>See Table 35-2.</p> <p>Setting this can cause unexpected behavior to software that depends on the availability of CPUID leaves greater than 3.</p>
		23		Shared	<p>xTPR Message Disable (R/W)</p> <p>See Table 35-2.</p>

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		24			<p>L1 Data Cache Context Mode (R/W)</p> <p>When set, the L1 data cache is placed in shared mode; when clear (default), the cache is placed in adaptive mode. This bit is only enabled for IA-32 processors that support Intel Hyper-Threading Technology. See Section 11.5.6, "L1 Data Cache Context Mode."</p> <p>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.</p> <p>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].</p>
		33:25			Reserved.
		34		Unique	<p>XD Bit Disable (R/W)</p> <p>See Table 35-2.</p>
		63:35			Reserved.
1A1H	417	MSR_PLATFORM_BRV	3, 4, 6	Shared	Platform Feature Requirements (R)
		17:0			Reserved.
		18			<p>PLATFORM Requirements</p> <p>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.</p>
		63:19			Reserved.
1D7H	471	MSR_LER_FROM_LIP	0, 1, 2, 3, 4, 6	Unique	<p>Last Exception Record From Linear IP (R)</p> <p>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.</p> <p>See Section 17.9.3, "Last Exception Records."</p>
		31:0			<p>From Linear IP</p> <p>Linear address of the last branch instruction.</p>
		63:32			Reserved.
1D7H	471	63:0		Unique	<p>From Linear IP</p> <p>Linear address of the last branch instruction (If IA-32e mode is active).</p>

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
1D8H	472	MSR_LER_TO_LIP	0, 1, 2, 3, 4, 6	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.9.3, "Last Exception Records."
		31:0			From Linear IP Linear address of the target of the last branch instruction.
		63:32			Reserved.
1D8H	472	63:0		Unique	From Linear IP Linear address of the target of the last branch instruction (If IA-32e mode is active).
1D9H	473	MSR_DEBUGCTLA	0, 1, 2, 3, 4, 6	Unique	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 17.9.1, "MSR_DEBUGCTLA MSR."
1DAH	474	MSR_LASTBRANCH_TOS	0, 1, 2, 3, 4, 6	Unique	Last Branch Record Stack TOS (R) Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 17.9.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture"; and addresses 1DBH-1DEH and 680H-68FH.
1DBH	475	MSR_LASTBRANCH_0	0, 1, 2	Unique	Last Branch Record 0 (R/W) One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH.
					See Section 17.9, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
1DDH	477	MSR_LASTBRANCH_2	0, 1, 2	Unique	Last Branch Record 2 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DEH	478	MSR_LASTBRANCH_3	0, 1, 2	Unique	Last Branch Record 3 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
200H	512	IA32_MTRR_PHYSBASE0	0, 1, 2, 3, 4, 6	Shared	Variable Range Base MTRR See Section 11.11.2.3, "Variable Range MTRRs."
201H	513	IA32_MTRR_PHYSMASK0	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
202H	514	IA32_MTRR_PHYSBASE1	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
203H	515	IA32_MTRR_PHYSMASK1	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
204H	516	IA32_MTRR_PHYSBASE2	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
205H	517	IA32_MTRR_PHYSMASK2	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
206H	518	IA32_MTRR_PHYSBASE3	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
207H	519	IA32_MTRR_PHYSMASK3	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
208H	520	IA32_MTRR_PHYSBASE4	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
209H	521	IA32_MTRR_PHYSMASK4	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20AH	522	IA32_MTRR_PHYSBASE5	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20BH	523	IA32_MTRR_PHYSMASK5	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20CH	524	IA32_MTRR_PHYSBASE6	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20DH	525	IA32_MTRR_PHYSMASK6	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20EH	526	IA32_MTRR_PHYSBASE7	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
20FH	527	IA32_MTRR_PHYSMASK7	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
250H	592	IA32_MTRR_FIX64K_00000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
258H	600	IA32_MTRR_FIX16K_80000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
259H	601	IA32_MTRR_FIX16K_A0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
268H	616	IA32_MTRR_FIX4K_C0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
269H	617	IA32_MTRR_FIX4K_C8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26AH	618	IA32_MTRR_FIX4K_D0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26BH	619	IA32_MTRR_FIX4K_D8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26CH	620	IA32_MTRR_FIX4K_E0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26DH	621	IA32_MTRR_FIX4K_E8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26EH	622	IA32_MTRR_FIX4K_F0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26FH	623	IA32_MTRR_FIX4K_F8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
277H	631	IA32_PAT	0, 1, 2, 3, 4, 6	Unique	Page Attribute Table See Section 11.11.2.2, "Fixed Range MTRRs."
2FFH	767	IA32_MTRR_DEF_TYPE	0, 1, 2, 3, 4, 6	Shared	Default Memory Types (R/W) See Table 35-2. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
300H	768	MSR_BPU_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
307H	775	MSR_MS_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
308H	776	MSR_FLAME_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
309H	777	MSR_FLAME_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
30AH	778	MSR_FLAME_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
30BH	779	MSR_FLAME_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
30CH	780	MSR_IQ_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
30DH	781	MSR_IQ_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
30EH	782	MSR_IQ_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
30FH	783	MSR_IQ_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.2, "Performance Counters."
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
364H	868	MSR_MS_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
365H	869	MSR_MS_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
366H	870	MSR_MS_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
367H	871	MSR_MS_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
368H	872	MSR_FLAME_CCCRO	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
369H	873	MSR_FLAME_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
36AH	874	MSR_FLAME_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
36BH	875	MSR_FLAME_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
36CH	876	MSR_IQ_CCCRO	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.3, "CCCR MSRs."
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3A6H	934	MSR_FLAME_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
3A7H	935	MSR_FLAME_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3AEH	942	MSR_SAAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3AFH	943	MSR_SAAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	Shared	See Section 18.11.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BBH	955	MSR_IQ_ESCR1	0, 1, 2	Shared	See Section 18.11.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3C0H	960	MSR_MS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3C9H	969	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3FOH	1008	MSR_TC_PRECISE_EVENT	0, 1, 2, 3, 4, 6	Shared	See Section 18.11.1, "ESCR MSRs."
3F1H	1009	MSR_PEBS_ENABLE	0, 1, 2, 3, 4, 6	Shared	Precise Event-Based Sampling (PEBS) (R/W) Controls the enabling of precise event sampling and replay tagging.
		12:0			See Table 19-24.
		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 18.12.3, "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 18.12.3, "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.
3F2H	1010	MSR_PEBS_MATRIX_VERT	0, 1, 2, 3, 4, 6	Shared	See Table 19-24.
400H	1024	IA32_MCO_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
402H	1026	IA32_MCO_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "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.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
403H	1027	IA32_MCO_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MCO_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
406H	1030	IA32_MC1_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the 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.
407H	1031	IA32_MC1_MISC		Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC1_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40AH	1034	IA32_MC2_ADDR			See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the 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.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
40BH	1035	IA32_MC2_MISC			See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	IA32_MC3_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the 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.
40FH	1039	IA32_MC3_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC3_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	IA32_MC4_ADDR			See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the 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.
413H	1043	IA32_MC4_MISC			See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
					When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	3, 4, 6	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	3, 4, 6	Unique	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	3, 4, 6	Unique	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and see Table 35-2.
483H	1155	IA32_VMX_EXIT_CTL	3, 4, 6	Unique	Capability Reporting Register of VM-exit Controls (R/O) See Appendix A.4, "VM-Exit Controls," and see Table 35-2.
484H	1156	IA32_VMX_ENTRY_CTL	3, 4, 6	Unique	Capability Reporting Register of VM-entry Controls (R/O) See Appendix A.5, "VM-Entry Controls," and see Table 35-2.
485H	1157	IA32_VMX_MISC	3, 4, 6	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data," and see Table 35-2.
486H	1158	IA32_VMX_CR0_FIXED0	3, 4, 6	Unique	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 35-2.
487H	1159	IA32_VMX_CR0_FIXED1	3, 4, 6	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 35-2.
488H	1160	IA32_VMX_CR4_FIXED0	3, 4, 6	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 35-2.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
489H	1161	IA32_VMX_CR4_FIXED1	3, 4, 6	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, “VMX-Fixed Bits in CR4,” and see Table 35-2.
48AH	1162	IA32_VMX_VMCS_ENUM	3, 4, 6	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, “VMCS Enumeration,” and see Table 35-2.
48BH	1163	IA32_VMX_PROCBASED_CTLDS2	3, 4, 6	Unique	Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O) See Appendix A.3, “VM-Execution Controls,” and see Table 35-2.
600H	1536	IA32_DS_AREA	0, 1, 2, 3, 4, 6	Unique	DS Save Area (R/W) See Table 35-2. See Section 18.11.4, “Debug Store (DS) Mechanism.”
680H	1664	MSR_LASTBRANCH_0_FROM_IP	3, 4, 6	Unique	Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (680H-68FH). This part of the stack contains pointers to the source instruction for one of the last 16 branches, exceptions, or interrupts taken by the processor. The MSRs at 680H-68FH, 6C0H-6CfH are not available in processor releases before family 0FH, model 03H. These MSRs replace MSRs previously located at 1DBH-1DEH, which performed the same function for early releases. See Section 17.9, “Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture).”
681H	1665	MSR_LASTBRANCH_1_FROM_IP	3, 4, 6	Unique	Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 680H.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	3, 4, 6	Unique	Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 680H.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	3, 4, 6	Unique	Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 680H.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	3, 4, 6	Unique	Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 680H.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	3, 4, 6	Unique	Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 680H.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
686H	1670	MSR_LASTBRANCH_6_FROM_IP	3, 4, 6	Unique	Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 680H.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	3, 4, 6	Unique	Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 680H.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	3, 4, 6	Unique	Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 680H.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	3, 4, 6	Unique	Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 680H.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	3, 4, 6	Unique	Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 680H.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	3, 4, 6	Unique	Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 680H.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	3, 4, 6	Unique	Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 680H.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	3, 4, 6	Unique	Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 680H.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	3, 4, 6	Unique	Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 680H.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	3, 4, 6	Unique	Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 680H.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	3, 4, 6	Unique	Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 17.9, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)."
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	3, 4, 6	Unique	Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 6C0H.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	3, 4, 6	Unique	Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 6C0H.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	3, 4, 6	Unique	Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 6C0H.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	3, 4, 6	Unique	Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 6C0H.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	3, 4, 6	Unique	Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 6COH.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	3, 4, 6	Unique	Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 6COH.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	3, 4, 6	Unique	Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 6COH.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	3, 4, 6	Unique	Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 6COH.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	3, 4, 6	Unique	Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 6COH.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	3, 4, 6	Unique	Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 6COH.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	3, 4, 6	Unique	Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 6COH.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	3, 4, 6	Unique	Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 6COH.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	3, 4, 6	Unique	Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 6COH.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	3, 4, 6	Unique	Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 6COH.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	3, 4, 6	Unique	Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 6COH.
C000_0080H		IA32_EFER	3, 4, 6	Unique	Extended Feature Enables See Table 35-2.
C000_0081H		IA32_STAR	3, 4, 6	Unique	System Call Target Address (R/W) See Table 35-2.
C000_0082H		IA32_LSTAR	3, 4, 6	Unique	IA-32e Mode System Call Target Address (R/W) See Table 35-2.
C000_0084H		IA32_FMASK	3, 4, 6	Unique	System Call Flag Mask (R/W) See Table 35-2.
C000_0100H		IA32_FS_BASE	3, 4, 6	Unique	Map of BASE Address of FS (R/W) See Table 35-2.
C000_0101H		IA32_GS_BASE	3, 4, 6	Unique	Map of BASE Address of GS (R/W) See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	3, 4, 6	Unique	Swap Target of BASE Address of GS (R/W) See Table 35-2.

Table 35-15 MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				

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.

...

27. Updates to Appendix B, Volume 3C

Change bars show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

...

B.1.1 16-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-1 enumerates the 16-bit control fields.

Table B-1 Encoding for 16-Bit Control Fields (0000_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Virtual-processor identifier (VPID) ¹	000000000B	00000000H
Posted-interrupt notification vector ²	000000001B	00000002H

NOTES:

1. This field exists only on processors that support the 1-setting of the "enable VPID" VM-execution control.
2. This field exists only on processors that support the 1-setting of the "process posted interrupts" VM-execution control.

B.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-2 enumerates 16-bit guest-state fields.

Table B-2 Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES selector	000000000B	00000800H
Guest CS selector	000000001B	00000802H
Guest SS selector	000000010B	00000804H
Guest DS selector	000000011B	00000806H
Guest FS selector	000000100B	00000808H
Guest GS selector	000000101B	0000080AH
Guest LDTR selector	000000110B	0000080CH

Table B-2 Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest TR selector	000000111B	0000080EH
Guest interrupt status ¹	000001000B	00000810H

NOTES:

1. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.

B.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-4 enumerates the 64-bit control fields.

Table B-4 Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

Field Name	Index	Encoding
Address of I/O bitmap A (full)	000000000B	00002000H
Address of I/O bitmap A (high)		00002001H
Address of I/O bitmap B (full)	000000001B	00002002H
Address of I/O bitmap B (high)		00002003H
Address of MSR bitmaps (full) ¹	000000010B	00002004H
Address of MSR bitmaps (high) ¹		00002005H
VM-exit MSR-store address (full)	000000011B	00002006H
VM-exit MSR-store address (high)		00002007H
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)		00002009H
VM-entry MSR-load address (full)	000000101B	0000200AH
VM-entry MSR-load address (high)		0000200BH
Executive-VMCS pointer (full)	000000110B	0000200CH
Executive-VMCS pointer (high)		0000200DH
TSC offset (full)	000001000B	00002010H
TSC offset (high)		00002011H
Virtual-APIC address (full) ²	000001001B	00002012H
Virtual-APIC address (high) ²		00002013H
APIC-access address (full) ³	000001010B	00002014H
APIC-access address (high) ³		00002015H
Posted-interrupt descriptor address (full) ⁴	000001011B	00002016H
Posted-interrupt descriptor address (high) ⁴		00002017H
VM-function controls (full) ⁵	000001100B	00002018H
VM-function controls (high) ⁵		00002019H

Table B-4 Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
EPT pointer (EPTP; full) ⁶	000001101B	0000201AH
EPT pointer (EPTP; high) ⁶		0000201BH
EOI-exit bitmap 0 (EOI_EXIT0; full) ⁷	000001110B	0000201CH
EOI-exit bitmap 0 (EOI_EXIT0; high) ⁷		0000201DH
EOI-exit bitmap 1 (EOI_EXIT1; full) ⁷	000001111B	0000201EH
EOI-exit bitmap 1 (EOI_EXIT1; high) ⁷		0000201FH
EOI-exit bitmap 2 (EOI_EXIT2; full) ⁷	000010000B	00002020H
EOI-exit bitmap 2 (EOI_EXIT2; high) ⁷		00002021H
EOI-exit bitmap 3 (EOI_EXIT3; full) ⁷	000010001B	00002022H
EOI-exit bitmap 3 (EOI_EXIT3; high) ⁷		00002023H
EPTP-list address (full) ⁸	000010010B	00002024H
EPTP-list address (high) ⁸		00002025H

NOTES:

1. This field exists only on processors that support the 1-setting of the “use MSR bitmaps” VM-execution control.
2. This field exists only on processors that support either the 1-setting of the “use TPR shadow” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “enable VM functions” VM-execution control.
6. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
7. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
8. This field exists only on processors that support the 1-setting of the “EPTP switching” VM-function control.

B.2.2 64-Bit Read-Only Data Field

A value of 1 in bits 11: 10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9: 1. There is only one such 64-bit field as given in Table B-5. (As with other 64-bit fields, this one has two encodings.)

Table B-5 Encodings for 64-Bit Read-Only Data Field (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full) ¹	000000000B	00002400H
Guest-physical address (high) ¹		00002401H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

B.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-6 enumerates the 64-bit guest-state fields.

Table B-6 Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
VMCS link pointer (full)	000000000B	00002800H
VMCS link pointer (high)		00002801H
Guest IA32_DEBUGCTL (full)	000000001B	00002802H
Guest IA32_DEBUGCTL (high)		00002803H
Guest IA32_PAT (full) ¹	000000010B	00002804H
Guest IA32_PAT (high) ¹		00002805H
Guest IA32_EFER (full) ²	000000011B	00002806H
Guest IA32_EFER (high) ²		00002807H
Guest IA32_PERF_GLOBAL_CTRL (full) ³	000000100B	00002808H
Guest IA32_PERF_GLOBAL_CTRL (high) ³		00002809H
Guest PDPTE0 (full) ⁴	000000101B	0000280AH
Guest PDPTE0 (high) ⁴		0000280BH
Guest PDPTE1 (full) ⁴	000000110B	0000280CH
Guest PDPTE1 (high) ⁴		0000280DH
Guest PDPTE2 (full) ⁴	000000111B	0000280EH
Guest PDPTE2 (high) ⁴		0000280FH
Guest PDPTE3 (full) ⁴	000001000B	00002810H
Guest PDPTE3 (high) ⁴		00002811H

NOTES:

1. This field exists only on processors that support either the 1-setting of the "load IA32_PAT" VM-entry control or that of the "save IA32_PAT" VM-exit control.
2. This field exists only on processors that support either the 1-setting of the "load IA32_EFER" VM-entry control or that of the "save IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-entry control.
4. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

B.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-7 enumerates the 64-bit control fields.

Table B-7 Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (full) ¹	000000000B	00002C00H
Host IA32_PAT (high) ¹		00002C01H

Table B-7 Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Host IA32_EFER (full) ²	000000001B	00002C02H
Host IA32_EFER (high) ²		00002C03H
Host IA32_PERF_GLOBAL_CTRL (full) ³	000000010B	00002C04H
Host IA32_PERF_GLOBAL_CTRL (high) ³		00002C05H

NOTES:

1. This field exists only on processors that support the 1-setting of the "load IA32_PAT" VM-exit control.
2. This field exists only on processors that support the 1-setting of the "load IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-exit control.

...

28. Updates to Appendix C, Volume 3C

Change bars show changes to Appendix C of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

...

Table C-1 Basic Exit Reasons

Basic Exit Reason	Description
0	Exception or non-maskable interrupt (NMI). Either: 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1. 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1. This case includes executions of BOUND that cause #BR, executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UD2 (they cause #UD).
1	External interrupt. An external interrupt arrived and the "external-interrupt exiting" VM-execution control was 1.
2	Triple fault. The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap.
3	INIT signal. An INIT signal arrived
4	Start-up IPI (SIPI). A SIPI arrived while the logical processor was in the "wait-for-SIPI" state.
5	I/O system-management interrupt (SMI). An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 34.15.2).
6	Other SMI. An SMI arrived and caused an SMM VM exit (see Section 34.15.2) but not immediately after retirement of an I/O instruction.
7	Interrupt window. At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the "interrupt-window exiting" VM-execution control was 1.
8	NMI window. At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the "NMI-window exiting" VM-execution control was 1.
9	Task switch. Guest software attempted a task switch.
10	CPUID. Guest software attempted to execute CPUID.
11	GETSEC. Guest software attempted to execute GETSEC.

Table C-1 Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
12	HLT. Guest software attempted to execute HLT and the “HLT exiting” VM-execution control was 1.
13	INVD. Guest software attempted to execute INVD.
14	INVLPG. Guest software attempted to execute INVLPG and the “INVLPG exiting” VM-execution control was 1.
15	RDPMC. Guest software attempted to execute RDPMC and the “RDPMC exiting” VM-execution control was 1.
16	RDTSC. Guest software attempted to execute RDTSC and the “RDTSC exiting” VM-execution control was 1.
17	RSM. Guest software attempted to execute RSM in SMM.
18	VMCALL. VMCALL was executed either by guest software (causing an ordinary VM exit) or by the executive monitor (causing an SMM VM exit; see Section 34.15.2).
19	VMCLEAR. Guest software attempted to execute VMCLEAR.
20	VMLAUNCH. Guest software attempted to execute VMLAUNCH.
21	VMPTRLD. Guest software attempted to execute VMPTRLD.
22	VMPTRST. Guest software attempted to execute VMPTRST.
23	VMREAD. Guest software attempted to execute VMREAD.
24	VMRESUME. Guest software attempted to execute VMRESUME.
25	VMWRITE. Guest software attempted to execute VMWRITE.
26	VMXOFF. Guest software attempted to execute VMXOFF.
27	VMXON. Guest software attempted to execute VMXON.
28	Control-register accesses. Guest software attempted to access CR0, CR3, CR4, or CR8 using CLTS, LMSW, or MOV CR and the VM-execution control fields indicate that a VM exit should occur (see Section 25.1 for details). This basic exit reason is not used for trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1.
29	MOV DR. Guest software attempted a MOV to or from a debug register and the “MOV-DR exiting” VM-execution control was 1.
30	I/O instruction. Guest software attempted to execute an I/O instruction and either: 1: The “use I/O bitmaps” VM-execution control was 0 and the “unconditional I/O exiting” VM-execution control was 1. 2: The “use I/O bitmaps” VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1.
31	RDMSR. Guest software attempted to execute RDMSR and either: 1: The “use MSR bitmaps” VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX was in the range 00000000H – 00001FFFH and the n^{th} bit in read bitmap for low MSRs is 1, where n was the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the n^{th} bit in read bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
32	WRMSR. Guest software attempted to execute WRMSR and either: 1: The “use MSR bitmaps” VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX was in the range 00000000H – 00001FFFH and the n^{th} bit in write bitmap for low MSRs is 1, where n was the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the n^{th} bit in write bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
33	VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 26.3.1.

Table C-1 Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
34	VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs. See Section 26.4.
36	MWAIT. Guest software attempted to execute MWAIT and the “MWAIT exiting” VM-execution control was 1.
37	Monitor trap flag. A VM entry occurred due to the 1-setting of the “monitor trap flag” VM-execution control and injection of an MTF VM exit as part of VM entry. See Section 25.5.2.
39	MONITOR. Guest software attempted to execute MONITOR and the “MONITOR exiting” VM-execution control was 1.
40	PAUSE. Either guest software attempted to execute PAUSE and the “PAUSE exiting” VM-execution control was 1 or the “PAUSE-loop exiting” VM-execution control was 1 and guest software executed a PAUSE loop with execution time exceeding PLE_Window (see Section 25.1.3).
41	VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 26.8).
43	TPR below threshold. The logical processor determined that the value of bits 7:4 of the byte at offset 080H on the virtual-APIC page was below that of the TPR threshold VM-execution control field while the “use TPR shadow” VM-execution control was 1 either as part of TPR virtualization (Section 29.1.2) or VM entry (Section 26.6.7).
44	APIC access. Guest software attempted to access memory at a physical address on the APIC-access page and the “virtualize APIC accesses” VM-execution control was 1 (see Section 29.4).
45	Virtualized EOI. EOI virtualization was performed for a virtual interrupt whose vector indexed a bit set in the EOI-exit bitmap.
46	Access to GDTR or IDTR. Guest software attempted to execute LGDT, LIDT, SGDT, or SIDT and the “descriptor-table exiting” VM-execution control was 1.
47	Access to LDTR or TR. Guest software attempted to execute LLDT, LTR, SLDT, or STR and the “descriptor-table exiting” VM-execution control was 1.
48	EPT violation. An attempt to access memory with a guest-physical address was disallowed by the configuration of the EPT paging structures.
49	EPT misconfiguration. An attempt to access memory with a guest-physical address encountered a misconfigured EPT paging-structure entry.
50	INVEPT. Guest software attempted to execute INVEPT.
51	RDTSCP. Guest software attempted to execute RDTSCP and the “enable RDTSCP” and “RDTSC exiting” VM-execution controls were both 1.
52	VMX-preemption timer expired. The preemption timer counted down to zero.
53	INVVPID. Guest software attempted to execute INVVPID.
54	WBINVD. Guest software attempted to execute WBINVD and the “WBINVD exiting” VM-execution control was 1.
55	XSETBV. Guest software attempted to execute XSETBV.
56	APIC write. Guest software completed a write to the virtual-APIC page that must be virtualized by VMM software (see Section 29.4.3.3).
57	RDRAND. Guest software attempted to execute RDRAND and the “RDRAND exiting” VM-execution control was 1.
58	INVPCID. Guest software attempted to execute INVPCID and the “enable INVPCID” and “INVLPG exiting” VM-execution controls were both 1.
59	VMFUNC. Guest software invoked a VM function with the VMFUNC instruction and the VM function either was not enabled or generated a function-specific condition causing a VM exit.

...

