

# Intel<sup>®</sup> 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

---

October 2017

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

Document Number: 252046-056



Intel technologies features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Learn more at [intel.com](http://intel.com), or from the OEM or retailer.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

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

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

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

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting <http://www.intel.com/design/literature.htm>.

Intel, the Intel logo, Intel Atom, Intel Core, Intel SpeedStep, MMX, Pentium, VTune, and Xeon 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.

Copyright © 1997-2017, 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"><li>Initial release</li></ul>	November 2002
-002	<ul style="list-style-type: none"><li>Added 1-10 Documentation Changes.</li><li>Removed old Documentation Changes items that already have been incorporated in the published Software Developer's manual</li></ul>	December 2002
-003	<ul style="list-style-type: none"><li>Added 9 -17 Documentation Changes.</li><li>Removed Documentation Change #6 - References to bits Gen and Len Deleted.</li><li>Removed Documentation Change #4 - VIF Information Added to CLI Discussion</li></ul>	February 2003
-004	<ul style="list-style-type: none"><li>Removed Documentation changes 1-17.</li><li>Added Documentation changes 1-24.</li></ul>	June 2003
-005	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-24.</li><li>Added Documentation Changes 1-15.</li></ul>	September 2003
-006	<ul style="list-style-type: none"><li>Added Documentation Changes 16- 34.</li></ul>	November 2003
-007	<ul style="list-style-type: none"><li>Updated Documentation changes 14, 16, 17, and 28.</li><li>Added Documentation Changes 35-45.</li></ul>	January 2004
-008	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-45.</li><li>Added Documentation Changes 1-5.</li></ul>	March 2004
-009	<ul style="list-style-type: none"><li>Added Documentation Changes 7-27.</li></ul>	May 2004
-010	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-27.</li><li>Added Documentation Changes 1.</li></ul>	August 2004
-011	<ul style="list-style-type: none"><li>Added Documentation Changes 2-28.</li></ul>	November 2004
-012	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-28.</li><li>Added Documentation Changes 1-16.</li></ul>	March 2005
-013	<ul style="list-style-type: none"><li>Updated title.</li><li>There are no Documentation Changes for this revision of the document.</li></ul>	July 2005
-014	<ul style="list-style-type: none"><li>Added Documentation Changes 1-21.</li></ul>	September 2005
-015	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-21.</li><li>Added Documentation Changes 1-20.</li></ul>	March 9, 2006
-016	<ul style="list-style-type: none"><li>Added Documentation changes 21-23.</li></ul>	March 27, 2006
-017	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-23.</li><li>Added Documentation Changes 1-36.</li></ul>	September 2006
-018	<ul style="list-style-type: none"><li>Added Documentation Changes 37-42.</li></ul>	October 2006
-019	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-42.</li><li>Added Documentation Changes 1-19.</li></ul>	March 2007
-020	<ul style="list-style-type: none"><li>Added Documentation Changes 20-27.</li></ul>	May 2007
-021	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-27.</li><li>Added Documentation Changes 1-6</li></ul>	November 2007
-022	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-6</li><li>Added Documentation Changes 1-6</li></ul>	August 2008
-023	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-6</li><li>Added Documentation Changes 1-21</li></ul>	March 2009



Revision	Description	Date
-024	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-21</li> <li>Added Documentation Changes 1-16</li> </ul>	June 2009
-025	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-16</li> <li>Added Documentation Changes 1-18</li> </ul>	September 2009
-026	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-18</li> <li>Added Documentation Changes 1-15</li> </ul>	December 2009
-027	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-15</li> <li>Added Documentation Changes 1-24</li> </ul>	March 2010
-028	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-24</li> <li>Added Documentation Changes 1-29</li> </ul>	June 2010
-029	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-29</li> <li>Added Documentation Changes 1-29</li> </ul>	September 2010
-030	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-29</li> <li>Added Documentation Changes 1-29</li> </ul>	January 2011
-031	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-29</li> <li>Added Documentation Changes 1-29</li> </ul>	April 2011
-032	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-29</li> <li>Added Documentation Changes 1-14</li> </ul>	May 2011
-033	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-14</li> <li>Added Documentation Changes 1-38</li> </ul>	October 2011
-034	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-38</li> <li>Added Documentation Changes 1-16</li> </ul>	December 2011
-035	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-16</li> <li>Added Documentation Changes 1-18</li> </ul>	March 2012
-036	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-18</li> <li>Added Documentation Changes 1-17</li> </ul>	May 2012
-037	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-17</li> <li>Added Documentation Changes 1-28</li> </ul>	August 2012
-038	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-28</li> <li>Add Documentation Changes 1-22</li> </ul>	January 2013
-039	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-22</li> <li>Add Documentation Changes 1-17</li> </ul>	June 2013
-040	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-17</li> <li>Add Documentation Changes 1-24</li> </ul>	September 2013
-041	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-24</li> <li>Add Documentation Changes 1-20</li> </ul>	February 2014
-042	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-20</li> <li>Add Documentation Changes 1-8</li> </ul>	February 2014
-043	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-8</li> <li>Add Documentation Changes 1-43</li> </ul>	June 2014
-044	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-43</li> <li>Add Documentation Changes 1-12</li> </ul>	September 2014
-045	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-12</li> <li>Add Documentation Changes 1-22</li> </ul>	January 2015
-046	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-22</li> <li>Add Documentation Changes 1-25</li> </ul>	April 2015
-047	<ul style="list-style-type: none"> <li>Removed Documentation Changes 1-25</li> <li>Add Documentation Changes 1-19</li> </ul>	June 2015



Revision	Description	Date
-048	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-19</li><li>Add Documentation Changes 1-33</li></ul>	September 2015
-049	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-33</li><li>Add Documentation Changes 1-33</li></ul>	December 2015
-050	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-33</li><li>Add Documentation Changes 1-9</li></ul>	April 2016
-051	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-9</li><li>Add Documentation Changes 1-20</li></ul>	June 2016
-052	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-20</li><li>Add Documentation Changes 1-22</li></ul>	September 2016
-053	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-22</li><li>Add Documentation Changes 1-26</li></ul>	December 2016
-054	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-26</li><li>Add Documentation Changes 1-20</li></ul>	March 2017
-055	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-20</li><li>Add Documentation Changes 1-28</li></ul>	July 2017
-056	<ul style="list-style-type: none"><li>Removed Documentation Changes 1-28</li><li>Add Documentation Changes 1-18</li></ul>	October 2017

§

# Preface

---

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

## Affected Documents

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

## Nomenclature

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

# Summary Tables of Changes

---

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

## Codes Used in Summary Tables

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

## Documentation Changes

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 10, Volume 1
2	Updates to Chapter 11, Volume 1
3	Updates to Chapter 14, Volume 1
4	Updates to Chapter 2, Volume 2A
5	Updates to Chapter 3, Volume 2A
6	Updates to Chapter 4, Volume 2B
7	Updates to Chapter 5, Volume 2C
8	Addition of Chapter 7, Volume 2D
9	Updates to Chapter 2, Volume 3A
10	Updates to Chapter 3, Volume 3A
11	Updates to Chapter 10, Volume 3A
12	Updates to Chapter 15, Volume 3B
13	Updates to Chapter 19, Volume 3B
14	Updates to Chapter 22, Volume 3B
15	Updates to Chapter 27, Volume 3C
16	Updates to Chapter 34, Volume 3C
17	Updates to Chapter 41, Volume 3D
18	Updates to Chapter 2, Volume 4



# *Documentation Changes*

---

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



## 1. Updates to Chapter 10, Volume 1

Change bars show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

-----  
Change to this chapter: Corrected Flush to Zero flag name in various places.

# CHAPTER 10

## PROGRAMMING WITH INTEL® STREAMING SIMD EXTENSIONS (INTEL® SSE)

---

The streaming SIMD extensions (SSE) were introduced into the IA-32 architecture in the Pentium III processor family. These extensions enhance the performance of IA-32 processors for advanced 2-D and 3-D graphics, motion video, image processing, speech recognition, audio synthesis, telephony, and video conferencing.

This chapter describes SSE. Chapter 11, “Programming with Intel® Streaming SIMD Extensions 2 (Intel® SSE2),” provides information to assist in writing application programs that use SSE2 extensions. Chapter 12, “Programming with Intel® SSE3, SSSE3, Intel® SSE4 and Intel® AESNI,” provides this information for SSE3 extensions.

### 10.1 OVERVIEW OF SSE EXTENSIONS

Intel MMX technology introduced single-instruction multiple-data (SIMD) capability into the IA-32 architecture, with the 64-bit MMX registers, 64-bit packed integer data types, and instructions that allowed SIMD operations to be performed on packed integers. SSE extensions expand the SIMD execution model by adding facilities for handling packed and scalar single-precision floating-point values contained in 128-bit registers.

If `CPUID.01H:EDX.SSE[bit 25] = 1`, SSE extensions are present.

SSE extensions add the following features to the IA-32 architecture, while maintaining backward compatibility with all existing IA-32 processors, applications and operating systems.

- Eight 128-bit data registers (called XMM registers) in non-64-bit modes; sixteen XMM registers are available in 64-bit mode.
- The 32-bit MXCSR register, which provides control and status bits for operations performed on XMM registers.
- The 128-bit packed single-precision floating-point data type (four IEEE single-precision floating-point values packed into a double quadword).
- Instructions that perform SIMD operations on single-precision floating-point values and that extend SIMD operations that can be performed on integers:
  - 128-bit Packed and scalar single-precision floating-point instructions that operate on data located in MMX registers
  - 64-bit SIMD integer instructions that support additional operations on packed integer operands located in MMX registers
- Instructions that save and restore the state of the MXCSR register.
- Instructions that support explicit prefetching of data, control of the cacheability of data, and control the ordering of store operations.
- Extensions to the CPUID instruction.

These features extend the IA-32 architecture’s SIMD programming model in four important ways:

- The ability to perform SIMD operations on four packed single-precision floating-point values enhances the performance of IA-32 processors for advanced media and communications applications that use computation-intensive algorithms to perform repetitive operations on large arrays of simple, native data elements.
- The ability to perform SIMD single-precision floating-point operations in XMM registers and SIMD integer operations in MMX registers provides greater flexibility and throughput for executing applications that operate on large arrays of floating-point and integer data.
- Cache control instructions provide the ability to stream data in and out of XMM registers without polluting the caches and the ability to prefetch data to selected cache levels before it is actually used. Applications that require regular access to large amounts of data benefit from these prefetching and streaming store capabilities.
- The SFENCE (store fence) instruction provides greater control over the ordering of store operations when using weakly-ordered memory types.

SSE extensions are fully compatible with all software written for IA-32 processors. All existing software continues to run correctly, without modification, on processors that incorporate SSE extensions. Enhancements to CPUID permit detection of SSE extensions. SSE extensions are accessible from all IA-32 execution modes: protected mode, real address mode, and virtual-8086 mode.

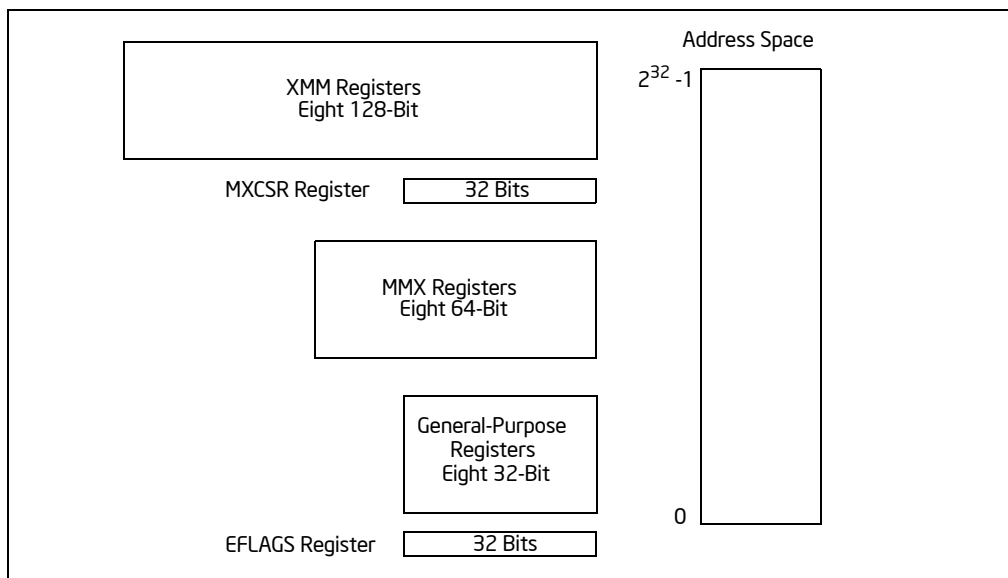
The following sections of this chapter describe the programming environment for SSE extensions, including: XMM registers, the packed single-precision floating-point data type, and SSE instructions. For additional information, see:

- Section 11.6, “Writing Applications with SSE/SSE2 Extensions”.
- Section 11.5, “SSE, SSE2, and SSE3 Exceptions,” describes the exceptions that can be generated with SSE/SSE2/SSE3 instructions.
- *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A & 2B*, provide a detailed description of these 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 these extensions into an operating-system environment.

## 10.2 SSE PROGRAMMING ENVIRONMENT

Figure 10-1 shows the execution environment for the SSE extensions. All SSE instructions operate on the XMM registers, MMX registers, and/or memory as follows:

- **XMM registers** — These eight registers (see Figure 10-2 and Section 10.2.2, “XMM Registers”) are used to operate on packed or scalar single-precision floating-point data. Scalar operations are operations performed on individual (unpacked) single-precision floating-point values stored in the low doubleword of an XMM register. XMM registers are referenced by the names XMM0 through XMM7.



**Figure 10-1. SSE Execution Environment**

- **MXCSR register** — This 32-bit register (see Figure 10-3 and Section 10.2.3, “MXCSR Control and Status Register”) provides status and control bits used in SIMD floating-point operations.
- **MMX registers** — These eight registers (see Figure 9-2) are used to perform operations on 64-bit packed integer data. They are also used to hold operands for some operations performed between the MMX and XMM registers. MMX registers are referenced by the names MM0 through MM7.
- **General-purpose registers** — The eight general-purpose registers (see Figure 3-5) are used along with the existing IA-32 addressing modes to address operands in memory. (MMX and XMM registers cannot be used to

address memory). The general-purpose registers are also used to hold operands for some SSE instructions and are referenced as EAX, EBX, ECX, EDX, EBP, ESI, EDI, and ESP.

- **EFLAGS register** — This 32-bit register (see Figure 3-8) is used to record result of some compare operations.

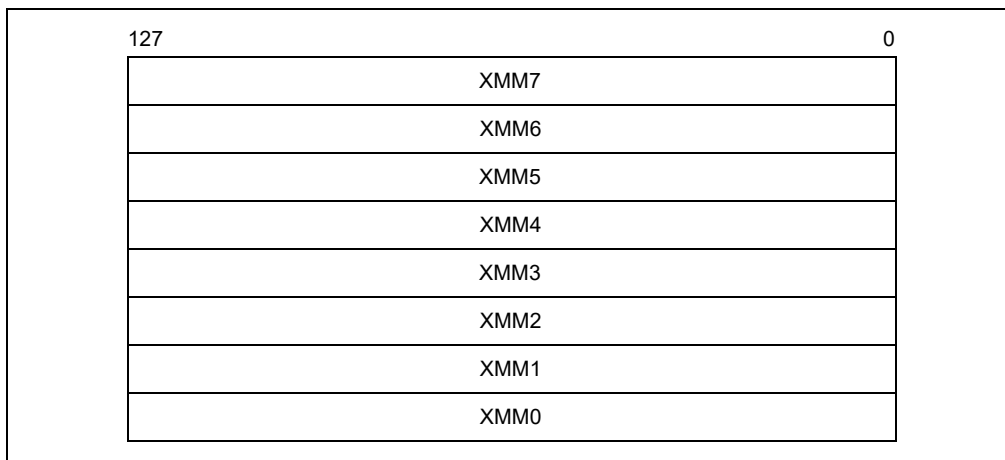
### 10.2.1 SSE in 64-Bit Mode and Compatibility Mode

In compatibility mode, SSE extensions function like they do in protected mode. In 64-bit mode, eight additional XMM registers are accessible. Registers XMM8-XMM15 are accessed by using REX prefixes. Memory operands are specified using the ModR/M, SIB encoding described in Section 3.7.5.

Some SSE instructions may be used to operate on general-purpose registers. Use the REX.W prefix to access 64-bit general-purpose registers. Note that if a REX prefix is used when it has no meaning, the prefix is ignored.

### 10.2.2 XMM Registers

Eight 128-bit XMM data registers were introduced into the IA-32 architecture with SSE extensions (see Figure 10-2). These registers can be accessed directly using the names XMM0 to XMM7; and they can be accessed independently from the x87 FPU and MMX registers and the general-purpose registers (that is, they are not aliased to any other of the processor's registers).



**Figure 10-2. XMM Registers**

SSE instructions use the XMM registers only to operate on packed single-precision floating-point operands. SSE2 extensions expand the functions of the XMM registers to operand on packed or scalar double-precision floating-point operands and packed integer operands (see Section 11.2, "SSE2 Programming Environment," and Section 12.1, "Programming Environment and Data types").

XMM registers can only be used to perform calculations on data; they cannot be used to address memory. Addressing memory is accomplished by using the general-purpose registers.

Data can be loaded into XMM registers or written from the registers to memory in 32-bit, 64-bit, and 128-bit increments. When storing the entire contents of an XMM register in memory (128-bit store), the data is stored in 16 consecutive bytes, with the low-order byte of the register being stored in the first byte in memory.

### 10.2.3 MXCSR Control and Status Register

The 32-bit MXCSR register (see Figure 10-3) contains control and status information for SSE, SSE2, and SSE3 SIMD floating-point operations. This register contains:

- flag and mask bits for SIMD floating-point exceptions
- rounding control field for SIMD floating-point operations

- flush-to-zero flag that provides a means of controlling underflow conditions on SIMD floating-point operations
- denormals-are-zeros flag that controls how SIMD floating-point instructions handle denormal source operands

The contents of this register can be loaded from memory with the LDMXCSR and FXRSTOR instructions and stored in memory with STMXCSR and FXSAVE.

Bits 16 through 31 of the MXCSR register are reserved and are cleared on a power-up or reset of the processor; attempting to write a non-zero value to these bits, using either the FXRSTOR or LDMXCSR instructions, will result in a general-protection exception (#GP) being generated.

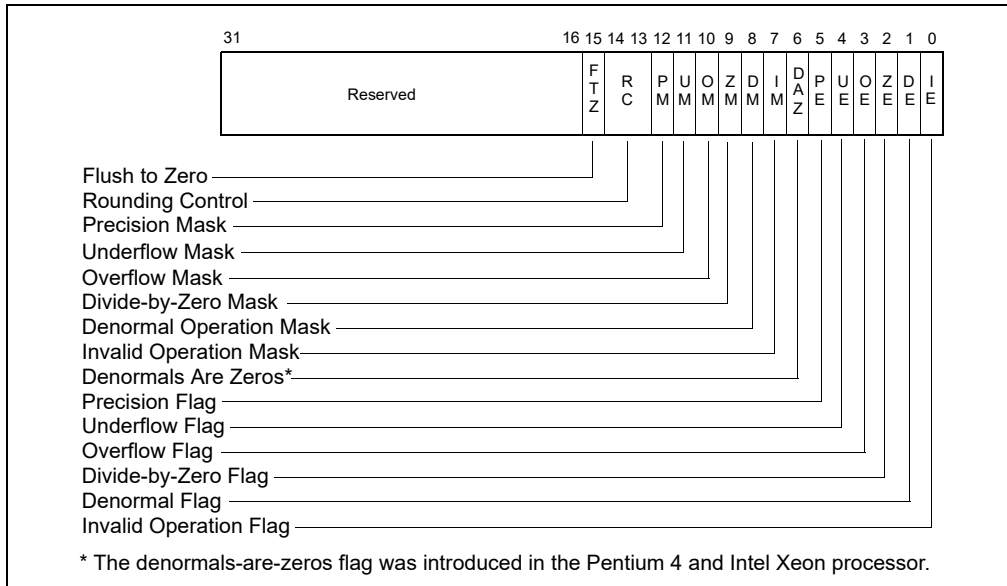


Figure 10-3. MXCSR Control/Status Register

### 10.2.3.1 SIMD Floating-Point Mask and Flag Bits

Bits 0 through 5 of the MXCSR register indicate whether a SIMD floating-point exception has been detected. They are “sticky” flags. That is, after a flag is set, it remains set until explicitly cleared. To clear these flags, use the LDMXCSR or the FXRSTOR instruction to write zeroes to them.

Bits 7 through 12 provide individual mask bits for the SIMD floating-point exceptions. An exception type is masked if the corresponding mask bit is set, and it is unmasked if the bit is clear. These mask bits are set upon a power-up or reset. This causes all SIMD floating-point exceptions to be initially masked.

If LDMXCSR or FXRSTOR clears a mask bit and sets the corresponding exception flag bit, a SIMD floating-point exception will not be generated as a result of this change. The unmasked exception will be generated only upon the execution of the next SSE/SSE2/SSE3 instruction that detects the unmasked exception condition.

For more information about the use of the SIMD floating-point exception mask and flag bits, see Section 11.5, “SSE, SSE2, and SSE3 Exceptions,” and Section 12.8, “SSE3/SSSE3 And SSE4 Exceptions.”

### 10.2.3.2 SIMD Floating-Point Rounding Control Field

Bits 13 and 14 of the MXCSR register (the rounding control [RC] field) control how the results of SIMD floating-point instructions are rounded. See Section 4.8.4, “Rounding,” for a description of the function and encoding of the rounding control bits.

### 10.2.3.3 Flush-To-Zero

Bit 15 (FTZ) of the MXCSR register enables the flush-to-zero mode, which controls the masked response to a SIMD floating-point underflow condition. When the underflow exception is masked and the flush-to-zero mode is enabled, the processor performs the following operations when it detects a floating-point underflow condition.

- Returns a zero result with the sign of the true result.
- Sets the precision and underflow exception flags.

If the underflow exception is not masked, the flush-to-zero bit is ignored.

The flush-to-zero mode is not compatible with IEEE Standard 754. The IEEE-mandated masked response to underflow is to deliver the denormalized result (see Section 4.8.3.2, “Normalized and Denormalized Finite Numbers”). The flush-to-zero mode is provided primarily for performance reasons. At the cost of a slight precision loss, faster execution can be achieved for applications where underflows are common and rounding the underflow result to zero can be tolerated.

The flush-to-zero bit is cleared upon a power-up or reset of the processor, disabling the flush-to-zero mode.

#### 10.2.3.4 Denormals-Are-Zeros

Bit 6 (DAZ) of the MXCSR register enables the denormals-are-zeros mode, which controls the processor’s response to a SIMD floating-point denormal operand condition. When the denormals-are-zeros flag is set, the processor converts all denormal source operands to a zero with the sign of the original operand before performing any computations on them. The processor does not set the denormal-operand exception flag (DE), regardless of the setting of the denormal-operand exception mask bit (DM); and it does not generate a denormal-operand exception if the exception is unmasked.

The denormals-are-zeros mode is not compatible with IEEE Standard 754 (see Section 4.8.3.2, “Normalized and Denormalized Finite Numbers”). The denormals-are-zeros mode is provided to improve processor performance for applications such as streaming media processing, where rounding a denormal operand to zero does not appreciably affect the quality of the processed data.

The denormals-are-zeros flag is cleared upon a power-up or reset of the processor, disabling the denormals-are-zeros mode.

The denormals-are-zeros mode was introduced in the Pentium 4 and Intel Xeon processor with the SSE2 extensions; however, it is fully compatible with the SSE SIMD floating-point instructions (that is, the denormals-are-zeros flag affects the operation of the SSE SIMD floating-point instructions). In earlier IA-32 processors and in some models of the Pentium 4 processor, this flag (bit 6) is reserved. See Section 11.6.3, “Checking for the DAZ Flag in the MXCSR Register,” for instructions for detecting the availability of this feature.

Attempting to set bit 6 of the MXCSR register on processors that do not support the DAZ flag will cause a general-protection exception (#GP). See Section 11.6.6, “Guidelines for Writing to the MXCSR Register,” for instructions for preventing such general-protection exceptions by using the MXCSR\_MASK value returned by the FXSAVE instruction.

### 10.2.4 Compatibility of SSE Extensions with SSE2/SSE3/MMX and the x87 FPU

The state (XMM registers and MXCSR register) introduced into the IA-32 execution environment with the SSE extensions is shared with SSE2 and SSE3 extensions. SSE/SSE2/SSE3 instructions are fully compatible; they can be executed together in the same instruction stream with no need to save state when switching between instruction sets.

XMM registers are independent of the x87 FPU and MMX registers, so SSE/SSE2/SSE3 operations performed on the XMM registers can be performed in parallel with operations on the x87 FPU and MMX registers (see Section 11.6.7, “Interaction of SSE/SSE2 Instructions with x87 FPU and MMX Instructions”).

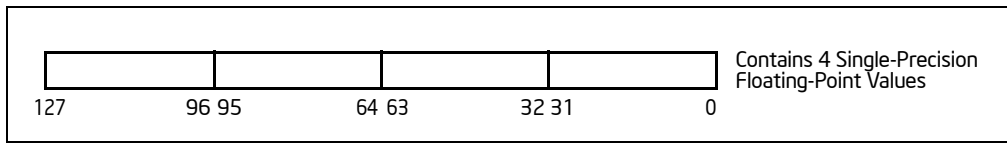
The FXSAVE and FXRSTOR instructions save and restore the SSE/SSE2/SSE3 states along with the x87 FPU and MMX state.

## 10.3 SSE DATA TYPES

SSE extensions introduced one data type, the 128-bit packed single-precision floating-point data type, to the IA-32 architecture (see Figure 10-4). This data type consists of four IEEE 32-bit single-precision floating-point values



packed into a double quadword. (See Figure 4-3 for the layout of a single-precision floating-point value; refer to Section 4.2.2, "Floating-Point Data Types," for a detailed description of the single-precision floating-point format.)



**Figure 10-4. 128-Bit Packed Single-Precision Floating-Point Data Type**

This 128-bit packed single-precision floating-point data type is operated on in the XMM registers or in memory. Conversion instructions are provided to convert two packed single-precision floating-point values into two packed doubleword integers or a scalar single-precision floating-point value into a doubleword integer (see Figure 11-8).

SSE extensions provide conversion instructions between XMM registers and MMX registers, and between XMM registers and general-purpose bit registers. See Figure 11-8.

The address of a 128-bit packed memory operand must be aligned on a 16-byte boundary, except in the following cases:

- The MOVUPS instruction supports unaligned accesses.
- Scalar instructions that use a 4-byte memory operand that is not subject to alignment requirements.

Figure 4-2 shows the byte order of 128-bit (double quadword) data types in memory.

## 10.4 SSE INSTRUCTION SET

SSE instructions are divided into four functional groups

- Packed and scalar single-precision floating-point instructions
- 64-bit SIMD integer instructions
- State management instructions
- Cacheability control, prefetch, and memory ordering instructions

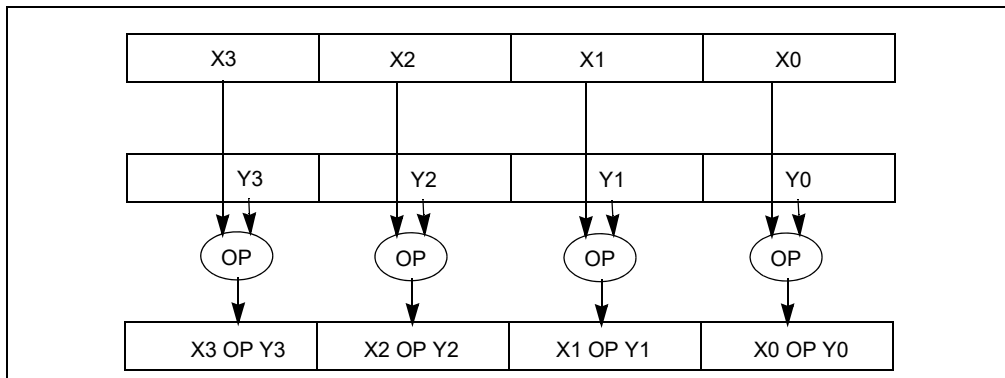
The following sections give an overview of each of the instructions in these groups.

### 10.4.1 SSE Packed and Scalar Floating-Point Instructions

The packed and scalar single-precision floating-point instructions are divided into the following subgroups:

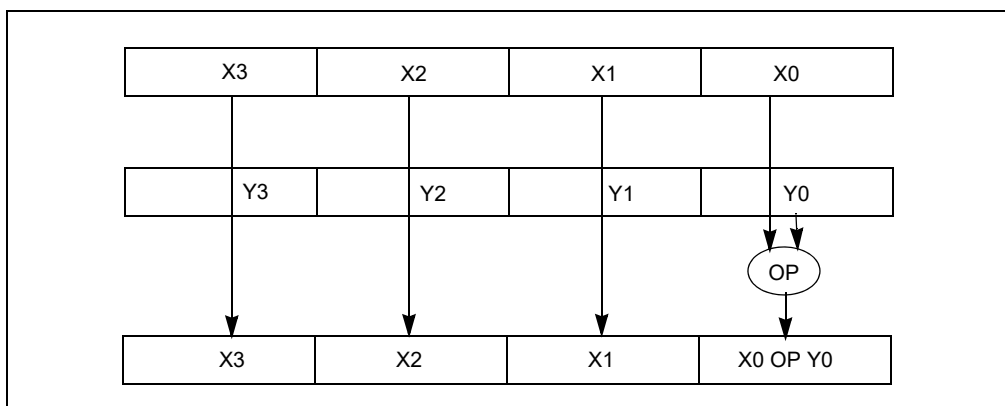
- Data movement instructions
- Arithmetic instructions
- Logical instructions
- Comparison instructions
- Shuffle instructions
- Conversion instructions

The packed single-precision floating-point instructions perform SIMD operations on packed single-precision floating-point operands (see Figure 10-5). Each source operand contains four single-precision floating-point values, and the destination operand contains the results of the operation (OP) performed in parallel on the corresponding values (X0 and Y0, X1 and Y1, X2 and Y2, and X3 and Y3) in each operand.



**Figure 10-5. Packed Single-Precision Floating-Point Operation**

The scalar single-precision floating-point instructions operate on the low (least significant) doublewords of the two source operands (X0 and Y0); see Figure 10-6. The three most significant doublewords (X1, X2, and X3) of the first source operand are passed through to the destination. The scalar operations are similar to the floating-point operations performed in the x87 FPU data registers with the precision control field in the x87 FPU control word set for single precision (24-bit significand), except that x87 stack operations use a 15-bit exponent range for the result, while SSE operations use an 8-bit exponent range.



**Figure 10-6. Scalar Single-Precision Floating-Point Operation**

### 10.4.1.1 SSE Data Movement Instructions

SSE data movement instructions move single-precision floating-point data between XMM registers and between an XMM register and memory.

The MOVAPS (move aligned packed single-precision floating-point values) instruction transfers a double quadword operand containing four packed single-precision floating-point values from memory to an XMM register and vice versa, or between XMM registers. The memory address must be aligned to a 16-byte boundary; otherwise, a general-protection exception (#GP) is generated.

The MOVUPS (move unaligned packed single-precision, floating-point) instruction performs the same operations as the MOVAPS instruction, except that 16-byte alignment of a memory address is not required.

The MOVSS (move scalar single-precision floating-point) instruction transfers a 32-bit single-precision floating-point operand from memory to the low doubleword of an XMM register and vice versa, or between XMM registers.

The MOVLPD (move low packed single-precision floating-point) instruction moves two packed single-precision floating-point values from memory to the low quadword of an XMM register and vice versa. The high quadword of the register is left unchanged.

The MOVHPS (move high packed single-precision floating-point) instruction moves two packed single-precision floating-point values from memory to the high quadword of an XMM register and vice versa. The low quadword of the register is left unchanged.

The MOVLHPS (move packed single-precision floating-point low to high) instruction moves two packed single-precision floating-point values from the low quadword of the source XMM register into the high quadword of the destination XMM register. The low quadword of the destination register is left unchanged.

The MOVHLPS (move packed single-precision floating-point high to low) instruction moves two packed single-precision floating-point values from the high quadword of the source XMM register into the low quadword of the destination XMM register. The high quadword of the destination register is left unchanged.

The MOVMSKPS (move packed single-precision floating-point mask) instruction transfers the most significant bit of each of the four packed single-precision floating-point numbers in an XMM register to a general-purpose register. This 4-bit value can then be used as a condition to perform branching.

### 10.4.1.2 SSE Arithmetic Instructions

SSE arithmetic instructions perform addition, subtraction, multiply, divide, reciprocal, square root, reciprocal of square root, and maximum/minimum operations on packed and scalar single-precision floating-point values.

The ADDPS (add packed single-precision floating-point values) and SUBPS (subtract packed single-precision floating-point values) instructions add and subtract, respectively, two packed single-precision floating-point operands.

The ADDSS (add scalar single-precision floating-point values) and SUBSS (subtract scalar single-precision floating-point values) instructions add and subtract, respectively, the low single-precision floating-point values of two operands and store the result in the low doubleword of the destination operand.

The MULPS (multiply packed single-precision floating-point values) instruction multiplies two packed single-precision floating-point operands.

The MULSS (multiply scalar single-precision floating-point values) instruction multiplies the low single-precision floating-point values of two operands and stores the result in the low doubleword of the destination operand.

The DIVPS (divide packed, single-precision floating-point values) instruction divides two packed single-precision floating-point operands.

The DIVSS (divide scalar single-precision floating-point values) instruction divides the low single-precision floating-point values of two operands and stores the result in the low doubleword of the destination operand.

The RCPPS (compute reciprocals of packed single-precision floating-point values) instruction computes the approximate reciprocals of values in a packed single-precision floating-point operand.

The RCPSS (compute reciprocal of scalar single-precision floating-point values) instruction computes the approximate reciprocal of the low single-precision floating-point value in the source operand and stores the result in the low doubleword of the destination operand.

The SQRTPS (compute square roots of packed single-precision floating-point values) instruction computes the square roots of the values in a packed single-precision floating-point operand.

The SQRTSS (compute square root of scalar single-precision floating-point values) instruction computes the square root of the low single-precision floating-point value in the source operand and stores the result in the low doubleword of the destination operand.

The RSQRTPS (compute reciprocals of square roots of packed single-precision floating-point values) instruction computes the approximate reciprocals of the square roots of the values in a packed single-precision floating-point operand.

The RSQRTSS (reciprocal of square root of scalar single-precision floating-point value) instruction computes the approximate reciprocal of the square root of the low single-precision floating-point value in the source operand and stores the result in the low doubleword of the destination operand.

The MAXPS (return maximum of packed single-precision floating-point values) instruction compares the corresponding values from two packed single-precision floating-point operands and returns the numerically greater value from each comparison to the destination operand.

The MAXSS (return maximum of scalar single-precision floating-point values) instruction compares the low values from two packed single-precision floating-point operands and returns the numerically greater value from the comparison to the low doubleword of the destination operand.

The MINPS (return minimum of packed single-precision floating-point values) instruction compares the corresponding values from two packed single-precision floating-point operands and returns the numerically lesser value from each comparison to the destination operand.

The MINSS (return minimum of scalar single-precision floating-point values) instruction compares the low values from two packed single-precision floating-point operands and returns the numerically lesser value from the comparison to the low doubleword of the destination operand.

## 10.4.2 SSE Logical Instructions

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

The ANDPS (bitwise logical AND of packed single-precision floating-point values) instruction returns the logical AND of two packed single-precision floating-point operands.

The ANDNPS (bitwise logical AND NOT of packed single-precision, floating-point values) instruction returns the logical AND NOT of two packed single-precision floating-point operands.

The ORPS (bitwise logical OR of packed single-precision, floating-point values) instruction returns the logical OR of two packed single-precision floating-point operands.

The XORPS (bitwise logical XOR of packed single-precision, floating-point values) instruction returns the logical XOR of two packed single-precision floating-point operands.

### 10.4.2.1 SSE Comparison Instructions

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

The CMPPS (compare packed single-precision floating-point values) instruction compares the corresponding values from two packed single-precision floating-point operands, using an immediate operand as a predicate, and returns a 32-bit mask result of all 1s or all 0s for each comparison to the destination operand. The value of the immediate operand allows the selection of any of 8 compare conditions: equal, less than, less than equal, unordered, not equal, not less than, not less than or equal, or ordered.

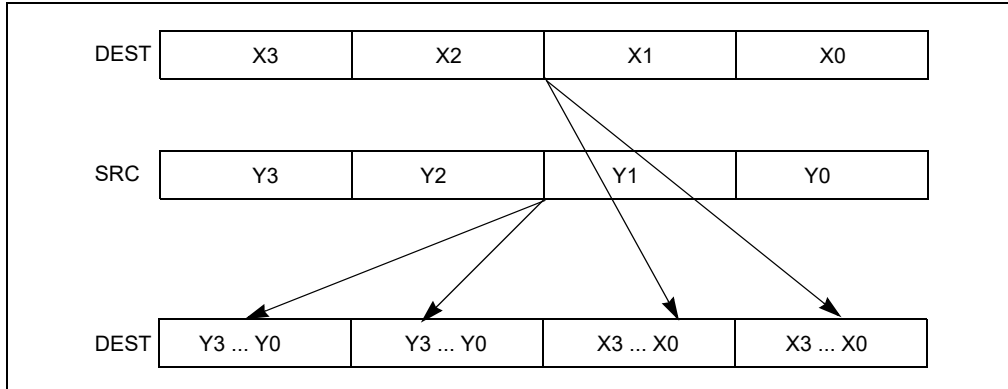
The CMPSS (compare scalar single-precision, floating-point values) instruction compares the low values from two packed single-precision floating-point operands, using an immediate operand as a predicate, and returns a 32-bit mask result of all 1s or all 0s for the comparison to the low doubleword of the destination operand. The immediate operand selects the compare conditions as with the CMPPS instruction.

The COMISS (compare scalar single-precision floating-point values and set EFLAGS) and UCOMISS (unordered compare scalar single-precision floating-point values and set EFLAGS) instructions compare the low values of two packed single-precision floating-point operands and set the ZF, PF, and CF flags in the EFLAGS register to show the result (greater than, less than, equal, or unordered). These two instructions differ as follows: the COMISS instruction signals a floating-point invalid-operation (#I) exception when a source operand is either a QNaN or an SNaN; the UCOMISS instruction only signals an invalid-operation exception when a source operand is an SNaN.

### 10.4.2.2 SSE Shuffle and Unpack Instructions

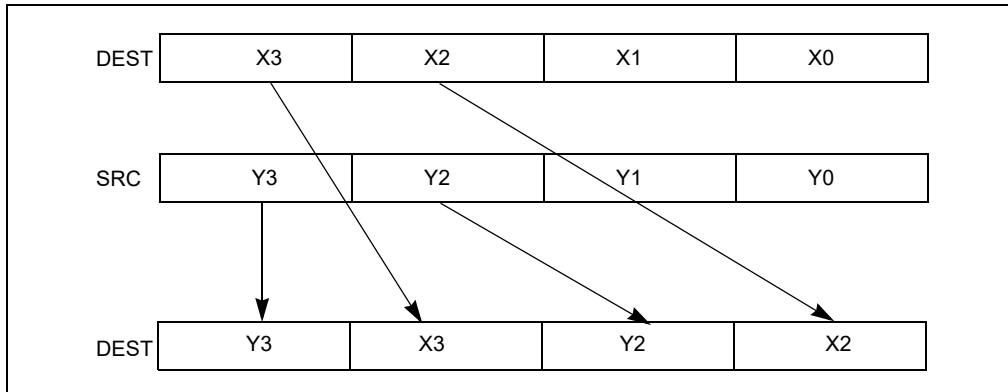
SSE shuffle and unpack instructions shuffle or interleave the contents of two packed single-precision floating-point values and store the results in the destination operand.

The SHUFPS (shuffle packed single-precision floating-point values) instruction places any two of the four packed single-precision floating-point values from the destination operand into the two low-order doublewords of the destination operand, and places any two of the four packed single-precision floating-point values from the source operand in the two high-order doublewords of the destination operand (see Figure 10-7). By using the same register for the source and destination operands, the SHUFPS instruction can shuffle four single-precision floating-point values into any order.



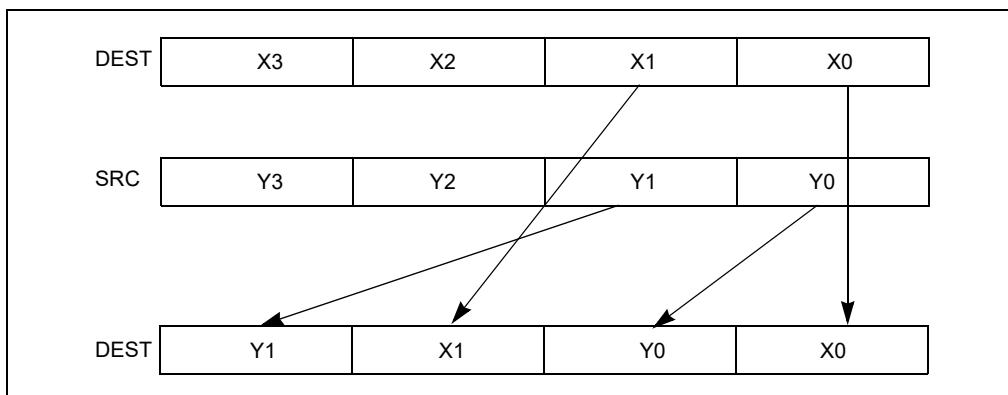
**Figure 10-7. SHUFPS Instruction, Packed Shuffle Operation**

The UNPCKHPS (unpack and interleave high packed single-precision floating-point values) instruction performs an interleaved unpack of the high-order single-precision floating-point values from the source and destination operands and stores the result in the destination operand (see Figure 10-8).



**Figure 10-8. UNPCKHPS Instruction, High Unpack and Interleave Operation**

The UNPCKLPS (unpack and interleave low packed single-precision floating-point values) instruction performs an interleaved unpack of the low-order single-precision floating-point values from the source and destination operands and stores the result in the destination operand (see Figure 10-9).



**Figure 10-9. UNPCKLPS Instruction, Low Unpack and Interleave Operation**

### 10.4.3 SSE Conversion Instructions

SSE conversion instructions (see Figure 11-8) support packed and scalar conversions between single-precision floating-point and doubleword integer formats.

The CVTPI2PS (convert packed doubleword integers to packed single-precision floating-point values) instruction converts two packed signed doubleword integers into two packed single-precision floating-point values. When the conversion is inexact, the result is rounded according to the rounding mode selected in the MXCSR register.

The CVTSI2SS (convert doubleword integer to scalar single-precision floating-point value) instruction converts a signed doubleword integer into a single-precision floating-point value. When the conversion is inexact, the result is rounded according to the rounding mode selected in the MXCSR register.

The CVTIPS2PI (convert packed single-precision floating-point values to packed doubleword integers) instruction converts two packed single-precision floating-point values into two packed signed doubleword integers. When the conversion is inexact, the result is rounded according to the rounding mode selected in the MXCSR register. The CVTTPS2PI (convert with truncation packed single-precision floating-point values to packed doubleword integers) instruction is similar to the CVTIPS2PI instruction, except that truncation is used to round a source value to an integer value (see Section 4.8.4.2, “Truncation with SSE and SSE2 Conversion Instructions”).

The CVTSS2SI (convert scalar single-precision floating-point value to doubleword integer) instruction converts a single-precision floating-point value into a signed doubleword integer. When the conversion is inexact, the result is rounded according to the rounding mode selected in the MXCSR register. The CVTTPS2SI (convert with truncation scalar single-precision floating-point value to doubleword integer) instruction is similar to the CVTSS2SI instruction, except that truncation is used to round the source value to an integer value (see Section 4.8.4.2, “Truncation with SSE and SSE2 Conversion Instructions”).

### 10.4.4 SSE 64-Bit SIMD Integer Instructions

SSE extensions add the following 64-bit packed integer instructions to the IA-32 architecture. These instructions operate on data in MMX registers and 64-bit memory locations.

#### NOTE

When SSE2 extensions are present in an IA-32 processor, these instructions are extended to operate on 128-bit operands in XMM registers and 128-bit memory locations.

The PAVGB (compute average of packed unsigned byte integers) and PAVGW (compute average of packed unsigned word integers) instructions compute a SIMD average of two packed unsigned byte or word integer operands, respectively. For each corresponding pair of data elements in the packed source operands, the elements are added together, a 1 is added to the temporary sum, and that result is shifted right one bit position.

The PEXTRW (extract word) instruction copies a selected word from an MMX register into a general-purpose register.

The PINSRW (insert word) instruction copies a word from a general-purpose register or from memory into a selected word location in an MMX register.

The PMAXUB (maximum of packed unsigned byte integers) instruction compares the corresponding unsigned byte integers in two packed operands and returns the greater of each comparison to the destination operand.

The PMINUB (minimum of packed unsigned byte integers) instruction compares the corresponding unsigned byte integers in two packed operands and returns the lesser of each comparison to the destination operand.

The PMAWSW (maximum of packed signed word integers) instruction compares the corresponding signed word integers in two packed operands and returns the greater of each comparison to the destination operand.

The PMINSW (minimum of packed signed word integers) instruction compares the corresponding signed word integers in two packed operands and returns the lesser of each comparison to the destination operand.

The PMOVMASKB (move byte mask) instruction creates an 8-bit mask from the packed byte integers in an MMX register and stores the result in the low byte of a general-purpose register. The mask contains the most significant bit of each byte in the MMX register. (When operating on 128-bit operands, a 16-bit mask is created.)

The PMULHUW (multiply packed unsigned word integers and store high result) instruction performs a SIMD unsigned multiply of the words in the two source operands and returns the high word of each result to an MMX register.

The PSADBW (compute sum of absolute differences) instruction computes the SIMD absolute differences of the corresponding unsigned byte integers in two source operands, sums the differences, and stores the sum in the low word of the destination operand.

The PSHUFW (shuffle packed word integers) instruction shuffles the words in the source operand according to the order specified by an 8-bit immediate operand and returns the result to the destination operand.

## 10.4.5 MXCSR State Management Instructions

The MXCSR state management instructions (LDMXCSR and STMXCSR) load and save the state of the MXCSR register, respectively. The LDMXCSR instruction loads the MXCSR register from memory, while the STMXCSR instruction stores the contents of the register to memory.

## 10.4.6 Cacheability Control, Prefetch, and Memory Ordering Instructions

SSE extensions introduce several new instructions to give programs more control over the caching of data. They also introduces the PREFETCH $h$  instructions, which provide the ability to prefetch data to a specified cache level, and the SFENCE instruction, which enforces program ordering on stores. These instructions are described in the following sections.

### 10.4.6.1 Cacheability Control Instructions

The following three instructions enable data from the MMX and XMM registers to be stored to memory using a non-temporal hint. The non-temporal hint directs the processor to store the data to memory without writing the data into the cache hierarchy. See Section 10.4.6.2, “Caching of Temporal vs. Non-Temporal Data,” for information about non-temporal stores and hints.

The MOVNTQ (store quadword using non-temporal hint) instruction stores packed integer data from an MMX register to memory, using a non-temporal hint.

The MOVNTPS (store packed single-precision floating-point values using non-temporal hint) instruction stores packed floating-point data from an XMM register to memory, using a non-temporal hint.

The MASKMOVQ (store selected bytes of quadword) instruction stores selected byte integers from an MMX register to memory, using a byte mask to selectively write the individual bytes. This instruction also uses a non-temporal hint.

### 10.4.6.2 Caching of Temporal vs. Non-Temporal Data

Data referenced by a program can be temporal (data will be used again) or non-temporal (data will be referenced once and not reused in the immediate future). For example, program code is generally temporal, whereas, multi-media data, such as the display list in a 3-D graphics application, is often non-temporal. To make efficient use of the processor’s caches, it is generally desirable to cache temporal data and not cache non-temporal data. Overloading the processor’s caches with non-temporal data is sometimes referred to as “polluting the caches.” The SSE and SSE2 cacheability control instructions enable a program to write non-temporal data to memory in a manner that minimizes pollution of caches.

These SSE and SSE2 non-temporal store instructions minimize cache pollutions by treating the memory being accessed as the write combining (WC) type. If a program specifies a non-temporal store with one of these instructions and the memory type of the destination region is write back (WB), write through (WT), or write combining (WC), the processor will do the following:

- If the memory location being written to is present in the cache hierarchy, the data in the caches is evicted.<sup>1</sup>

1. Some older CPU implementations (e.g., Pentium M) allowed addresses being written with a non-temporal store instruction to be updated in-place if the memory type was not WC and line was already in the cache.



- The non-temporal data is written to memory with WC semantics.

See also: Chapter 11, “Memory Cache Control,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

Using the WC semantics, the store transaction will be weakly ordered, meaning that the data may not be written to memory in program order, and the store will not write allocate (that is, the processor will not fetch the corresponding cache line into the cache hierarchy, prior to performing the store). Also, different processor implementations may choose to collapse and combine these stores.

The memory type of the region being written to can override the non-temporal hint, if the memory address specified for the non-temporal store is in uncacheable memory. Uncacheable as referred to here means that the region being written to has been mapped with either an uncacheable (UC) or write protected (WP) memory type.

In general, WC semantics require software to ensure coherence, with respect to other processors and other system agents (such as graphics cards). Appropriate use of synchronization and fencing must be performed for producer-consumer usage models. Fencing ensures that all system agents have global visibility of the stored data; for instance, failure to fence may result in a written cache line staying within a processor and not being visible to other agents.

The memory type visible on the bus in the presence of memory type aliasing is implementation specific. As one possible example, the memory type written to the bus may reflect the memory type for the first store to this line, as seen in program order; other alternatives are possible. This behavior should be considered reserved, and dependence on the behavior of any particular implementation risks future incompatibility.

#### NOTE

Some older CPU implementations (e.g., Pentium M) may implement non-temporal stores by updating in place data that already reside in the cache hierarchy. For such processors, the destination region should also be mapped as WC. If mapped as WB or WT, there is the potential for speculative processor reads to bring the data into the caches; in this case, non-temporal stores would then update in place, and data would not be flushed from the processor by a subsequent fencing operation.

### 10.4.6.3 PREFETCHh Instructions

The `PREFETCHh` instructions permit programs to load data into the processor at a suggested cache level, so that the data is closer to the processor’s load and store unit when it is needed. These instructions fetch 32 aligned bytes (or more, depending on the implementation) containing the addressed byte to a location in the cache hierarchy specified by the temporal locality hint (see Table 10-1). In this table, the first-level cache is closest to the processor and second-level cache is farther away from the processor than the first-level cache. The hints specify a prefetch of either temporal or non-temporal data (see Section 10.4.6.2, “Caching of Temporal vs. Non-Temporal Data”). Subsequent accesses to temporal data are treated like normal accesses, while those to non-temporal data will continue to minimize cache pollution. If the data is already present at a level of the cache hierarchy that is closer to the processor, the `PREFETCHh` instruction will not result in any data movement. The `PREFETCHh` instructions do not affect functional behavior of the program.

See Section 11.6.13, “Cacheability Hint Instructions,” for additional information about the `PREFETCHh` instructions.

**Table 10-1. `PREFETCHh` Instructions Caching Hints**

<b>PREFETCHh Instruction Mnemonic</b>	<b>Actions</b>
<code>PREFETCHT0</code>	Temporal data—fetch data into all levels of cache hierarchy: <ul style="list-style-type: none"> <li>▪ Pentium III processor—1st-level cache or 2nd-level cache</li> <li>▪ Pentium 4 and Intel Xeon processor—2nd-level cache</li> </ul>
<code>PREFETCHT1</code>	Temporal data—fetch data into level 2 cache and higher <ul style="list-style-type: none"> <li>▪ Pentium III processor—2nd-level cache</li> <li>▪ Pentium 4 and Intel Xeon processor—2nd-level cache</li> </ul>



**Table 10-1. PREFETCHh Instructions Caching Hints (Contd.)**

PREFETCHh Instruction Mnemonic	Actions
PREFETCHT2	Temporal data—fetch data into level 2 cache and higher <ul style="list-style-type: none"> <li>▪ Pentium III processor—2nd-level cache</li> <li>▪ Pentium 4 and Intel Xeon processor—2nd-level cache</li> </ul>
PREFETCHNTA	Non-temporal data—fetch data into location close to the processor, minimizing cache pollution <ul style="list-style-type: none"> <li>▪ Pentium III processor—1st-level cache</li> <li>▪ Pentium 4 and Intel Xeon processor—2nd-level cache</li> </ul>

#### 10.4.6.4 SFENCE Instruction

The SFENCE (Store Fence) instruction controls write ordering by creating a fence for memory store operations. This instruction guarantees that the result of every store instruction that precedes the store fence in program order is globally visible before any store instruction that follows the fence. The SFENCE instruction provides an efficient way of ensuring ordering between procedures that produce weakly-ordered data and procedures that consume that data.

## 10.5 FXSAVE AND FXRSTOR INSTRUCTIONS

The FXSAVE and FXRSTOR instructions were introduced into the IA-32 architecture in the Pentium II processor family (prior to the introduction of the SSE extensions). The original versions of these instructions performed a fast save and restore, respectively, of the x87 execution environment (x87 state). (By saving the state of the x87 FPU data registers, the FXSAVE and FXRSTOR instructions implicitly save and restore the state of the MMX registers.)

The SSE extensions expanded the scope of these instructions to save and restore the states of the XMM registers and the MXCSR register (SSE state), along with x87 state.

The FXSAVE and FXRSTOR instructions can be used in place of the FSAVE/FNSAVE and FRSTOR instructions; however, the operation of the FXSAVE and FXRSTOR instructions are not identical to the operation of FSAVE/FNSAVE and FRSTOR.

### NOTE

The FXSAVE and FXRSTOR instructions are not considered part of the SSE instruction group. They have a separate CPUID feature bit to indicate whether they are present (if CPUID.01H:EDX.FXSR[bit 24] = 1).

The CPUID feature bit for SSE extensions does not indicate the presence of FXSAVE and FXRSTOR.

The FXSAVE and FXRSTOR instructions organize x87 state and SSE state in a region of memory called the FXSAVE area. Section 10.5.1 provides details of the FXSAVE area and its format. Section 10.5.2 describes operation of FXSAVE, and Section 10.5.3 describes the operation of FXRSTOR.

### 10.5.1 FXSAVE Area

The FXSAVE and FXRSTOR instructions organize x87 state and SSE state in a region of memory called the FXSAVE area. Each of the instructions takes a memory operand that specifies the 16-byte aligned base address of the FXSAVE area on which it operates.

Every FXSAVE area comprises the 512 bytes starting at the area's base address. Table 10-2 illustrates the format of the first 416 bytes of the legacy region of an FXSAVE area.

**Table 10-2. Format of an FXSAVE Area**

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
Reserved	CS or FPU IP bits 63:32	FPU IP bits 31:0		FOP	Rsvd.	FTW	FSW	FCW	<b>0</b>
MXCSR_MASK		MXCSR		Reserved	DS or FPU DP bits 63:32		FPU DP bits 31:0		<b>16</b>
Reserved			ST0/MM0						<b>32</b>
Reserved			ST1/MM1						<b>48</b>
Reserved			ST2/MM2						<b>64</b>
Reserved			ST3/MM3						<b>80</b>
Reserved			ST4/MM4						<b>96</b>
Reserved			ST5/MM5						<b>112</b>
Reserved			ST6/MM6						<b>128</b>
Reserved			ST7/MM7						<b>144</b>
XMM0									<b>160</b>
XMM1									<b>176</b>
XMM2									<b>192</b>
XMM3									<b>208</b>
XMM4									<b>224</b>
XMM5									<b>240</b>
XMM6									<b>256</b>
XMM7									<b>272</b>
XMM8									<b>288</b>
XMM9									<b>304</b>
XMM10									<b>320</b>
XMM11									<b>336</b>
XMM12									<b>352</b>
XMM13									<b>368</b>
XMM14									<b>384</b>
XMM15									<b>400</b>

The x87 state component comprises bytes 23:0 and bytes 159:32. The SSE state component comprises bytes 31:24 and bytes 415:160. FXSAVE and FXRSTOR do not use bytes 511:416; bytes 463:416 are reserved. Section 10.5.2 and Section 10.5.3 provide details of how FXSAVE and FXRSTOR use an FXSAVE area.

### 10.5.1.1 x87 State

Table 10-2 illustrates how FXSAVE and FXRSTOR organize x87 state and SSE state; the x87 state is listed below, along with details of its interactions with FXSAVE and FXRSTOR:

- Bytes 1:0, 3:2, and 7:6 are used for x87 FPU Control Word (FCW), x87 FPU Status Word (FSW), and x87 FPU Opcode (FOP), respectively.

- Byte 4 is used for an abridged version of the x87 FPU Tag Word (FTW). The following items describe its usage:
  - For each  $j$ ,  $0 \leq j \leq 7$ , FXSAVE saves a 0 into bit  $j$  of byte 4 if x87 FPU data register  $ST_j$  has an empty tag; otherwise, FXSAVE saves a 1 into bit  $j$  of byte 4.
  - For each  $j$ ,  $0 \leq j \leq 7$ , FXRSTOR establishes the tag value for x87 FPU data register  $ST_j$  as follows. If bit  $j$  of byte 4 is 0, the tag for  $ST_j$  in the tag register for that data register is marked empty (11B); otherwise, the x87 FPU sets the tag for  $ST_j$  based on the value being loaded into that register (see below).
- Bytes 15:8 are used as follows:
  - If the instruction has no REX prefix, or if  $REX.W = 0$ :
    - Bytes 11:8 are used for bits 31:0 of the x87 FPU Instruction Pointer Offset (FIP).
    - If  $CPUID.(EAX=07H,ECX=0H):EBX[\text{bit } 13] = 0$ , bytes 13:12 are used for x87 FPU Instruction Pointer Selector (FPU CS). Otherwise, the processor deprecates the FPU CS value: FXSAVE saves it as 0000H.
    - Bytes 15:14 are not used.
  - If the instruction has a REX prefix with  $REX.W = 1$ , bytes 15:8 are used for the full 64 bits of FIP.
- Bytes 23:16 are used as follows:
  - If the instruction has no REX prefix, or if  $REX.W = 0$ :
    - Bytes 19:16 are used for bits 31:0 of the x87 FPU Data Pointer Offset (FDP).
    - If  $CPUID.(EAX=07H,ECX=0H):EBX[\text{bit } 13] = 0$ , bytes 21:20 are used for x87 FPU Data Pointer Selector (FPU DS). Otherwise, the processor deprecates the FPU DS value: FXSAVE saves it as 0000H.
    - Bytes 23:22 are not used.
  - If the instruction has a REX prefix with  $REX.W = 1$ , bytes 23:16 are used for the full 64 bits of FDP.
- Bytes 31:24 are used for SSE state (see Section 10.5.1.2).
- Bytes 159:32 are used for the registers  $ST_0$ – $ST_7$  ( $MM_0$ – $MM_7$ ). Each of the 8 registers is allocated a 128-bit region, with the low 80 bits used for the register and the upper 48 bits unused.

### 10.5.1.2 SSE State

Table 10-2 illustrates how FXSAVE and FXRSTOR organize x87 state and SSE state; the SSE state is listed below, along with details of its interactions with FXSAVE and FXRSTOR:

- Bytes 23:0 are used for x87 state (see Section 10.5.1.1).
- Bytes 27:24 are used for the MXCSR register. FXRSTOR generates a general-protection fault (#GP) in response to an attempt to set any of the reserved bits in the MXCSR register.
- Bytes 31:28 are used for the MXCSR\_MASK value. FXRSTOR ignores this field.
- Bytes 159:32 are used for x87 state.
- Bytes 287:160 are used for the registers  $XMM_0$ – $XMM_7$ .
- Bytes 415:288 are used for the registers  $XMM_8$ – $XMM_{15}$ . These fields are used only in 64-bit mode. Executions of FXSAVE outside 64-bit mode do not write to these bytes; executions of FXRSTOR outside 64-bit mode do not read these bytes and do not update  $XMM_8$ – $XMM_{15}$ .

If  $CR4.OSFXSR = 0$ , FXSAVE and FXRSTOR may or may not operate on SSE state; this behavior is implementation dependent. Moreover, SSE instructions cannot be used unless  $CR4.OSFXSR = 1$ .

## 10.5.2 Operation of FXSAVE

The FXSAVE instruction takes a single memory operand, which is an FXSAVE area. The instruction stores x87 state and SSE state to the FXSAVE area. See Section 10.5.1.1 and Section 10.5.1.2 for details regarding mode-specific operation and operation determined by instruction prefixes.

### 10.5.3 Operation of FXRSTOR

The FXRSTOR instruction takes a single memory operand, which is an FXSAVE area. If the value at bytes 27:24 of the FXSAVE area is not a legal value for the MXCSR register (e.g., the value sets reserved bits). Otherwise, the instruction loads x87 state and SSE state from the FXSAVE area. See Section 10.5.1.1 and Section 10.5.1.2 for details regarding mode-specific operation and operation determined by instruction prefixes.

## 10.6 HANDLING SSE INSTRUCTION EXCEPTIONS

See Section 11.5, “SSE, SSE2, and SSE3 Exceptions,” for a detailed discussion of the general and SIMD floating-point exceptions that can be generated with the SSE instructions and for guidelines for handling these exceptions when they occur.

## 10.7 WRITING APPLICATIONS WITH THE SSE EXTENSIONS

See Section 11.6, “Writing Applications with SSE/SSE2 Extensions,” for additional information about writing applications and operating-system code using the SSE extensions.



## 2. Updates to Chapter 11, Volume 1

Change bars show changes to Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

-----  
Change to this chapter: Minor updates to clarify MFENCE/SFENCE/LFENCE operation.

The streaming SIMD extensions 2 (SSE2) were introduced into the IA-32 architecture in the Pentium 4 and Intel Xeon processors. These extensions enhance the performance of IA-32 processors for advanced 3-D graphics, video decoding/encoding, speech recognition, E-commerce, Internet, scientific, and engineering applications.

This chapter describes the SSE2 extensions and provides information to assist in writing application programs that use these and the SSE extensions.

### 11.1 OVERVIEW OF SSE2 EXTENSIONS

SSE2 extensions use the single instruction multiple data (SIMD) execution model that is used with MMX technology and SSE extensions. They extend this model with support for packed double-precision floating-point values and for 128-bit packed integers.

If `CPUID.01H:EDX.SSE2[bit 26] = 1`, SSE2 extensions are present.

SSE2 extensions add the following features to the IA-32 architecture, while maintaining backward compatibility with all existing IA-32 processors, applications and operating systems.

- Six data types:
  - 128-bit packed double-precision floating-point (two IEEE Standard 754 double-precision floating-point values packed into a double quadword)
  - 128-bit packed byte integers
  - 128-bit packed word integers
  - 128-bit packed doubleword integers
  - 128-bit packed quadword integers
- Instructions to support the additional data types and extend existing SIMD integer operations:
  - Packed and scalar double-precision floating-point instructions
  - Additional 64-bit and 128-bit SIMD integer instructions
  - 128-bit versions of SIMD integer instructions introduced with the MMX technology and the SSE extensions
  - Additional cacheability-control and instruction-ordering instructions
- Modifications to existing IA-32 instructions to support SSE2 features:
  - Extensions and modifications to the CPUID instruction
  - Modifications to the RDPMC instruction

These new features extend the IA-32 architecture's SIMD programming model in three important ways:

- They provide the ability to perform SIMD operations on pairs of packed double-precision floating-point values. This permits higher precision computations to be carried out in XMM registers, which enhances processor performance in scientific and engineering applications and in applications that use advanced 3-D geometry techniques (such as ray tracing). Additional flexibility is provided with instructions that operate on single (scalar) double-precision floating-point values located in the low quadword of an XMM register.
- They provide the ability to operate on 128-bit packed integers (bytes, words, doublewords, and quadwords) in XMM registers. This provides greater flexibility and greater throughput when performing SIMD operations on packed integers. The capability is particularly useful for applications such as RSA authentication and RC5 encryption. Using the full set of SIMD registers, data types, and instructions provided with the MMX technology and SSE/SSE2 extensions, programmers can develop algorithms that finely mix packed single- and double-precision floating-point data and 64- and 128-bit packed integer data.
- SSE2 extensions enhance the support introduced with SSE extensions for controlling the cacheability of SIMD data. SSE2 cache control instructions provide the ability to stream data in and out of the XMM registers without polluting the caches and the ability to prefetch data before it is actually used.

SSE2 extensions are fully compatible with all software written for IA-32 processors. All existing software continues to run correctly, without modification, on processors that incorporate SSE2 extensions, as well as in the presence of applications that incorporate these extensions. Enhancements to the CPUID instruction permit detection of the SSE2 extensions. Also, because the SSE2 extensions use the same registers as the SSE extensions, no new operating-system support is required for saving and restoring program state during a context switch beyond that provided for the SSE extensions.

SSE2 extensions are accessible from all IA-32 execution modes: protected mode, real address mode, virtual 8086 mode.

The following sections in this chapter describe the programming environment for SSE2 extensions including: the 128-bit XMM floating-point register set, data types, and SSE2 instructions. It also describes exceptions that can be generated with the SSE and SSE2 instructions and gives guidelines for writing applications with SSE and SSE2 extensions.

For additional information about SSE2 extensions, see:

- *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A & 2B*, provide a detailed description of individual SSE3 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 the SSE and SSE2 extensions into an operating-system environment.

## 11.2 SSE2 PROGRAMMING ENVIRONMENT

Figure 11-1 shows the programming environment for SSE2 extensions. No new registers or other instruction execution state are defined with SSE2 extensions. SSE2 instructions use the XMM registers, the MMX registers, and/or IA-32 general-purpose registers, as follows:

- **XMM registers** — These eight registers (see Figure 10-2) are used to operate on packed or scalar double-precision floating-point data. Scalar operations are operations performed on individual (unpacked) double-precision floating-point values stored in the low quadword of an XMM register. XMM registers are also used to perform operations on 128-bit packed integer data. They are referenced by the names XMM0 through XMM7.

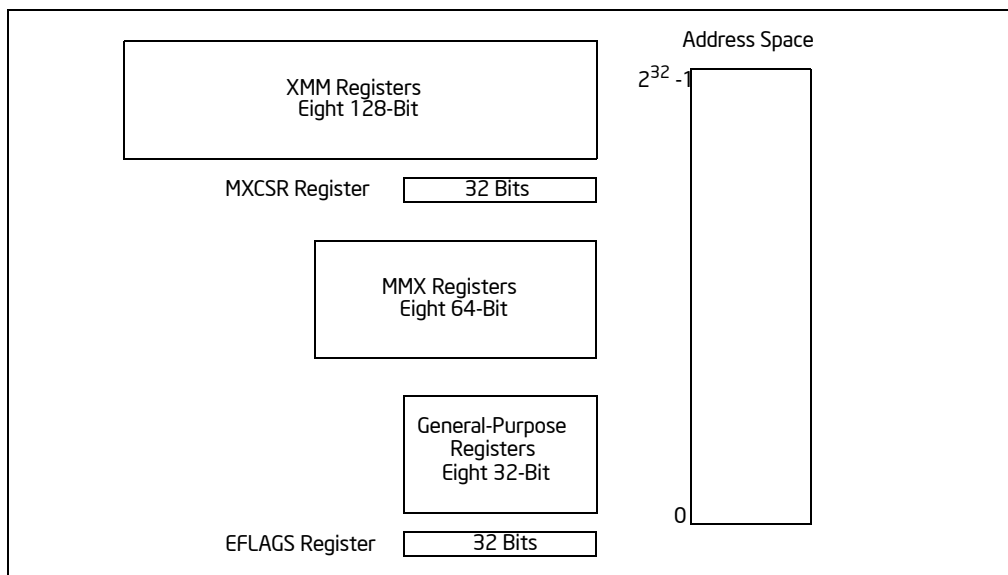


Figure 11-1. Streaming SIMD Extensions 2 Execution Environment

- **MXCSR register** — This 32-bit register (see Figure 10-3) provides status and control bits used in floating-point operations. The denormals-are-zeros and flush-to-zero flags in this register provide a higher performance alternative for the handling of denormal source operands and denormal (underflow) results. For more



information on the functions of these flags see Section 10.2.3.4, “Denormals-Are-Zeros,” and Section 10.2.3.3, “Flush-To-Zero.”

- **MMX registers** — These eight registers (see Figure 9-2) are used to perform operations on 64-bit packed integer data. They are also used to hold operands for some operations performed between MMX and XMM registers. MMX registers are referenced by the names MM0 through MM7.
- **General-purpose registers** — The eight general-purpose registers (see Figure 3-5) are used along with the existing IA-32 addressing modes to address operands in memory. MMX and XMM registers cannot be used to address memory. The general-purpose registers are also used to hold operands for some SSE2 instructions. These registers are referenced by the names EAX, EBX, ECX, EDX, EBP, ESI, EDI, and ESP.
- **EFLAGS register** — This 32-bit register (see Figure 3-8) is used to record the results of some compare operations.

### 11.2.1 SSE2 in 64-Bit Mode and Compatibility Mode

In compatibility mode, SSE2 extensions function like they do in protected mode. In 64-bit mode, eight additional XMM registers are accessible. Registers XMM8-XMM15 are accessed by using REX prefixes.

Memory operands are specified using the ModR/M, SIB encoding described in Section 3.7.5.

Some SSE2 instructions may be used to operate on general-purpose registers. Use the REX.W prefix to access 64-bit general-purpose registers. Note that if a REX prefix is used when it has no meaning, the prefix is ignored.

### 11.2.2 Compatibility of SSE2 Extensions with SSE, MMX Technology and x87 FPU Programming Environment

SSE2 extensions do not introduce any new state to the IA-32 execution environment beyond that of SSE. SSE2 extensions represent an enhancement of SSE extensions; they are fully compatible and share the same state information. SSE and SSE2 instructions can be executed together in the same instruction stream without the need to save state when switching between instruction sets.

XMM registers are independent of the x87 FPU and MMX registers; so SSE and SSE2 operations performed on XMM registers can be performed in parallel with x87 FPU or MMX technology operations (see Section 11.6.7, “Interaction of SSE/SSE2 Instructions with x87 FPU and MMX Instructions”).

The FXSAVE and FXRSTOR instructions save and restore the SSE and SSE2 states along with the x87 FPU and MMX states.

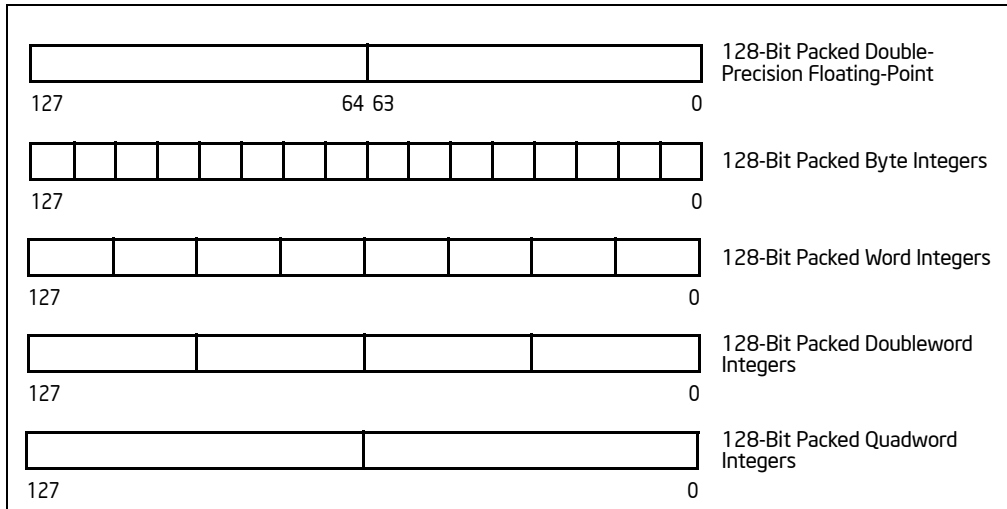
### 11.2.3 Denormals-Are-Zeros Flag

The denormals-are-zeros flag (bit 6 in the MXCSR register) was introduced into the IA-32 architecture with the SSE2 extensions. See Section 10.2.3.4, “Denormals-Are-Zeros,” for a description of this flag.

## 11.3 SSE2 DATA TYPES

SSE2 extensions introduced one 128-bit packed floating-point data type and four 128-bit SIMD integer data types to the IA-32 architecture (see Figure 11-2).

- **Packed double-precision floating-point** — This 128-bit data type consists of two IEEE 64-bit double-precision floating-point values packed into a double quadword. (See Figure 4-3 for the layout of a 64-bit double-precision floating-point value; refer to Section 4.2.2, “Floating-Point Data Types,” for a detailed description of double-precision floating-point values.)
- **128-bit packed integers** — The four 128-bit packed integer data types can contain 16 byte integers, 8 word integers, 4 doubleword integers, or 2 quadword integers. (Refer to Section 4.6.2, “128-Bit Packed SIMD Data Types,” for a detailed description of the 128-bit packed integers.)



**Figure 11-2. Data Types Introduced with the SSE2 Extensions**

All of these data types are operated on in XMM registers or memory. Instructions are provided to convert between these 128-bit data types and the 64-bit and 32-bit data types.

The address of a 128-bit packed memory operand must be aligned on a 16-byte boundary, except in the following cases:

- a MOVUPD instruction which supports unaligned accesses
- scalar instructions that use an 8-byte memory operand that is not subject to alignment requirements

Figure 4-2 shows the byte order of 128-bit (double quadword) and 64-bit (quadword) data types in memory.

## 11.4 SSE2 INSTRUCTIONS

The SSE2 instructions are divided into four functional groups:

- Packed and scalar double-precision floating-point instructions
- 64-bit and 128-bit SIMD integer instructions
- 128-bit extensions of SIMD integer instructions introduced with the MMX technology and the SSE extensions
- Cacheability-control and instruction-ordering instructions

The following sections provide more information about each group.

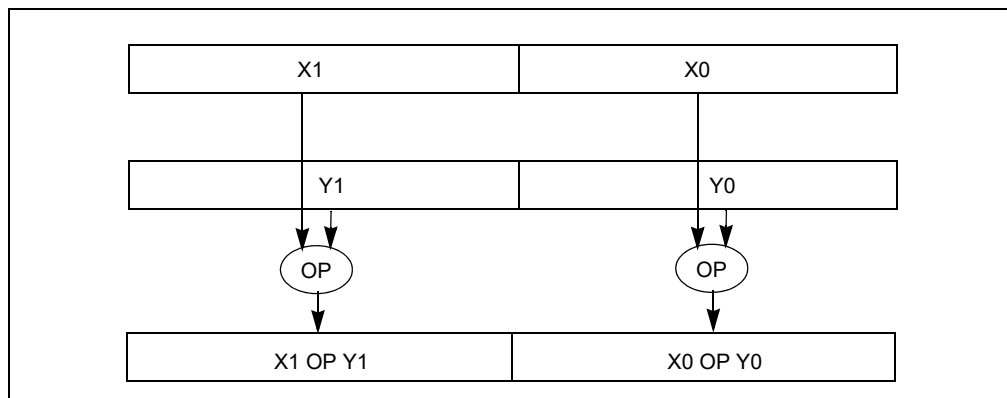
### 11.4.1 Packed and Scalar Double-Precision Floating-Point Instructions

The packed and scalar double-precision floating-point instructions are divided into the following sub-groups:

- Data movement instructions
- Arithmetic instructions
- Comparison instructions
- Conversion instructions
- Logical instructions
- Shuffle instructions

The packed double-precision floating-point instructions perform SIMD operations similarly to the packed single-precision floating-point instructions (see Figure 11-3). Each source operand contains two double-precision floating-

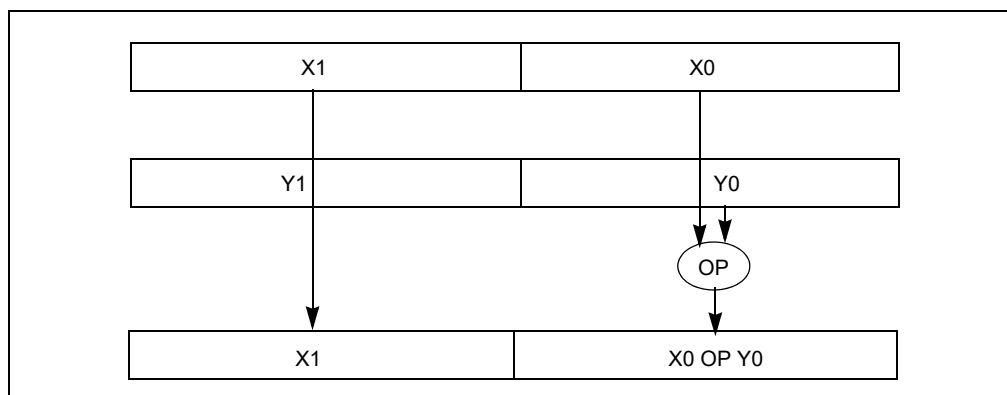
point values, and the destination operand contains the results of the operation (OP) performed in parallel on the corresponding values (X0 and Y0, and X1 and Y1) in each operand.



**Figure 11-3. Packed Double-Precision Floating-Point Operations**

The scalar double-precision floating-point instructions operate on the low (least significant) quadwords of two source operands (X0 and Y0), as shown in Figure 11-4. The high quadword (X1) of the first source operand is passed through to the destination. The scalar operations are similar to the floating-point operations performed in x87 FPU data registers with the precision control field in the x87 FPU control word set for double precision (53-bit significand), except that x87 stack operations use a 15-bit exponent range for the result while SSE2 operations use an 11-bit exponent range.

See Section 11.6.8, “Compatibility of SIMD and x87 FPU Floating-Point Data Types,” for more information about obtaining compatible results when performing both scalar double-precision floating-point operations in XMM registers and in x87 FPU data registers.



**Figure 11-4. Scalar Double-Precision Floating-Point Operations**

### 11.4.1.1 Data Movement Instructions

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

The MOVAPD (move aligned packed double-precision floating-point) instruction transfers a 128-bit packed double-precision floating-point operand from memory to an XMM register or vice versa, or between XMM registers. The memory address must be aligned to a 16-byte boundary; if not, a general-protection exception (GP#) is generated.

The MOVUPD (move unaligned packed double-precision floating-point) instruction transfers a 128-bit packed double-precision floating-point operand from memory to an XMM register or vice versa, or between XMM registers. Alignment of the memory address is not required.

The MOVSD (move scalar double-precision floating-point) instruction transfers a 64-bit double-precision floating-point operand from memory to the low quadword of an XMM register or vice versa, or between XMM registers. Alignment of the memory address is not required, unless alignment checking is enabled.

The MOVHPD (move high packed double-precision floating-point) instruction transfers a 64-bit double-precision floating-point operand from memory to the high quadword of an XMM register or vice versa. The low quadword of the register is left unchanged. Alignment of the memory address is not required, unless alignment checking is enabled.

The MOVLPD (move low packed double-precision floating-point) instruction transfers a 64-bit double-precision floating-point operand from memory to the low quadword of an XMM register or vice versa. The high quadword of the register is left unchanged. Alignment of the memory address is not required, unless alignment checking is enabled.

The MOVMSKPD (move packed double-precision floating-point mask) instruction extracts the sign bit of each of the two packed double-precision floating-point numbers in an XMM register and saves them in a general-purpose register. This 2-bit value can then be used as a condition to perform branching.

### 11.4.1.2 SSE2 Arithmetic Instructions

SSE2 arithmetic instructions perform addition, subtraction, multiply, divide, square root, and maximum/minimum operations on packed and scalar double-precision floating-point values.

The ADDPD (add packed double-precision floating-point values) and SUBPD (subtract packed double-precision floating-point values) instructions add and subtract, respectively, two packed double-precision floating-point operands.

The ADDSD (add scalar double-precision floating-point values) and SUBSD (subtract scalar double-precision floating-point values) instructions add and subtract, respectively, the low double-precision floating-point values of two operands and stores the result in the low quadword of the destination operand.

The MULPD (multiply packed double-precision floating-point values) instruction multiplies two packed double-precision floating-point operands.

The MULSD (multiply scalar double-precision floating-point values) instruction multiplies the low double-precision floating-point values of two operands and stores the result in the low quadword of the destination operand.

The DIVPD (divide packed double-precision floating-point values) instruction divides two packed double-precision floating-point operands.

The DIVSD (divide scalar double-precision floating-point values) instruction divides the low double-precision floating-point values of two operands and stores the result in the low quadword of the destination operand.

The SQRTPD (compute square roots of packed double-precision floating-point values) instruction computes the square roots of the values in a packed double-precision floating-point operand.

The SQRTSD (compute square root of scalar double-precision floating-point values) instruction computes the square root of the low double-precision floating-point value in the source operand and stores the result in the low quadword of the destination operand.

The MAXPD (return maximum of packed double-precision floating-point values) instruction compares the corresponding values in two packed double-precision floating-point operands and returns the numerically greater value from each comparison to the destination operand.

The MAXSD (return maximum of scalar double-precision floating-point values) instruction compares the low double-precision floating-point values from two packed double-precision floating-point operands and returns the numerically higher value from the comparison to the low quadword of the destination operand.

The MINPD (return minimum of packed double-precision floating-point values) instruction compares the corresponding values from two packed double-precision floating-point operands and returns the numerically lesser value from each comparison to the destination operand.

The `MINSD` (return minimum of scalar double-precision floating-point values) instruction compares the low values from two packed double-precision floating-point operands and returns the numerically lesser value from the comparison to the low quadword of the destination operand.

### 11.4.1.3 SSE2 Logical Instructions

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

The `ANDPD` (bitwise logical AND of packed double-precision floating-point values) instruction returns the logical AND of two packed double-precision floating-point operands.

The `ANDNPD` (bitwise logical AND NOT of packed double-precision floating-point values) instruction returns the logical AND NOT of two packed double-precision floating-point operands.

The `ORPD` (bitwise logical OR of packed double-precision floating-point values) instruction returns the logical OR of two packed double-precision floating-point operands.

The `XORPD` (bitwise logical XOR of packed double-precision floating-point values) instruction returns the logical XOR of two packed double-precision floating-point operands.

### 11.4.1.4 SSE2 Comparison Instructions

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

The `CMPPD` (compare packed double-precision floating-point values) instruction compares the corresponding values from two packed double-precision floating-point operands, using an immediate operand as a predicate, and returns a 64-bit mask result of all 1s or all 0s for each comparison to the destination operand. The value of the immediate operand allows the selection of any of eight compare conditions: equal, less than, less than equal, unordered, not equal, not less than, not less than or equal, or ordered.

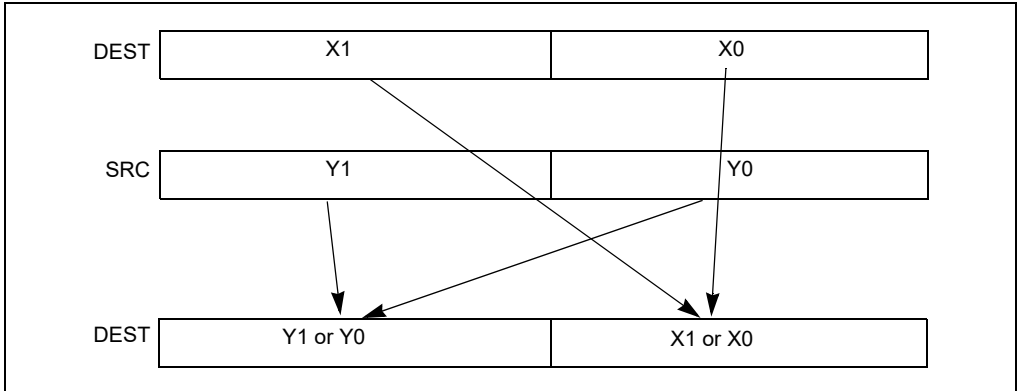
The `CMPSD` (compare scalar double-precision floating-point values) instruction compares the low values from two packed double-precision floating-point operands, using an immediate operand as a predicate, and returns a 64-bit mask result of all 1s or all 0s for the comparison to the low quadword of the destination operand. The immediate operand selects the compare condition as with the `CMPPD` instruction.

The `COMISD` (compare scalar double-precision floating-point values and set `EFLAGS`) and `UCOMISD` (unordered compare scalar double-precision floating-point values and set `EFLAGS`) instructions compare the low values of two packed double-precision floating-point operands and set the `ZF`, `PF`, and `CF` flags in the `EFLAGS` register to show the result (greater than, less than, equal, or unordered). These two instructions differ as follows: the `COMISD` instruction signals a floating-point invalid-operation (`#I`) exception when a source operand is either a `QNaN` or an `SNaN`; the `UCOMISD` instruction only signals an invalid-operation exception when a source operand is an `SNaN`.

### 11.4.1.5 SSE2 Shuffle and Unpack Instructions

SSE2 shuffle instructions shuffle the contents of two packed double-precision floating-point values and store the results in the destination operand.

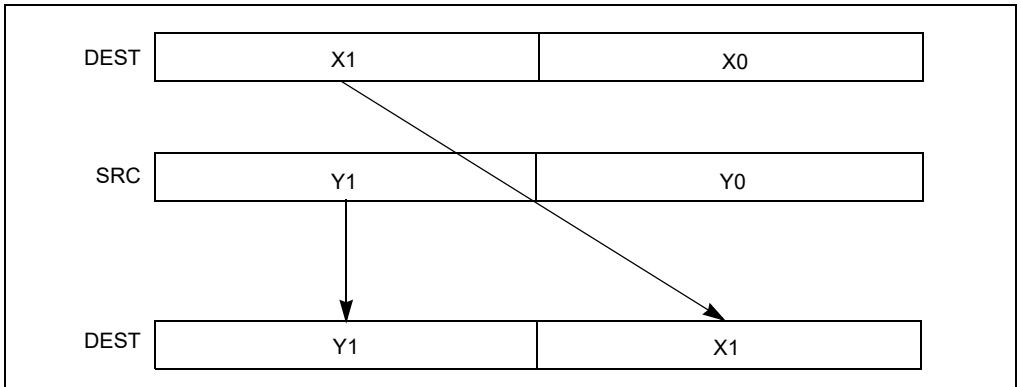
The `SHUFPS` (shuffle packed double-precision floating-point values) instruction places either of the two packed double-precision floating-point values from the destination operand in the low quadword of the destination operand, and places either of the two packed double-precision floating-point values from source operand in the high quadword of the destination operand (see Figure 11-5). By using the same register for the source and destination operands, the `SHUFPS` instruction can swap two packed double-precision floating-point values.



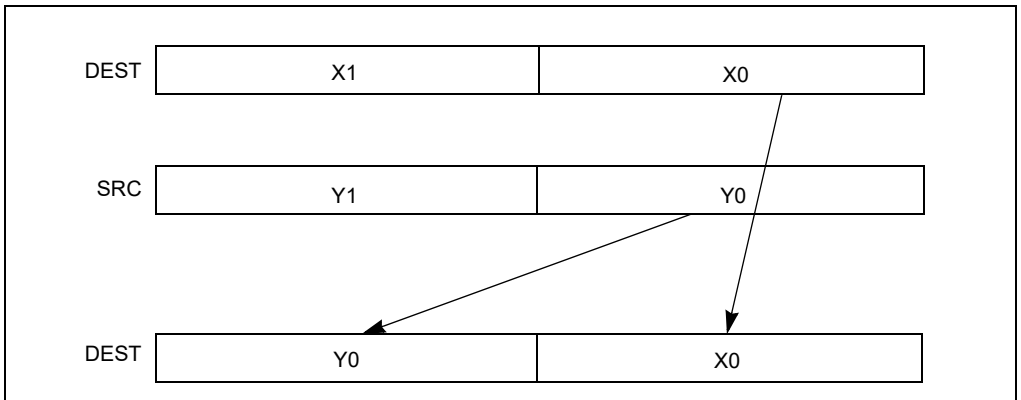
**Figure 11-5. SHUFPS Instruction, Packed Shuffle Operation**

The UNPCKHPD (unpack and interleave high packed double-precision floating-point values) instruction performs an interleaved unpack of the high values from the source and destination operands and stores the result in the destination operand (see Figure 11-6).

The UNPCKLPD (unpack and interleave low packed double-precision floating-point values) instruction performs an interleaved unpack of the low values from the source and destination operands and stores the result in the destination operand (see Figure 11-7).



**Figure 11-6. UNPCKHPD Instruction, High Unpack and Interleave Operation**



**Figure 11-7. UNPCKLPD Instruction, Low Unpack and Interleave Operation**

### 11.4.1.6 SSE2 Conversion Instructions

SSE2 conversion instructions (see Figure 11-8) support packed and scalar conversions between:

- Double-precision and single-precision floating-point formats
- Double-precision floating-point and doubleword integer formats
- Single-precision floating-point and doubleword integer formats

**Conversion between double-precision and single-precision floating-points values** — The following instructions convert operands between double-precision and single-precision floating-point formats. The operands being operated on are contained in XMM registers or memory (at most, one operand can reside in memory; the destination is always an MMX register).

The CVTSP2PD (convert packed single-precision floating-point values to packed double-precision floating-point values) instruction converts two packed single-precision floating-point values to two double-precision floating-point values.

The CVTPD2PS (convert packed double-precision floating-point values to packed single-precision floating-point values) instruction converts two packed double-precision floating-point values to two single-precision floating-point values. When a conversion is inexact, the result is rounded according to the rounding mode selected in the MXCSR register.

The CVTSS2SD (convert scalar single-precision floating-point value to scalar double-precision floating-point value) instruction converts a single-precision floating-point value to a double-precision floating-point value.

The CVTSD2SS (convert scalar double-precision floating-point value to scalar single-precision floating-point value) instruction converts a double-precision floating-point value to a single-precision floating-point value. When the conversion is inexact, the result is rounded according to the rounding mode selected in the MXCSR register.

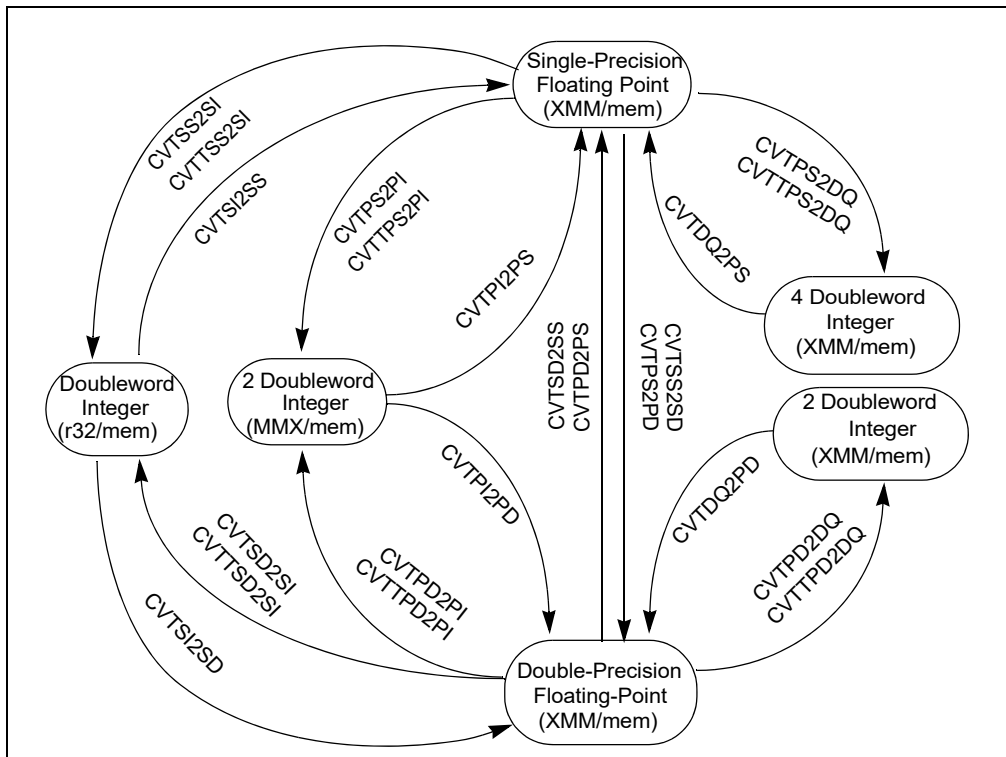


Figure 11-8. SSE and SSE2 Conversion Instructions

**Conversion between double-precision floating-point values and doubleword integers** — The following instructions convert operands between double-precision floating-point and doubleword integer formats. Operands

are housed in XMM registers, MMX registers, general registers or memory (at most one operand can reside in memory; the destination is always an XMM, MMX, or general register).

The CVTPD2PI (convert packed double-precision floating-point values to packed doubleword integers) instruction converts two packed double-precision floating-point numbers to two packed signed doubleword integers, with the result stored in an MMX register. When rounding to an integer value, the source value is rounded according to the rounding mode in the MXCSR register. The CVTTPD2PI (convert with truncation packed double-precision floating-point values to packed doubleword integers) instruction is similar to the CVTPD2PI instruction except that truncation is used to round a source value to an integer value (see Section 4.8.4.2, “Truncation with SSE and SSE2 Conversion Instructions”).

The CVTPI2PD (convert packed doubleword integers to packed double-precision floating-point values) instruction converts two packed signed doubleword integers to two double-precision floating-point values.

The CVTPD2DQ (convert packed double-precision floating-point values to packed doubleword integers) instruction converts two packed double-precision floating-point numbers to two packed signed doubleword integers, with the result stored in the low quadword of an XMM register. When rounding an integer value, the source value is rounded according to the rounding mode selected in the MXCSR register. The CVTTPD2DQ (convert with truncation packed double-precision floating-point values to packed doubleword integers) instruction is similar to the CVTPD2DQ instruction except that truncation is used to round a source value to an integer value (see Section 4.8.4.2, “Truncation with SSE and SSE2 Conversion Instructions”).

The CVTDQ2PD (convert packed doubleword integers to packed double-precision floating-point values) instruction converts two packed signed doubleword integers located in the low-order doublewords of an XMM register to two double-precision floating-point values.

The CVTSD2SI (convert scalar double-precision floating-point value to doubleword integer) instruction converts a double-precision floating-point value to a doubleword integer, and stores the result in a general-purpose register. When rounding an integer value, the source value is rounded according to the rounding mode selected in the MXCSR register. The CVTSSD2SI (convert with truncation scalar double-precision floating-point value to doubleword integer) instruction is similar to the CVTSD2SI instruction except that truncation is used to round the source value to an integer value (see Section 4.8.4.2, “Truncation with SSE and SSE2 Conversion Instructions”).

The CVTSI2SD (convert doubleword integer to scalar double-precision floating-point value) instruction converts a signed doubleword integer in a general-purpose register to a double-precision floating-point number, and stores the result in an XMM register.

**Conversion between single-precision floating-point and doubleword integer formats** — These instructions convert between packed single-precision floating-point and packed doubleword integer formats. Operands are housed in XMM registers, MMX registers, general registers, or memory (the latter for at most one source operand). The destination is always an XMM, MMX, or general register. These SSE2 instructions supplement conversion instructions (CVTPI2PS, CVTPS2PI, CVTTPS2PI, CVTSI2SS, CVTSS2SI, and CVTSS2SI) introduced with SSE extensions.

The CVTPS2DQ (convert packed single-precision floating-point values to packed doubleword integers) instruction converts four packed single-precision floating-point values to four packed signed doubleword integers, with the source and destination operands in XMM registers or memory (the latter for at most one source operand). When the conversion is inexact, the rounded value according to the rounding mode selected in the MXCSR register is returned. The CVTTPS2DQ (convert with truncation packed single-precision floating-point values to packed doubleword integers) instruction is similar to the CVTPS2DQ instruction except that truncation is used to round a source value to an integer value (see Section 4.8.4.2, “Truncation with SSE and SSE2 Conversion Instructions”).

The CVTDQ2PS (convert packed doubleword integers to packed single-precision floating-point values) instruction converts four packed signed doubleword integers to four packed single-precision floating-point numbers, with the source and destination operands in XMM registers or memory (the latter for at most one source operand). When the conversion is inexact, the rounded value according to the rounding mode selected in the MXCSR register is returned.

## 11.4.2 SSE2 64-Bit and 128-Bit SIMD Integer Instructions

SSE2 extensions add several 128-bit packed integer instructions to the IA-32 architecture. Where appropriate, a 64-bit version of each of these instructions is also provided. The 128-bit versions of instructions operate on data in XMM registers; 64-bit versions operate on data in MMX registers. The instructions follow.



The MOVDQA (move aligned double quadword) instruction transfers a double quadword operand from memory to an XMM register or vice versa; or between XMM registers. The memory address must be aligned to a 16-byte boundary; otherwise, a general-protection exception (#GP) is generated.

The MOVDQU (move unaligned double quadword) instruction performs the same operations as the MOVDQA instruction, except that 16-byte alignment of a memory address is not required.

The PADDQ (packed quadword add) instruction adds two packed quadword integer operands or two single quadword integer operands, and stores the results in an XMM or MMX register, respectively. This instruction can operate on either unsigned or signed (two's complement notation) integer operands.

The PSUBQ (packed quadword subtract) instruction subtracts two packed quadword integer operands or two single quadword integer operands, and stores the results in an XMM or MMX register, respectively. Like the PADDQ instruction, PSUBQ can operate on either unsigned or signed (two's complement notation) integer operands.

The PMULUDQ (multiply packed unsigned doubleword integers) instruction performs an unsigned multiply of unsigned doubleword integers and returns a quadword result. Both 64-bit and 128-bit versions of this instruction are available. The 64-bit version operates on two doubleword integers stored in the low doubleword of each source operand, and the quadword result is returned to an MMX register. The 128-bit version performs a packed multiply of two pairs of doubleword integers. Here, the doublewords are packed in the first and third doublewords of the source operands, and the quadword results are stored in the low and high quadwords of an XMM register.

The PSHUFLW (shuffle packed low words) instruction shuffles the word integers packed into the low quadword of the source operand and stores the shuffled result in the low quadword of the destination operand. An 8-bit immediate operand specifies the shuffle order.

The PSHUFW (shuffle packed high words) instruction shuffles the word integers packed into the high quadword of the source operand and stores the shuffled result in the high quadword of the destination operand. An 8-bit immediate operand specifies the shuffle order.

The PSHUFD (shuffle packed doubleword integers) instruction shuffles the doubleword integers packed into the source operand and stores the shuffled result in the destination operand. An 8-bit immediate operand specifies the shuffle order.

The PSLLDQ (shift double quadword left logical) instruction shifts the contents of the source operand to the left by the amount of bytes specified by an immediate operand. The empty low-order bytes are cleared (set to 0).

The PSRLDQ (shift double quadword right logical) instruction shifts the contents of the source operand to the right by the amount of bytes specified by an immediate operand. The empty high-order bytes are cleared (set to 0).

The PUNPCKHQDQ (Unpack high quadwords) instruction interleaves the high quadword of the source operand and the high quadword of the destination operand and writes them to the destination register.

The PUNPCKLQDQ (Unpack low quadwords) instruction interleaves the low quadwords of the source operand and the low quadwords of the destination operand and writes them to the destination register.

Two additional SSE instructions enable data movement from the MMX registers to the XMM registers.

The MOVQ2DQ (move quadword integer from MMX to XMM registers) instruction moves the quadword integer from an MMX source register to an XMM destination register.

The MOVDQ2Q (move quadword integer from XMM to MMX registers) instruction moves the low quadword integer from an XMM source register to an MMX destination register.

### 11.4.3 128-Bit SIMD Integer Instruction Extensions

All of 64-bit SIMD integer instructions introduced with MMX technology and SSE extensions (with the exception of the PSHUFW instruction) have been extended by SSE2 extensions to operate on 128-bit packed integer operands located in XMM registers. The 128-bit versions of these instructions follow the same SIMD conventions regarding packed operands as the 64-bit versions. For example, where the 64-bit version of the PADDQ instruction operates on 8 packed bytes, the 128-bit version operates on 16 packed bytes.

## 11.4.4 Cacheability Control and Memory Ordering Instructions

SSE2 extensions that give programs more control over the caching, loading, and storing of data. are described below.

### 11.4.4.1 FLUSH Cache Line

The CLFLUSH (flush cache line) instruction writes and invalidates the cache line associated with a specified linear address. The invalidation is for all levels of the processor's cache hierarchy, and it is broadcast throughout the cache coherency domain.

#### NOTE

CLFLUSH was introduced with the SSE2 extensions. However, the instruction can be implemented in IA-32 processors that do not implement the SSE2 extensions. Detect CLFLUSH using the feature bit (if CPUID.01H:EDX.CLFSH[bit 19] = 1).

### 11.4.4.2 Cacheability Control Instructions

The following four instructions enable data from XMM and general-purpose registers to be stored to memory using a non-temporal hint. The non-temporal hint directs the processor to store data to memory without writing the data into the cache hierarchy. See Section 10.4.6.2, "Caching of Temporal vs. Non-Temporal Data," for more information about non-temporal stores and hints.

The MOVNTDQ (store double quadword using non-temporal hint) instruction stores packed integer data from an XMM register to memory, using a non-temporal hint.

The MOVNTPD (store packed double-precision floating-point values using non-temporal hint) instruction stores packed double-precision floating-point data from an XMM register to memory, using a non-temporal hint.

The MOVNTI (store doubleword using non-temporal hint) instruction stores integer data from a general-purpose register to memory, using a non-temporal hint.

The MASKMOVDQU (store selected bytes of double quadword) instruction stores selected byte integers from an XMM register to memory, using a byte mask to selectively write the individual bytes. The memory location does not need to be aligned on a natural boundary. This instruction also uses a non-temporal hint.

### 11.4.4.3 Memory Ordering Instructions

SSE2 extensions introduce two new fence instructions (LFENCE and MFENCE) as companions to the SFENCE instruction introduced with SSE extensions.

The LFENCE instruction establishes a memory fence for loads. It guarantees ordering between two loads and prevents speculative loads from passing the load fence (that is, no speculative loads are allowed until all loads specified before the load fence have been carried out).

The MFENCE instruction establishes a memory fence for both loads and stores. The processor ensures that no load or store after MFENCE will execute until all loads and stores before MFENCE are globally visible.<sup>1</sup> Note that the sequences LFENCE;SFENCE and SFENCE;LFENCE are not equivalent to MFENCE because neither ensures that older stores are globally observed prior to younger loads.

### 11.4.4.4 Pause

The PAUSE instruction is provided to improve the performance of "spin-wait loops" executed on a Pentium 4 or Intel Xeon processor. On a Pentium 4 processor, it also provides the added benefit of reducing processor power consumption while executing a spin-wait loop. It is recommended that a PAUSE instruction always be included in the code sequence for a spin-wait loop.

---

1. A load is considered to become globally visible when the value to be loaded is determined

## 11.4.5 Branch Hints

SSE2 extensions designate two instruction prefixes (2EH and 3EH) to provide branch hints to the processor (see “Instruction Prefixes” in Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*). These prefixes can only be used with the *Jcc* instruction and only at the machine code level (that is, there are no mnemonics for the branch hints).

## 11.5 SSE, SSE2, AND SSE3 EXCEPTIONS

SSE/SSE2/SSE3 extensions generate two general types of exceptions:

- Non-numeric exceptions
- SIMD floating-point exceptions<sup>1</sup>

SSE/SSE2/SSE3 instructions can generate the same type of memory-access and non-numeric exceptions as other IA-32 architecture instructions. Existing exception handlers can generally handle these exceptions without any code modification. See “Providing Non-Numeric Exception Handlers for Exceptions Generated by the SSE, SSE2 and SSE3 Instructions” in Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*, for a list of the non-numeric exceptions that can be generated by SSE/SSE2/SSE3 instructions and for guidelines for handling these exceptions.

SSE/SSE2/SSE3 instructions do not generate numeric exceptions on packed integer operations; however, they can generate numeric (SIMD floating-point) exceptions on packed single-precision and double-precision floating-point operations. These SIMD floating-point exceptions are defined in the IEEE Standard 754 for Binary Floating-Point Arithmetic and are the same exceptions that are generated for x87 FPU instructions. See Section 11.5.1, “SIMD Floating-Point Exceptions,” for a description of these exceptions.

### 11.5.1 SIMD Floating-Point Exceptions

SIMD floating-point exceptions are those exceptions that can be generated by SSE/SSE2/SSE3 instructions that operate on packed or scalar floating-point operands.

Six classes of SIMD floating-point exceptions can be generated:

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

All of these exceptions (except the denormal operand exception) are defined in IEEE Standard 754, and they are the same exceptions that are generated with the x87 floating-point instructions. Section 4.9, “Overview of Floating-Point Exceptions,” gives a detailed description of these exceptions and of how and when they are generated. The following sections discuss the implementation of these exceptions in SSE/SSE2/SSE3 extensions.

All SIMD floating-point exceptions are precise and occur as soon as the instruction completes execution.

Each of the six exception conditions has a corresponding flag (IE, DE, ZE, OE, UE, and PE) and mask bit (IM, DM, ZM, OM, UM, and PM) in the MXCSR register (see Figure 10-3). The mask bits can be set with the LDMXCSR or FXRSTOR instruction; the mask and flag bits can be read with the STMXCSR or FXSAVE instruction.

The OSXMMEXCEPT flag (bit 10) of control register CR4 provides additional control over generation of SIMD floating-point exceptions by allowing the operating system to indicate whether or not it supports software exception handlers for SIMD floating-point exceptions. If an unmasked SIMD floating-point exception is generated and the OSXMMEXCEPT flag is set, the processor invokes a software exception handler by generating a SIMD floating-

1. The FISTTP instruction in SSE3 does not generate SIMD floating-point exceptions, but it can generate x87 FPU floating-point exceptions.





### 11.5.2.5 Numeric Underflow Exception (#U)

The processor reports a numeric underflow exception whenever the magnitude of the rounded result of an arithmetic instruction, with unbounded exponent, is less than the smallest possible normalized, finite value that will fit in the destination operand and the numeric-underflow exception is not masked. If the numeric underflow exception is masked, both underflow and the inexact-result condition must be detected before numeric underflow is reported. This exception can be generated with the ADDPS, ADDSS, ADDPD, ADDSD, SUBPS, SUBSS, SUBPD, SUBSD, MULPS, MULSS, MULPD, MULSD, DIVPS, DIVSS, DIVPD, DIVSD, CVTSD2SS, CVTSD2PS, ADDSUBPD, ADDSUBPS, HADDPD, HADDPS, HSUBPD, and HSUBPS instructions. The flag (UE) and mask (UM) bits for the numeric underflow exception are bits 4 and 11, respectively, in the MXCSR register.

The flush-to-zero flag (bit 15) of the MXCSR register provides an additional option for handling numeric underflow exceptions. When this flag is set and the numeric underflow exception is masked, tiny results are returned as a zero with the sign of the true result (see Section 10.2.3.3, "Flush-To-Zero").

Underflow will occur when a tiny non-zero result is detected (the result has to be also inexact if underflow exceptions are masked), as described in the IEEE Standard 754-2008. While DAZ does not affect the rules for signaling IEEE exceptions, operations on denormal inputs might have different results when DAZ=1. As a consequence, DAZ can have an effect on the floating-point exceptions - including the underflow exception - when observed for a given operation involving denormal inputs.

See Section 4.9.1.5, "Numeric Underflow Exception (#U)," for more information about the numeric underflow exception. See Section 11.5.4, "Handling SIMD Floating-Point Exceptions in Software," for information on handling unmasked exceptions.

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

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

The flag (PE) and mask (PM) bits for the inexact-result exception are bits 2 and 12, respectively, in the MXCSR register.

See Section 4.9.1.6, "Inexact-Result (Precision) Exception (#P)," for more information about the inexact-result exception. See Section 11.5.4, "Handling SIMD Floating-Point Exceptions in Software," for information on handling unmasked exceptions.

In flush-to-zero mode, the inexact result exception is reported.

## 11.5.3 Generating SIMD Floating-Point Exceptions

When the processor executes a packed or scalar floating-point instruction, it looks for and reports on SIMD floating-point exception conditions using two sequential steps:

1. Looks for, reports on, and handles pre-computation exception conditions (invalid-operand, divide-by-zero, and denormal operand)
2. Looks for, reports on, and handles post-computation exception conditions (numeric overflow, numeric underflow, and inexact result)

If both pre- and post-computational exceptions are unmasked, it is possible for the processor to generate a SIMD floating-point exception (#XM) twice during the execution of an SSE, SSE2 or SSE3 instruction: once when it detects and handles a pre-computational exception and when it detects a post-computational exception.

### 11.5.3.1 Handling Masked Exceptions

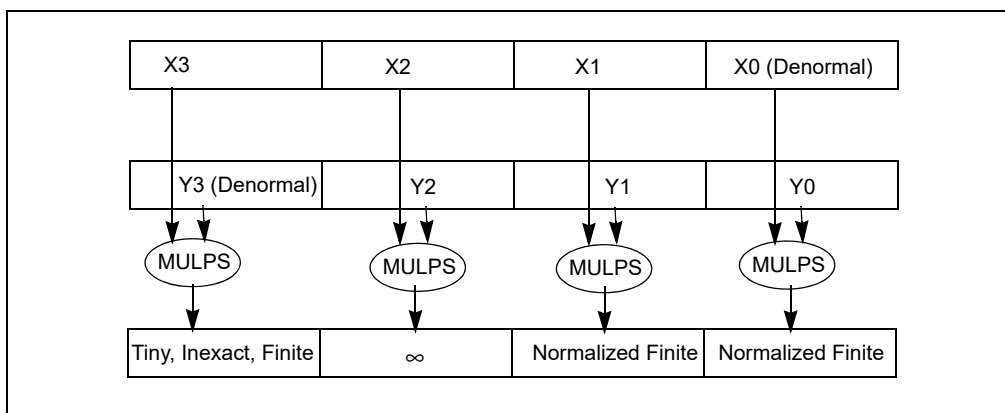
If all exceptions are masked, the processor handles the exceptions it detects by placing the masked result (or results for packed operands) in a destination operand and continuing program execution. The masked result may be a rounded normalized value, signed infinity, a denormal finite number, zero, a QNaN floating-point indefinite, or



a QNaN depending on the exception condition detected. In most cases, the corresponding exception flag bit in MXCSR is also set. The one situation where an exception flag is not set is when an underflow condition is detected and it is not accompanied by an inexact result.

When operating on packed floating-point operands, the processor returns a masked result for each of the sub-operand computations and sets a separate set of internal exception flags for each computation. It then performs a logical-OR on the internal exception flag settings and sets the exception flags in the MXCSR register according to the results of OR operations.

For example, Figure 11-9 shows the results of an MULPS instruction. In the example, all SIMD floating-point exceptions are masked. Assume that a denormal exception condition is detected prior to the multiplication of sub-operands X0 and Y0, no exception condition is detected for the multiplication of X1 and Y1, a numeric overflow exception condition is detected for the multiplication of X2 and Y2, and another denormal exception is detected prior to the multiplication of sub-operands X3 and Y3. Because denormal exceptions are masked, the processor uses the denormal source values in the multiplications of (X0 and Y0) and of (X3 and Y3) passing the results of the multiplications through to the destination operand. With the denormal operand, the result of the X0 and Y0 computation is a normalized finite value, with no exceptions detected. However, the X3 and Y3 computation produces a tiny and inexact result. This causes the corresponding internal numeric underflow and inexact-result exception flags to be set.



**Figure 11-9. Example Masked Response for Packed Operations**

For the multiplication of X2 and Y2, the processor stores the floating-point  $\infty$  in the destination operand, and sets the corresponding internal sub-operand numeric overflow flag. The result of the X1 and Y1 multiplication is passed through to the destination operand, with no internal sub-operand exception flags being set. Following the computations, the individual sub-operand exceptions flags for denormal operand, numeric underflow, inexact result, and numeric overflow are OR'd and the corresponding flags are set in the MXCSR register.

The net result of this computation is that:

- Multiplication of X0 and Y0 produces a normalized finite result
- Multiplication of X1 and Y1 produces a normalized finite result
- Multiplication of X2 and Y2 produces a floating-point  $\infty$  result
- Multiplication of X3 and Y3 produces a tiny, inexact, finite result
- Denormal operand, numeric underflow, numeric underflow, and inexact result flags are set in the MXCSR register

### 11.5.3.2 Handling Unmasked Exceptions

If all exceptions are unmasked, the processor:

1. First detects any pre-computation exceptions: it ORs those exceptions, sets the appropriate exception flags, leaves the source and destination operands unaltered, and goes to step 2. If it does not detect any pre-computation exceptions, it goes to step 5.

2. Checks CR4.OSXMMEXCPT[bit 10]. If this flag is set, the processor goes to step 3; if the flag is clear, it generates an invalid-opcode exception (#UD) and makes an implicit call to the invalid-opcode exception handler.
3. Generates a SIMD floating-point exception (#XM) and makes an implicit call to the SIMD floating-point exception handler.
4. If the exception handler is able to fix the source operands that generated the pre-computation exceptions or mask the condition in such a way as to allow the processor to continue executing the instruction, the processor resumes instruction execution as described in step 5.
5. Upon returning from the exception handler (or if no pre-computation exceptions were detected), the processor checks for post-computation exceptions. If the processor detects any post-computation exceptions: it ORs those exceptions, sets the appropriate exception flags, leaves the source and destination operands unaltered, and repeats steps 2, 3, and 4.
6. Upon returning from the exceptions handler in step 4 (or if no post-computation exceptions were detected), the processor completes the execution of the instruction.

The implication of this procedure is that for unmasked exceptions, the processor can generate a SIMD floating-point exception (#XM) twice: once if it detects pre-computation exception conditions and a second time if it detects post-computation exception conditions. For example, if SIMD floating-point exceptions are unmasked for the computation shown in Figure 11-9, the processor would generate one SIMD floating-point exception for denormal operand conditions and a second SIMD floating-point exception for overflow and underflow (no inexact result exception would be generated because the multiplications of X0 and Y0 and of X1 and Y1 are exact).

### 11.5.3.3 Handling Combinations of Masked and Unmasked Exceptions

In situations where both masked and unmasked exceptions are detected, the processor will set exception flags for the masked and the unmasked exceptions. However, it will not return masked results until after the processor has detected and handled unmasked post-computation exceptions and returned from the exception handler (as in step 6 above) to finish executing the instruction.

### 11.5.4 Handling SIMD Floating-Point Exceptions in Software

Section 4.9.3, “Typical Actions of a Floating-Point Exception Handler,” shows actions that may be carried out by a SIMD floating-point exception handler. The SSE/SSE2/SSE3 state is saved with the FXSAVE instruction (see Section 11.6.5, “Saving and Restoring the SSE/SSE2 State”).

### 11.5.5 Interaction of SIMD and x87 FPU Floating-Point Exceptions

SIMD floating-point exceptions are generated independently from x87 FPU floating-point exceptions. SIMD floating-point exceptions do not cause assertion of the FERR# pin (independent of the value of CR0.NE[bit 5]). They ignore the assertion and deassertion of the IGNNE# pin.

If applications use SSE/SSE2/SSE3 instructions along with x87 FPU instructions (in the same task or program), consider the following:

- SIMD floating-point exceptions are reported independently from the x87 FPU floating-point exceptions. SIMD and x87 FPU floating-point exceptions can be unmasked independently. Separate x87 FPU and SIMD floating-point exception handlers must be provided if the same exception is unmasked for x87 FPU and for SSE/SSE2/SSE3 operations.
- The rounding mode specified in the MXCSR register does not affect x87 FPU instructions. Likewise, the rounding mode specified in the x87 FPU control word does not affect the SSE/SSE2/SSE3 instructions. To use the same rounding mode, the rounding control bits in the MXCSR register and in the x87 FPU control word must be set explicitly to the same value.
- The flush-to-zero mode set in the MXCSR register for SSE/SSE2/SSE3 instructions has no counterpart in the x87 FPU. For compatibility with the x87 FPU, set the flush-to-zero bit to 0.



- The denormals-are-zeros mode set in the MXCSR register for SSE/SSE2/SSE3 instructions has no counterpart in the x87 FPU. For compatibility with the x87 FPU, set the denormals-are-zeros bit to 0.
- An application that expects to detect x87 FPU exceptions that occur during the execution of x87 FPU instructions will not be notified if exceptions occurs during the execution of corresponding SSE/SSE2/SSE3<sup>1</sup> instructions, unless the exception masks that are enabled in the x87 FPU control word have also been enabled in the MXCSR register and the application is capable of handling SIMD floating-point exceptions (#XM).
  - Masked exceptions that occur during an SSE/SSE2/SSE3 library call cannot be detected by unmasking the exceptions after the call (in an attempt to generate the fault based on the fact that an exception flag is set). A SIMD floating-point exception flag that is set when the corresponding exception is unmasked will not generate a fault; only the next occurrence of that unmasked exception will generate a fault.
  - An application which checks the x87 FPU status word to determine if any masked exception flags were set during an x87 FPU library call will also need to check the MXCSR register to detect a similar occurrence of a masked exception flag being set during an SSE/SSE2/SSE3 library call.

## 11.6 WRITING APPLICATIONS WITH SSE/SSE2 EXTENSIONS

The following sections give some guidelines for writing application programs and operating-system code that uses the SSE and SSE2 extensions. Because SSE and SSE2 extensions share the same state and perform companion operations, these guidelines apply to both sets of extensions.

*Chapter 13* in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, discusses the interface to the processor for context switching as well as other operating system considerations when writing code that uses SSE/SSE2/SSE3 extensions.

### 11.6.1 General Guidelines for Using SSE/SSE2 Extensions

The following guidelines describe how to take full advantage of the performance gains available with the SSE and SSE2 extensions:

- Ensure that the processor supports the SSE and SSE2 extensions.
- Ensure that your operating system supports the SSE and SSE2 extensions. (Operating system support for the SSE extensions implies support for SSE2 extension and vice versa.)
- Use stack and data alignment techniques to keep data properly aligned for efficient memory use.
- Use the non-temporal store instructions offered with the SSE and SSE2 extensions.
- Employ the optimization and scheduling techniques described in the *Intel Pentium 4 Optimization Reference Manual* (see Section 1.4, "Related Literature," for the order number for this manual).

### 11.6.2 Checking for SSE/SSE2 Support

Before an application attempts to use the SSE and/or SSE2 extensions, it should check that they are present on the processor:

1. Check that the processor supports the CPUID instruction. Bit 21 of the EFLAGS register can be used to check processor's support the CPUID instruction.
2. Check that the processor supports the SSE and/or SSE2 extensions (true if CPUID.01H:EDX.SSE[bit 25] = 1 and/or CPUID.01H:EDX.SSE2[bit 26] = 1).

Operating system must provide system level support for handling SSE state, exceptions before an application can use the SSE and/or SSE2 extensions (see *Chapter 13* in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).

---

1. SSE3 refers to ADDSUBPD, ADDSUBPS, HADDPD, HADDPS, HSUBPD and HSUBPS; the only other SSE3 instruction that can raise floating-point exceptions is FISTTP: it can generate x87 FPU invalid operation and inexact result exceptions.

If the processor attempts to execute an unsupported SSE or SSE2 instruction, the processor will generate an invalid-opcode exception (#UD). If an operating system did not provide adequate system level support for SSE, executing an SSE or SSE2 instructions can also generate #UD.

### 11.6.3 Checking for the DAZ Flag in the MXCSR Register

The denormals-are-zero flag in the MXCSR register is available in most of the Pentium 4 processors and in the Intel Xeon processor, with the exception of some early steppings. To check for the presence of the DAZ flag in the MXCSR register, do the following:

1. Establish a 512-byte FXSAVE area in memory.
2. Clear the FXSAVE area to all 0s.
3. Execute the FXSAVE instruction, using the address of the first byte of the cleared FXSAVE area as a source operand. See "FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State" in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*, for a description of the FXSAVE instruction and the layout of the FXSAVE image.
4. Check the value in the MXCSR\_MASK field in the FXSAVE image (bytes 28 through 31).
  - If the value of the MXCSR\_MASK field is 00000000H, the DAZ flag and denormals-are-zero mode are not supported.
  - If the value of the MXCSR\_MASK field is non-zero and bit 6 is set, the DAZ flag and denormals-are-zero mode are supported.

If the DAZ flag is not supported, then it is a reserved bit and attempting to write a 1 to it will cause a general-protection exception (#GP). See Section 11.6.6, "Guidelines for Writing to the MXCSR Register," for general guidelines for preventing general-protection exceptions when writing to the MXCSR register.

### 11.6.4 Initialization of SSE/SSE2 Extensions

The SSE and SSE2 state is contained in the XMM and MXCSR registers. Upon a hardware reset of the processor, this state is initialized as follows (see Table 11-2):

- All SIMD floating-point exceptions are masked (bits 7 through 12 of the MXCSR register is set to 1).
- All SIMD floating-point exception flags are cleared (bits 0 through 5 of the MXCSR register is set to 0).
- The rounding control is set to round-nearest (bits 13 and 14 of the MXCSR register are set to 00B).
- The flush-to-zero mode is disabled (bit 15 of the MXCSR register is set to 0).
- The denormals-are-zeros mode is disabled (bit 6 of the MXCSR register is set to 0). If the denormals-are-zeros mode is not supported, this bit is reserved and will be set to 0 on initialization.
- Each of the XMM registers is cleared (set to all zeros).

**Table 11-2. SSE and SSE2 State Following a Power-up/Reset or INIT**

Registers	Power-Up or Reset	INIT
XMM0 through XMM7	+0.0	Unchanged
MXCSR	1F80H	Unchanged

If the processor is reset by asserting the INIT# pin, the SSE and SSE2 state is not changed.

### 11.6.5 Saving and Restoring the SSE/SSE2 State

The FXSAVE instruction saves the x87 FPU, MMX, SSE and SSE2 states (which includes the contents of eight XMM registers and the MXCSR registers) in a 512-byte block of memory. The FXRSTOR instruction restores the saved SSE and SSE2 state from memory. See the FXSAVE instruction in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*, for the layout of the 512-byte state block.

In addition to saving and restoring the SSE and SSE2 state, FXSAVE and FXRSTOR also save and restore the x87 FPU state (because MMX registers are aliased to the x87 FPU data registers this includes saving and restoring the MMX state). For greater code efficiency, it is suggested that FXSAVE and FXRSTOR be substituted for the FSAVE, FNSAVE and FRSTOR instructions in the following situations:

- When a context switch is being made in a multitasking environment
- During calls and returns from interrupt and exception handlers

In situations where the code is switching between x87 FPU and MMX technology computations (without a context switch or a call to an interrupt or exception), the FSAVE/FNSAVE and FRSTOR instructions are more efficient than the FXSAVE and FXRSTOR instructions.

### 11.6.6 Guidelines for Writing to the MXCSR Register

The MXCSR has several reserved bits, and attempting to write a 1 to any of these bits will cause a general-protection exception (#GP) to be generated. To allow software to identify these reserved bits, the MXCSR\_MASK value is provided. Software can determine this mask value as follows:

1. Establish a 512-byte FXSAVE area in memory.
2. Clear the FXSAVE area to all 0s.
3. Execute the FXSAVE instruction, using the address of the first byte of the cleared FXSAVE area as a source operand. See “FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State” in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*, for a description of FXSAVE and the layout of the FXSAVE image.
4. Check the value in the MXCSR\_MASK field in the FXSAVE image (bytes 28 through 31).
  - If the value of the MXCSR\_MASK field is 00000000H, then the MXCSR\_MASK value is the default value of 0000FFBFH. Note that this value indicates that bit 6 of the MXCSR register is reserved; this setting indicates that the denormals-are-zero mode is not supported on the processor.
  - If the value of the MXCSR\_MASK field is non-zero, the MXCSR\_MASK value should be used as the MXCSR\_MASK.

All bits set to 0 in the MXCSR\_MASK value indicate reserved bits in the MXCSR register. Thus, if the MXCSR\_MASK value is AND’d with a value to be written into the MXCSR register, the resulting value will be assured of having all its reserved bits set to 0, preventing the possibility of a general-protection exception being generated when the value is written to the MXCSR register.

For example, the default MXCSR\_MASK value when 00000000H is returned in the FXSAVE image is 0000FFBFH. If software AND’s a value to be written to MXCSR register with 0000FFBFH, bit 6 of the result (the DAZ flag) will be ensured of being set to 0, which is the required setting to prevent general-protection exceptions on processors that do not support the denormals-are-zero mode.

To prevent general-protection exceptions, the MXCSR\_MASK value should be AND’d with the value to be written into the MXCSR register in the following situations:

- Operating system routines that receive a parameter from an application program and then write that value to the MXCSR register (either with an FXRSTOR or LDMXCSR instruction)
- Any application program that writes to the MXCSR register and that needs to run robustly on several different IA-32 processors

Note that all bits in the MXCSR\_MASK value that are set to 1 indicate features that are supported by the MXCSR register; they can be treated as feature flags for identifying processor capabilities.

### 11.6.7 Interaction of SSE/SSE2 Instructions with x87 FPU and MMX Instructions

The XMM registers and the x87 FPU and MMX registers represent separate execution environments, which has certain ramifications when executing SSE, SSE2, MMX, and x87 FPU instructions in the same code module or when mixing code modules that contain these instructions:

- Those SSE and SSE2 instructions that operate only on XMM registers (such as the packed and scalar floating-point instructions and the 128-bit SIMD integer instructions) in the same instruction stream with 64-bit SIMD integer or x87 FPU instructions without any restrictions. For example, an application can perform the majority of its floating-point computations in the XMM registers, using the packed and scalar floating-point instructions, and at the same time use the x87 FPU to perform trigonometric and other transcendental computations. Likewise, an application can perform packed 64-bit and 128-bit SIMD integer operations together without restrictions.
- Those SSE and SSE2 instructions that operate on MMX registers (such as the CVTTPS2PI, CVTTPS2PI, CVTPI2PS, CVTPD2PI, CVTTPD2PI, CVTPI2PD, MOVDQ2Q, MOVQ2DQ, PADDQ, and PSUBQ instructions) can also be executed in the same instruction stream as 64-bit SIMD integer or x87 FPU instructions, however, here they are subject to the restrictions on the simultaneous use of MMX technology and x87 FPU instructions, which include:
  - Transition from x87 FPU to MMX technology instructions or to SSE or SSE2 instructions that operate on MMX registers should be preceded by saving the state of the x87 FPU.
  - Transition from MMX technology instructions or from SSE or SSE2 instructions that operate on MMX registers to x87 FPU instructions should be preceded by execution of the EMMS instruction.

### 11.6.8 Compatibility of SIMD and x87 FPU Floating-Point Data Types

SSE and SSE2 extensions operate on the same single-precision and double-precision floating-point data types that the x87 FPU operates on. However, when operating on these data types, the SSE and SSE2 extensions operate on them in their native format (single-precision or double-precision), in contrast to the x87 FPU which extends them to double extended-precision floating-point format to perform computations and then rounds the result back to a single-precision or double-precision format before writing results to memory. Because the x87 FPU operates on a higher precision format and then rounds the result to a lower precision format, it may return a slightly different result when performing the same operation on the same single-precision or double-precision floating-point values than is returned by the SSE and SSE2 extensions. The difference occurs only in the least-significant bits of the significand.

### 11.6.9 Mixing Packed and Scalar Floating-Point and 128-Bit SIMD Integer Instructions and Data

SSE and SSE2 extensions define typed operations on packed and scalar floating-point data types and on 128-bit SIMD integer data types, but IA-32 processors do not enforce this typing at the architectural level. They only enforce it at the microarchitectural level. Therefore, when a Pentium 4 or Intel Xeon processor loads a packed or scalar floating-point operand or a 128-bit packed integer operand from memory into an XMM register, it does not check that the actual data being loaded matches the data type specified in the instruction. Likewise, when the processor performs an arithmetic operation on the data in an XMM register, it does not check that the data being operated on matches the data type specified in the instruction.

As a general rule, because data typing of SIMD floating-point and integer data types is not enforced at the architectural level, it is the responsibility of the programmer, assembler, or compiler to insure that code enforces data typing. Failure to enforce correct data typing can lead to computations that return unexpected results.

For example, in the following code sample, two packed single-precision floating-point operands are moved from memory into XMM registers (using MOVAPS instructions); then a double-precision packed add operation (using the ADDPD instruction) is performed on the operands:

```
movaps      xmm0, [eax] ; EAX register contains pointer to packed
                ; single-precision floating-point operand
movaps      xmm1, [ebx]
addpd       xmm0, xmm1
```

Pentium 4 and Intel Xeon processors execute these instructions without generating an invalid-operand exception (#UD) and will produce the expected results in register XMM0 (that is, the high and low 64-bits of each register will be treated as a double-precision floating-point value and the processor will operate on them accordingly). Because the data types operated on and the data type expected by the ADDPD instruction were inconsistent, the instruction

may result in a SIMD floating-point exception (such as numeric overflow [#O] or invalid operation [#I]) being generated, but the actual source of the problem (inconsistent data types) is not detected.

The ability to operate on an operand that contains a data type that is inconsistent with the typing of the instruction being executed, permits some valid operations to be performed. For example, the following instructions load a packed double-precision floating-point operand from memory to register XMM0, and a mask to register XMM1; then they use XORPD to toggle the sign bits of the two packed values in register XMM0.

```
movapd    xmm0, [eax] ; EAX register contains pointer to packed
           ; double-precision floating-point operand
movaps    xmm1, [ebx] ; EBX register contains pointer to packed
           ; double-precision floating-point mask
xorpd     xmm0, xmm1 ; XOR operation toggles sign bits using
           ; the mask in xmm1
```

In this example: XORPS or PXOR can be used in place of XORPD and yield the same correct result. However, because of the type mismatch between the operand data type and the instruction data type, a latency penalty will be incurred due to implementations of the instructions at the microarchitecture level.

Latency penalties can also be incurred by using move instructions of the wrong type. For example, MOVAPS and MOVAPD can both be used to move a packed single-precision operand from memory to an XMM register. However, if MOVAPD is used, a latency penalty will be incurred when a correctly typed instruction attempts to use the data in the register.

Note that these latency penalties are not incurred when moving data from XMM registers to memory.

## 11.6.10 Interfacing with SSE/SSE2 Procedures and Functions

SSE and SSE2 extensions allow direct access to XMM registers. This means that all existing interface conventions between procedures and functions that apply to the use of the general-purpose registers (EAX, EBX, etc.) also apply to XMM register usage.

### 11.6.10.1 Passing Parameters in XMM Registers

The state of XMM registers is preserved across procedure (or function) boundaries. Parameters can be passed from one procedure to another using XMM registers.

### 11.6.10.2 Saving XMM Register State on a Procedure or Function Call

The state of XMM registers can be saved in two ways: using an FXSAVE instruction or a move instruction. FXSAVE saves the state of all XMM registers (along with the state of MXCSR and the x87 FPU registers). This instruction is typically used for major changes in the context of the execution environment, such as a task switch. FXRSTOR restores the XMM, MXCSR, and x87 FPU registers stored with FXSAVE.

In cases where only XMM registers must be saved, or where selected XMM registers need to be saved, move instructions (MOVAPS, MOVUPS, MOVSS, MOVAPD, MOVUPD, MOVSD, MOVDQA, and MOVDQU) can be used. These instructions can also be used to restore the contents of XMM registers. To avoid performance degradation when saving XMM registers to memory or when loading XMM registers from memory, be sure to use the appropriately typed move instructions.

The move instructions can also be used to save the contents of XMM registers on the stack. Here, the stack pointer (in the ESP register) can be used as the memory address to the next available byte in the stack. Note that the stack pointer is not automatically incremented when using a move instruction (as it is with PUSH).

A move-instruction procedure that saves the contents of an XMM register to the stack is responsible for decrementing the value in the ESP register by 16. Likewise, a move-instruction procedure that loads an XMM register from the stack needs also to increment the ESP register by 16. To avoid performance degradation when moving the contents of XMM registers, use the appropriately typed move instructions.

Use the LDMXCSR and STMXCSR instructions to save and restore, respectively, the contents of the MXCSR register on a procedure call and return.

### 11.6.10.3 Caller-Save Recommendation for Procedure and Function Calls

When making procedure (or function) calls from SSE or SSE2 code, a caller-save convention is recommended for saving the state of the calling procedure. Using this convention, any register whose content must survive intact across a procedure call must be stored in memory by the calling procedure prior to executing the call.

The primary reason for using the caller-save convention is to prevent performance degradation. XMM registers can contain packed or scalar double-precision floating-point, packed single-precision floating-point, and 128-bit packed integer data types. The called procedure has no way of knowing the data types in XMM registers following a call; so it is unlikely to use the correctly typed move instruction to store the contents of XMM registers in memory or to restore the contents of XMM registers from memory.

As described in Section 11.6.9, “Mixing Packed and Scalar Floating-Point and 128-Bit SIMD Integer Instructions and Data,” executing a move instruction that does not match the type for the data being moved to/from XMM registers will be carried out correctly, but can lead to a greater instruction latency.

### 11.6.11 Updating Existing MMX Technology Routines Using 128-Bit SIMD Integer Instructions

SSE2 extensions extend all 64-bit MMX SIMD integer instructions to operate on 128-bit SIMD integers using XMM registers. The extended 128-bit SIMD integer instructions operate like the 64-bit SIMD integer instructions; this simplifies the porting of MMX technology applications. However, there are considerations:

- To take advantage of wider 128-bit SIMD integer instructions, MMX technology code must be recompiled to reference the XMM registers instead of MMX registers.
- Computation instructions that reference memory operands that are not aligned on 16-byte boundaries should be replaced with an unaligned 128-bit load (MOVUDQ instruction) followed by a version of the same computation operation that uses register instead of memory operands. Use of 128-bit packed integer computation instructions with memory operands that are not 16-byte aligned results in a general protection exception (#GP).
- Extension of the PSHUFW instruction (shuffle word across 64-bit integer operand) across a full 128-bit operand is emulated by a combination of the following instructions: PSHUFW, PSHUFLW, and PSHUFD.
- Use of the 64-bit shift by bit instructions (PSRLQ, PSSLQ) can be extended to 128 bits in either of two ways:
  - Use of PSRLQ and PSSLQ, along with masking logic operations.
  - Rewriting the code sequence to use PSRLDQ and PSLLDQ (shift double quadword operand by bytes)
- Loop counters need to be updated, since each 128-bit SIMD integer instruction operates on twice the amount of data as its 64-bit SIMD integer counterpart.

### 11.6.12 Branching on Arithmetic Operations

There are no condition codes in SSE or SSE2 states. A packed-data comparison instruction generates a mask which can then be transferred to an integer register. The following code sequence provides an example of how to perform a conditional branch, based on the result of an SSE2 arithmetic operation.

```

cmppd    XMM0, XMM1    ; generates a mask in XMM0
movmskpd EAX, XMM0    ; moves a 2 bit mask to eax
test     EAX, 0        ; compare with desired result
jne      BRANCH TARGET

```

The COMISD and UCOMISD instructions update the EFLAGS as the result of a scalar comparison. A conditional branch can then be scheduled immediately following COMISD/UCOMISD.



### 11.6.13 Cacheability Hint Instructions

SSE and SSE2 cacheability control instructions enable the programmer to control prefetching, caching, loading and storing of data. When correctly used, these instructions improve application performance.

To make efficient use of the processor's super-scalar microarchitecture, a program needs to provide a steady stream of data to the executing program to avoid stalling the processor. `PREFETCHh` instructions minimize the latency of data accesses in performance-critical sections of application code by allowing data to be fetched into the processor cache hierarchy in advance of actual usage.

`PREFETCHh` instructions do not change the user-visible semantics of a program, although they may affect performance. The operation of these instructions is implementation-dependent. Programmers may need to tune code for each IA-32 processor implementation. Excessive usage of `PREFETCHh` instructions may waste memory bandwidth and reduce performance. For more detailed information on the use of prefetch hints, refer to Chapter 7, "Optimizing Cache Usage," in the *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

The non-temporal store instructions (`MOVNTI`, `MOVNTPD`, `MOVNTPS`, `MOVNTDQ`, `MOVNTQ`, `MASKMOVQ`, and `MASKMOVDQU`) minimize cache pollution when writing non-temporal data to memory (see Section 10.4.6.1, "Cacheability Control Instructions" and Section 10.4.6.2, "Caching of Temporal vs. Non-Temporal Data"). They prevent non-temporal data from being written into processor caches on a store operation.

Besides reducing cache pollution, the use of weakly-ordered memory types can be important under certain data sharing relationships, such as a producer-consumer relationship. The use of weakly ordered memory can make the assembling of data more efficient; but care must be taken to ensure that the consumer obtains the data that the producer intended. Some common usage models that may be affected in this way by weakly-ordered stores are:

- Library functions that use weakly ordered memory to write results
- Compiler-generated code that writes weakly-ordered results
- Hand-crafted code

The degree to which a consumer of data knows that the data is weakly ordered can vary for these cases. As a result, the `SFENCE` or `MFENCE` instruction should be used to ensure ordering between routines that produce weakly-ordered data and routines that consume the data. `SFENCE` and `MFENCE` provide a performance-efficient way to ensure ordering by guaranteeing that every store instruction that precedes `SFENCE`/`MFENCE` in program order is globally visible before a store instruction that follows the fence.

### 11.6.14 Effect of Instruction Prefixes on the SSE/SSE2 Instructions

Table 11-3 describes the effects of instruction prefixes on SSE and SSE2 instructions. (Table 11-3 also applies to SIMD integer and SIMD floating-point instructions in SSE3.) Unpredictable behavior can range from prefixes being treated as a reserved operation on one generation of IA-32 processors to generating an invalid opcode exception on another generation of processors.

See also "Instruction Prefixes" in Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*, for complete description of instruction prefixes.

#### NOTE

Some SSE/SSE2/SSE3 instructions have two-byte opcodes that are either 2 bytes or 3 bytes in length. Two-byte opcodes that are 3 bytes in length consist of: a mandatory prefix (F2H, F3H, or 66H), 0FH, and an opcode byte. See Table 11-3.

**Table 11-3. Effect of Prefixes on SSE, SSE2, and SSE3 Instructions**

Prefix Type	Effect on SSE, SSE2 and SSE3 Instructions
Address Size Prefix (67H)	Affects instructions with a memory operand.
	Reserved for instructions without a memory operand and may result in unpredictable behavior.
Operand Size (66H)	Reserved and may result in unpredictable behavior.
Segment Override (2EH,36H,3EH,26H,64H,65H)	Affects instructions with a memory operand.
	Reserved for instructions without a memory operand and may result in unpredictable behavior.
Repeat Prefixes (F2H and F3H)	Reserved and may result in unpredictable behavior.
Lock Prefix (F0H)	Reserved; generates invalid opcode exception (#UD).
Branch Hint Prefixes(E2H and E3H)	Reserved and may result in unpredictable behavior.



### 3. Updates to Chapter 14, Volume 1

Change bars show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

-----  
Change to this chapter: Minor update to Section 14.5.2 "Fused-Multiply-ADD (FMA) Numeric Behavior".

Intel® Advanced Vector Extensions (Intel® AVX) introduces 256-bit vector processing capability. The Intel AVX instruction set extends 128-bit SIMD instruction sets by employing a new instruction encoding scheme via a vector extension prefix (VEX). Intel AVX also offers several enhanced features beyond those available in prior generations of 128-bit SIMD extensions.

FMA (Fused Multiply Add) extensions enhances Intel AVX further in floating-point numeric computations. FMA provides high-throughput, arithmetic operations cover fused multiply-add, fused multiply-subtract, fused multiply add/subtract interleave, signed-reversed multiply on fused multiply-add and multiply-subtract.

Intel AVX2 provides 256-bit integer SIMD extensions that accelerate computation across integer and floating-point domains using 256-bit vector registers.

This chapter summarizes the key features of Intel AVX, FMA and AVX2.

## 14.1 INTEL AVX OVERVIEW

Intel AVX introduces the following architectural enhancements:

- Support for 256-bit wide vectors with the YMM vector register set.
- 256-bit floating-point instruction set enhancement with up to 2X performance gain relative to 128-bit Streaming SIMD extensions.
- Enhancement of legacy 128-bit SIMD instruction extensions to support three-operand syntax and to simplify compiler vectorization of high-level language expressions.
- VEX prefix-encoded instruction syntax support for generalized three-operand syntax to improve instruction programming flexibility and efficient encoding of new instruction extensions.
- Most VEX-encoded 128-bit and 256-bit AVX instructions (with both load and computational operation semantics) are not restricted to 16-byte or 32-byte memory alignment.
- Support flexible deployment of 256-bit AVX code, 128-bit AVX code, legacy 128-bit code and scalar code.

With the exception of SIMD instructions operating on MMX registers, almost all legacy 128-bit SIMD instructions have AVX equivalents that support three operand syntax. 256-bit AVX instructions employ three-operand syntax and some with 4-operand syntax.

### 14.1.1 256-Bit Wide SIMD Register Support

Intel AVX introduces support for 256-bit wide SIMD registers (YMM0-YMM7 in operating modes that are 32-bit or less, YMM0-YMM15 in 64-bit mode). The lower 128-bits of the YMM registers are aliased to the respective 128-bit XMM registers.

Legacy SSE instructions (i.e. SIMD instructions operating on XMM state but not using the VEX prefix, also referred to non-VEX encoded SIMD instructions) will not access the upper bits beyond bit 128 of the YMM registers. AVX instructions with a VEX prefix and vector length of 128-bits zeroes the upper bits (above bit 128) of the YMM register.

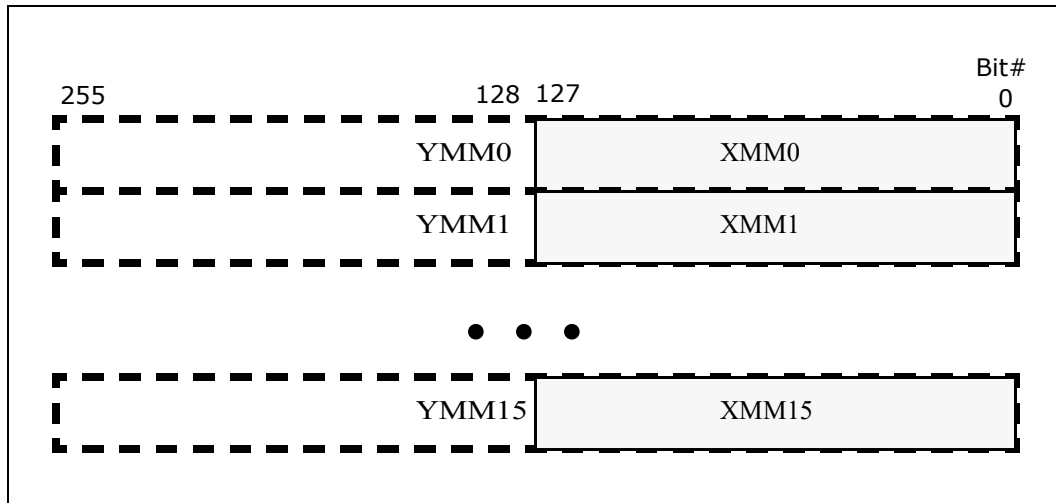


Figure 14-1. 256-Bit Wide SIMD Register

### 14.1.2 Instruction Syntax Enhancements

Intel AVX employs an instruction encoding scheme using a new prefix (known as “VEX” prefix). Instruction encoding using the VEX prefix can directly encode a register operand within the VEX prefix. This support two new instruction syntax in Intel 64 architecture:

- A non-destructive operand (in a three-operand instruction syntax): The non-destructive source reduces the number of registers, register-register copies and explicit load operations required in typical SSE loops, reduces code size, and improves micro-fusion opportunities.
- A third source operand (in a four-operand instruction syntax) via the upper 4 bits in an 8-bit immediate field. Support for the third source operand is defined for selected instructions (e.g. VBLENDVPD, VBLENDVPS, PBLENDVB).

Two-operand instruction syntax previously expressed in legacy SSE instruction as

```
ADDPS xmm1, xmm2/m128
```

128-bit AVX equivalent can be expressed in three-operand syntax as

```
VADDPS xmm1, xmm2, xmm3/m128
```

In four-operand syntax, the extra register operand is encoded in the immediate byte.

Note SIMD instructions supporting three-operand syntax but processing only 128-bits of data are considered part of the 256-bit SIMD instruction set extensions of AVX, because bits 255:128 of the destination register are zeroed by the processor.

### 14.1.3 VEX Prefix Instruction Encoding Support

Intel AVX introduces a new prefix, referred to as VEX, in the Intel 64 and IA-32 instruction encoding format. Instruction encoding using the VEX prefix provides the following capabilities:

- Direct encoding of a register operand within VEX. This provides instruction syntax support for non-destructive source operand.
- Efficient encoding of instruction syntax operating on 128-bit and 256-bit register sets.

- Compaction of REX prefix functionality: The equivalent functionality of the REX prefix is encoded within VEX.
- Compaction of SIMD prefix functionality and escape byte encoding: The functionality of SIMD prefix (66H, F2H, F3H) on opcode is equivalent to an opcode extension field to introduce new processing primitives. This functionality is replaced by a more compact representation of opcode extension within the VEX prefix. Similarly, the functionality of the escape opcode byte (0FH) and two-byte escape (0F38H, 0F3AH) are also compacted within the VEX prefix encoding.
- Most VEX-encoded SIMD numeric and data processing instruction semantics with memory operand have relaxed memory alignment requirements than instructions encoded using SIMD prefixes (see Section 14.9).

VEX prefix encoding applies to SIMD instructions operating on YMM registers, XMM registers, and in some cases with a general-purpose register as one of the operand. VEX prefix is not supported for instructions operating on MMX or x87 registers. Details of VEX prefix and instruction encoding are discussed in Chapter 2, "Instruction Format," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

## 14.2 FUNCTIONAL OVERVIEW

Intel AVX provide comprehensive functional improvements over previous generations of SIMD instruction extensions. The functional improvements include:

- 256-bit floating-point arithmetic primitives: AVX enhances existing 128-bit floating-point arithmetic instructions with 256-bit capabilities for floating-point processing. Table 14-1 lists SIMD instructions promoted to AVX.
- Enhancements for flexible SIMD data movements: AVX provides a number of new data movement primitives to enable efficient SIMD programming in relation to loading non-unit-strided data into SIMD registers, intra-register SIMD data manipulation, conditional expression and branch handling, etc. Enhancements for SIMD data movement primitives cover 256-bit and 128-bit vector floating-point data, and across 128-bit integer SIMD data processing using VEX-encoded instructions.

**Table 14-1. Promoted SSE/SSE2/SSE3/SSSE3/SSE4 Instructions**

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction	If No, Reason?	
yes	yes	YY OF 1X	MOVUPS	scalar	
no	yes		MOVSS		
yes	yes		MOVUPD		
no	yes		MOVSD		scalar
no	yes		MOVLPS		Note 1
no	yes		MOVLPD		Note 1
no	yes		MOVLHPS		Redundant with VPERMILPS
yes	yes		MOVDDUP		
yes	yes		MOVSLDUP		
yes	yes		UNPCKLPS		
yes	yes		UNPCKLPD		
yes	yes		UNPCKHPS		
yes	yes		UNPCKHPD		
no	yes		MOVHPS		Note 1
no	yes		MOVHPD		Note 1
no	yes		MOVHLPS		Redundant with VPERMILPS
yes	yes		MOVAPS		
yes	yes		MOVSHDUP		
yes	yes		MOVAPD		
no	no		CVTPI2PS		MMX

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction	If No, Reason?
no	yes	YY OF 5X	CVTSI2SS	scalar
no	no		CVTPI2PD	MMX
no	yes		CVTSI2SD	scalar
no	yes		MOVNTPS	
no	yes		MOVNTPD	
no	no		CVTTPS2PI	MMX
no	yes		CVTSS2SI	scalar
no	no		CVTTPD2PI	MMX
no	yes		CVTSD2SI	scalar
no	no		CVTPS2PI	MMX
no	yes		CVTSS2SI	scalar
no	no		CVTPD2PI	MMX
no	yes		CVTSD2SI	scalar
no	yes		UCOMISS	scalar
no	yes		UCOMISD	scalar
no	yes		COMISS	scalar
no	yes		COMISD	scalar
yes	yes		MOVMSKPS	
yes	yes		MOVMSKPD	
yes	yes		SQRTPS	
no	yes		SQRTSS	scalar
yes	yes		SQRTPD	
no	yes		SQRTSD	scalar
yes	yes		RSQRTPS	
no	yes		RSQRTSS	scalar
yes	yes		RCPPS	
no	yes		RCPSS	scalar
yes	yes		ANDPS	
yes	yes		ANDPD	
yes	yes		ANDNPS	
yes	yes		ANDNPD	
yes	yes		ORPS	
yes	yes		ORPD	
yes	yes		XORPS	
yes	yes		XORPD	
yes	yes		ADDPS	
no	yes		ADDSS	scalar
yes	yes		ADDPD	
no	yes		ADDSD	scalar
yes	yes		MULPS	
no	yes	MULSS	scalar	
yes	yes	MULPD		
no	yes	MULSD	scalar	
yes	yes	CVTTPS2PD		

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction	If No, Reason?
no	yes		CVTSS2SD	scalar
yes	yes		CVTPD2PS	
no	yes		CVTSD2SS	scalar
yes	yes		CVTDQ2PS	
yes	yes		CVTPS2DQ	
yes	yes		CVTTPS2DQ	
yes	yes		SUBPS	
no	yes		SUBSS	scalar
yes	yes		SUBPD	
no	yes		SUBSD	scalar
yes	yes		MINPS	
no	yes		MINSS	scalar
yes	yes		MINPD	
no	yes		MINSB	scalar
yes	yes		DIVPS	
no	yes		DIVSS	scalar
yes	yes		DIVPD	
no	yes		DIVSD	scalar
yes	yes		MAXPS	
no	yes		MAXSS	scalar
yes	yes		MAXPD	
no	yes		MAXSD	scalar
no	yes	YY OF 6X	PUNPCKLBW	VI
no	yes		PUNPCKLWD	VI
no	yes		PUNPCKLDQ	VI
no	yes		PACKSSWB	VI
no	yes		PCMPGTB	VI
no	yes		PCMPGTW	VI
no	yes		PCMPGTD	VI
no	yes		PACKUSWB	VI
no	yes		PUNPCKHBW	VI
no	yes		PUNPCKHWD	VI
no	yes		PUNPCKHDQ	VI
no	yes		PACKSSDW	VI
no	yes		PUNPCKLQDQ	VI
no	yes		PUNPCKHQDQ	VI
no	yes		MOVB	scalar
no	yes		MOVQ	scalar
yes	yes		MOVBQ	
yes	yes		MOVBQ	
no	yes	YY OF 7X	PSHUFB	VI
no	yes		PSHUFBW	VI
no	yes		PSHUFLW	VI
no	yes		PCMPEQB	VI

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction	If No, Reason?
no	yes		PCMPEQW	VI
no	yes		PCMPEQD	VI
yes	yes		HADDPD	
yes	yes		HADDPS	
yes	yes		HSUBPD	
yes	yes		HSUBPS	
no	yes		MOVD	VI
no	yes		MOVQ	VI
yes	yes		MOVDQA	
yes	yes		MOVDQU	
no	yes	YY OF AX	LDMXCSR	
no	yes		STMXCSR	
yes	yes	YY OF CX	CMPPS	
no	yes		CMPSS	scalar
yes	yes		CMPPD	
no	yes		CMPSD	scalar
no	yes		PINSRW	VI
no	yes		PEXTRW	VI
yes	yes		SHUFPS	
yes	yes		SHUFPD	
yes	yes	YY OF DX	ADDSUBPD	
yes	yes		ADDSUBPS	
no	yes		PSRLW	VI
no	yes		PSRLD	VI
no	yes		PSRLQ	VI
no	yes		PADDQ	VI
no	yes		PMULLW	VI
no	no		MOVQ2DQ	MMX
no	no		MOVDQ2Q	MMX
no	yes		PMOVMSKB	VI
no	yes		PSUBUSB	VI
no	yes		PSUBUSW	VI
no	yes		PMINUB	VI
no	yes		PAND	VI
no	yes		PADDUSB	VI
no	yes		PADDUSW	VI
no	yes		PMAXUB	VI
no	yes		PANDN	VI
no	yes	YY OF EX	PAVGB	VI
no	yes		PSRAW	VI
no	yes		PSRAD	VI
no	yes		PAVGW	VI
no	yes		PMULHUW	VI
no	yes		PMULHW	VI

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction	If No, Reason?
yes	yes		CVTPD2DQ	
yes	yes		CVTTPD2DQ	
yes	yes		CVTDQ2PD	
no	yes		MOVNTDQ	VI
no	yes		PSUBSB	VI
no	yes		PSUBSW	VI
no	yes		PMINSW	VI
no	yes		POR	VI
no	yes		PADDSB	VI
no	yes		PADDSW	VI
no	yes		PMAXSW	VI
no	yes		PXOR	VI
yes	yes	YY OF FX	LDDQU	VI
no	yes		PSLLW	VI
no	yes		PSLLD	VI
no	yes		PSLLQ	VI
no	yes		PMULUDQ	VI
no	yes		PMADDWD	VI
no	yes		PSADBW	VI
no	yes		MASKMOVDQU	
no	yes		PSUBB	VI
no	yes		PSUBW	VI
no	yes		PSUBD	VI
no	yes		PSUBQ	VI
no	yes		PADDB	VI
no	yes		PADDW	VI
no	yes		PADDQ	VI
no	yes	SSSE3	PHADDW	VI
no	yes		PHADDSW	VI
no	yes		PHADDQ	VI
no	yes		PHSUBW	VI
no	yes		PHSUBSW	VI
no	yes		PHSUBD	VI
no	yes		PMADDUBSW	VI
no	yes		PALIGNR	VI
no	yes		PSHUFB	VI
no	yes		PMULHRSW	VI
no	yes		PSIGNB	VI
no	yes		PSIGNW	VI
no	yes		PSIGND	VI
no	yes		PABSB	VI
no	yes		PABSW	VI
no	yes		PABSD	VI
yes	yes	SSE4.1	BLENDPS	



VEX.256 Encoding	VEX.128 Encoding	Group	Instruction	If No, Reason?
yes	yes		BLENDDP	
yes	yes		BLENDVPS	Note 2
yes	yes		BLENDVPD	Note 2
no	yes		DPPD	
yes	yes		DPPS	
no	yes		EXTRACTPS	Note 3
no	yes		INSERTPS	Note 3
no	yes		MOVNTDQA	
no	yes		MPSADBW	VI
no	yes		PACKUSDW	VI
no	yes		PBLENDVB	VI
no	yes		PBLENDW	VI
no	yes		PCMPEQQ	VI
no	yes		PEXTRD	VI
no	yes		PEXTRQ	VI
no	yes		PEXTRB	VI
no	yes		PEXTRW	VI
no	yes		PHMINPOSUW	VI
no	yes		PINSRB	VI
no	yes		PINSRD	VI
no	yes		PINSRQ	VI
no	yes		PMAXSB	VI
no	yes		PMAXSD	VI
no	yes		PMAXUD	VI
no	yes		PMAXUW	VI
no	yes		PMINSB	VI
no	yes		PMINSD	VI
no	yes		PMINUD	VI
no	yes		PMINUW	VI
no	yes		PMOVSXxx	VI
no	yes		PMOVZXxx	VI
no	yes		PMULDQ	VI
no	yes		PMULLD	VI
yes	yes		PTEST	
yes	yes		ROUNDPD	
yes	yes		ROUNDPS	
no	yes		ROUNDSD	scalar
no	yes		ROUNDSS	scalar
no	yes	SSE4.2	PCMPGTQ	VI
no	no	SSE4.2	CRC32c	integer
no	yes		PCMPESTRI	VI
no	yes		PCMPESTRM	VI

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction	If No, Reason?
no	yes		PCMPISTRI	VI
no	yes		PCMPISTRM	VI
no	no	SSE4.2	POPCNT	integer

### 14.2.1 256-bit Floating-Point Arithmetic Processing Enhancements

Intel AVX provides 35 256-bit floating-point arithmetic instructions, see Table 14-2. The arithmetic operations cover add, subtract, multiply, divide, square-root, compare, max, min, round, etc., on single-precision and double-precision floating-point data.

The enhancement in AVX on floating-point compare operation provides 32 conditional predicates to improve programming flexibility in evaluating conditional expressions.

**Table 14-2. Promoted 256-Bit and 128-bit Arithmetic AVX Instructions**

VEX.256 Encoding	VEX.128 Encoding	Legacy Instruction Mnemonic
yes	yes	SQRTPS, SQRTPD, RSQRTPS, RCPPS
yes	yes	ADDPS, ADDPD, SUBPS, SUBPD
yes	yes	MULPS, MULPD, DIVPS, DIVPD
yes	yes	CVTQ2PS, CVTQ2PD
yes	yes	CVTDQ2PS, CVTDP2DQ
yes	yes	CVTTPS2DQ, CVTTPD2DQ
yes	yes	CVTPD2DQ, CVTDQ2PD
yes	yes	MINPS, MINPD, MAXPS, MAXPD
yes	yes	HADDPD, HADDPS, HSUBPD, HSUBPS
yes	yes	CMPPS, CMPPD
yes	yes	ADDSUBPD, ADDSUBPS, DPPS
yes	yes	ROUNDPD, ROUNDPS

### 14.2.2 256-bit Non-Arithmetic Instruction Enhancements

Intel AVX provides new primitives for handling data movement within 256-bit floating-point vectors and promotes many 128-bit floating data processing instructions to handle 256-bit floating-point vectors.

AVX includes 39 256-bit data movement and processing instructions that are promoted from previous generations of SIMD instruction extensions, ranging from logical, blend, convert, test, unpacking, shuffling, load and stores (see Table 14-3).

**Table 14-3. Promoted 256-bit and 128-bit Data Movement AVX Instructions**

VEX.256 Encoding	VEX.128 Encoding	Legacy Instruction Mnemonic
yes	yes	MOVAPS, MOVAPD, MOVDQA
yes	yes	MOVUPS, MOVUPD, MOVDQU
yes	yes	MOVMSKPS, MOVMSKPD
yes	yes	LDDQU, MOVNTPS, MOVNTPD, MOVNTDQ, MOVNTDQA
yes	yes	MOVSHDUP, MOVSLDUP, MOVDDUP

**Table 14-3. Promoted 256-bit and 128-bit Data Movement AVX Instructions**

VEX.256 Encoding	VEX.128 Encoding	Legacy Instruction Mnemonic
yes	yes	UNPCKHPD, UNPCKHPS, UNPCKLPD
yes	yes	BLENDPS, BLENDPD
yes	yes	SHUFPS, SHUFPS, UNPCKLPS
yes	yes	BLENDVPS, BLENDVPD
yes	yes	PTEST, MOVMSKPD, MOVMSKPS
yes	yes	XORPS, XORPD, ORPS, ORPD
yes	yes	ANDNPD, ANDNPS, ANDPD, ANDPS

AVX introduces 18 new data processing instructions that operate on 256-bit vectors, Table 14-4. These new primitives cover the following operations:

- Non-unit-strided fetching of SIMD data. AVX provides several flexible SIMD floating-point data fetching primitives:
  - broadcast of single or multiple data elements into a 256-bit destination,
  - masked move primitives to load or store SIMD data elements conditionally,
- Intra-register manipulation of SIMD data elements. AVX provides several flexible SIMD floating-point data manipulation primitives:
  - insert/extract multiple SIMD floating-point data elements to/from 256-bit SIMD registers
  - permute primitives to facilitate efficient manipulation of floating-point data elements in 256-bit SIMD registers
- Branch handling. AVX provides several primitives to enable handling of branches in SIMD programming:
  - new variable blend instructions supports four-operand syntax with non-destructive source syntax. This is more flexible than the equivalent SSE4 instruction syntax which uses the XMM0 register as the implied mask for blend selection.
  - Packed TEST instructions for floating-point data.

**Table 14-4. 256-bit AVX Instruction Enhancement**

Instruction	Description
VBROADCASTF128 ymm1, m128	Broadcast 128-bit floating-point values in mem to low and high 128-bits in ymm1.
VBROADCASTSD ymm1, m64	Broadcast double-precision floating-point element in mem to four locations in ymm1.
VBROADCASTSS ymm1, m32	Broadcast single-precision floating-point element in mem to eight locations in ymm1.
VEXTRACTF128 xmm1/m128, ymm2, imm8	Extracts 128-bits of packed floating-point values from ymm2 and store results in xmm1/mem.
VINSERTF128 ymm1, ymm2, xmm3/m128, imm8	Insert 128-bits of packed floating-point values from xmm3/mem and the remaining values from ymm2 into ymm1
VMASKMOVPS ymm1, ymm2, m256	Load packed single-precision values from mem using mask in ymm2 and store in ymm1
VMASKMOVDPD ymm1, ymm2, m256	Load packed double-precision values from mem using mask in ymm2 and store in ymm1
VMASKMOVPS m256, ymm1, ymm2	Store packed single-precision values from ymm2 mask in ymm1
VMASKMOVDPD m256, ymm1, ymm2	Store packed double-precision values from ymm2 using mask in ymm1
VPERMILPD ymm1, ymm2, ymm3/m256	Permute Double-Precision Floating-Point values in ymm2 using controls from xmm3/mem and store result in ymm1

**Table 14-4. 256-bit AVX Instruction Enhancement**

Instruction	Description
VPERMILPD ymm1, ymm2/m256 imm8	Permute Double-Precision Floating-Point values in ymm2/mem using controls from imm8 and store result in ymm1
VPERMILPS ymm1, ymm2, ymm/m256	Permute Single-Precision Floating-Point values in ymm2 using controls from ymm3/mem and store result in ymm1
VPERMILPS ymm1, ymm2/m256, imm8	Permute Single-Precision Floating-Point values in ymm2/mem using controls from imm8 and store result in ymm1
VPERM2F128 ymm1, ymm2, ymm3/m256, imm8	Permute 128-bit floating-point fields in ymm2 and ymm3/mem using controls from imm8 and store result in ymm1
VTESTPS ymm1, ymm2/m256	Set ZF if ymm2/mem AND ymm1 result is all 0s in packed single-precision sign bits. Set CF if ymm2/mem AND NOT ymm1 result is all 0s in packed single-precision sign bits.
VTESTPD ymm1, ymm2/m256	Set ZF if ymm2/mem AND ymm1 result is all 0s in packed double-precision sign bits. Set CF if ymm2/mem AND NOT ymm1 result is all 0s in packed double-precision sign bits.
VZEROALL	Zero all YMM registers
VZERoupper	Zero upper 128 bits of all YMM registers

### 14.2.3 Arithmetic Primitives for 128-bit Vector and Scalar processing

Intel AVX provides a full complement of 128-bit numeric processing instructions that employ VEX-prefix encoding. These VEX-encoded instructions generally provide the same functionality over instructions operating on XMM register that are encoded using SIMD prefixes. The 128-bit numeric processing instructions in AVX cover floating-point and integer data processing; across 128-bit vector and scalar processing. Table 14-5 lists the state of promotion of legacy SIMD arithmetic ISA to VEX-128 encoding. Legacy SIMD floating-point arithmetic ISA promoted to VEX-256 encoding also support VEX-128 encoding (see Table 14-2).

The enhancement in AVX on 128-bit floating-point compare operation provides 32 conditional predicates to improve programming flexibility in evaluating conditional expressions. This contrasts with floating-point SIMD compare instructions in SSE and SSE2 supporting only 8 conditional predicates.

**Table 14-5. Promotion of Legacy SIMD ISA to 128-bit Arithmetic AVX instruction**

VEX.256 Encoding	VEX.128 Encoding	Instruction	Reason Not Promoted
no	no	CVTPI2PS, CVTPI2PD, CVTPD2PI	MMX
no	no	CVTTPS2PI, CVTTPD2PI, CVTPS2PI	MMX
no	yes	CVTSI2SS, CVTSI2SD, CVTSD2SI	scalar
no	yes	CVTSS2SI, CVTSS2SD, CVTSS2SI	scalar
no	yes	COMISD, RSQRTSS, RCPSS	scalar
no	yes	UCOMISS, UCOMISD, COMISS,	scalar
no	yes	ADDSS, ADDSD, SUBSS, SUBSD	scalar
no	yes	MULSS, MULSD, DIVSS, DIVSD	scalar
no	yes	SQRTSS, SQRTSD	scalar
no	yes	CVTSS2SD, CVTSD2SS	scalar
no	yes	MINSS, MINSD, MAXSS, MAXSD	scalar
no	yes	PAND, PANDN, POR, PXOR	VI
no	yes	PCMPGTB, PCMPGTW, PCMPGTD	VI

**Table 14-5. Promotion of Legacy SIMD ISA to 128-bit Arithmetic AVX instruction**

VEX.256 Encoding	VEX.128 Encoding	Instruction	Reason Not Promoted
no	yes	PMADDWD, PMADDUBSW	VI
no	yes	PAVGB, PAVGW, PMULUDQ	VI
no	yes	PCMPEQB, PCMPEQW, PCMPEQD	VI
no	yes	PMULLW, PMULHUW, PMULHW	VI
no	yes	PSUBSW, PADDsw, PSADBw	VI
no	yes	PADDUSB, PADDUSW, PADDsB	VI
no	yes	PSUBUSB, PSUBUSW, PSUBsB	VI
no	yes	PMINUB, PMINSW	VI
no	yes	PMAXUB, PMAxSW	VI
no	yes	PADDB, PADDW, PADDD, PADDQ	VI
no	yes	PSUBB, PSUBW, PSUBD, PSUBQ	VI
no	yes	PSLLW, PSLLD, PSLLQ, PSRAW	VI
no	yes	PSRLW, PSRLD, PSRLQ, PSRAD	VI
CPUID.SSSE3			
no	yes	PHSUBW, PHSUBD, PHSUBSW	VI
no	yes	PHADDW, PHADDD, PHADDSW	VI
no	yes	PMULHRSW	VI
no	yes	PSIGNB, PSIGNW, PSIGND	VI
no	yes	PABSB, PABSW, PABSD	VI
CPUID.SSE4_1			
no	yes	DPPD	
no	yes	PHMINPOSUW, MPSADBw	VI
no	yes	PMAxSB, PMAxSD, PMAxUD	VI
no	yes	PMINSB, PMINSD, PMINUD	VI
no	yes	PMAxUW, PMINUW	VI
no	yes	PMOVSXxx, PMOVZXxx	VI
no	yes	PMULDQ, PMULLD	VI
no	yes	ROUNDSD, ROUNDSS	scalar
CPUID.POPCNT			
no	yes	POPCNT	integer
CPUID.SSE4_2			
no	yes	PCMPGTQ	VI
no	no	CRC32	integer
no	yes	PCMPESTRI, PCMPESTRM	VI
no	yes	PCMPISTRi, PCMPISTRM	VI
CPUID.CLMUL			
no	yes	PCLMULQDQ	VI
CPUID.AESNI			

**Table 14-5. Promotion of Legacy SIMD ISA to 128-bit Arithmetic AVX instruction**

VEX.256 Encoding	VEX.128 Encoding	Instruction	Reason Not Promoted
no	yes	AESDEC, AESDECLAST	VI
no	yes	AESENC, AESENCLAST	VI
no	yes	AESIMX, AESKEYGENASSIST	VI

Description of Column “Reason not promoted?”

**MMX:** Instructions referencing MMX registers do not support VEX

**Scalar:** Scalar instructions are not promoted to 256-bit

**integer:** integer instructions are not promoted.

**VI:** “Vector Integer” instructions are not promoted to 256-bit

### 14.2.4 Non-Arithmetic Primitives for 128-bit Vector and Scalar Processing

Intel AVX provides a full complement of data processing instructions that employ VEX-prefix encoding. These VEX-encoded instructions generally provide the same functionality over instructions operating on XMM register that are encoded using SIMD prefixes.

A subset of new functionalities listed in Table 14-4 is also extended via VEX.128 encoding. These enhancements in AVX on 128-bit data processing primitives include 11 new instructions (see Table 14-6) with the following capabilities:

- Non-unit-strided fetching of SIMD data. AVX provides several flexible SIMD floating-point data fetching primitives:
  - broadcast of single data element into a 128-bit destination,
  - masked move primitives to load or store SIMD data elements conditionally,
- Intra-register manipulation of SIMD data elements. AVX provides several flexible SIMD floating-point data manipulation primitives:
  - permute primitives to facilitate efficient manipulation of floating-point data elements in 128-bit SIMD registers
- Branch handling. AVX provides several primitives to enable handling of branches in SIMD programming:
  - new variable blend instructions supports four-operand syntax with non-destructive source syntax. Branching conditions dependent on floating-point data or integer data can benefit from Intel AVX. This is more flexible than non-VEX encoded instruction syntax that uses the XMM0 register as implied mask for blend selection. While variable blend with implied XMM0 syntax is supported in SSE4 using SIMD prefix encoding, VEX-encoded 128-bit variable blend instructions only support the more flexible four-operand syntax.
  - Packed TEST instructions for floating-point data.

**Table 14-6. 128-bit AVX Instruction Enhancement**

Instruction	Description
VBROADCASTSS xmm1, m32	Broadcast single-precision floating-point element in mem to four locations in xmm1.
VMASKMOVPS xmm1, xmm2, m128	Load packed single-precision values from mem using mask in xmm2 and store in xmm1
VMASKMOVPSD xmm1, xmm2, m128	Load packed double-precision values from mem using mask in xmm2 and store in xmm1
VMASKMOVPS m128, xmm1, xmm2	Store packed single-precision values from xmm2 using mask in xmm1
VMASKMOVPSD m128, xmm1, xmm2	Store packed double-precision values from xmm2 using mask in xmm1

**Table 14-6. 128-bit AVX Instruction Enhancement**

Instruction	Description
VPERMILPD xmm1, xmm2, xmm3/m128	Permute Double-Precision Floating-Point values in xmm2 using controls from xmm3/mem and store result in xmm1
VPERMILPD xmm1, xmm2/m128, imm8	Permute Double-Precision Floating-Point values in xmm2/mem using controls from imm8 and store result in xmm1
VPERMILPS xmm1, xmm2, xmm3/m128	Permute Single-Precision Floating-Point values in xmm2 using controls from xmm3/mem and store result in xmm1
VPERMILPS xmm1, xmm2/m128, imm8	Permute Single-Precision Floating-Point values in xmm2/mem using controls from imm8 and store result in xmm1
VTESTPS xmm1, xmm2/m128	Set ZF if xmm2/mem AND xmm1 result is all 0s in packed single-precision sign bits. Set CF if xmm2/mem AND NOT xmm1 result is all 0s in packed single-precision sign bits.
VTESTPD xmm1, xmm2/m128	Set ZF if xmm2/mem AND xmm1 result is all 0s in packed single precision sign bits. Set CF if xmm2/mem AND NOT xmm1 result is all 0s in packed double-precision sign bits.

The 128-bit data processing instructions in AVX cover floating-point and integer data movement primitives. Legacy SIMD non-arithmetic ISA promoted to VEX-256 encoding also support VEX-128 encoding (see Table 14-3). Table 14-7 lists the state of promotion of the remaining legacy SIMD non-arithmetic ISA to VEX-128 encoding.

**Table 14-7. Promotion of Legacy SIMD ISA to 128-bit Non-Arithmetic AVX instruction**

VEX.256 Encoding	VEX.128 Encoding	Instruction	Reason Not Promoted
no	no	MOVQ2DQ, MOVDQ2Q	MMX
no	yes	LDMXCSR, STMXCSR	
no	yes	MOVSS, MOVSD, CMPSS, CMPSD	scalar
no	yes	MOVHPS, MOVHPD	Note 1
no	yes	MOVLPS, MOVLPD	Note 1
no	yes	MOVLHPS, MOVHLPS	Redundant with VPERMILPS
no	yes	MOVQ, MOVD	scalar
no	yes	PACKUSWB, PACKSSDW, PACKSSWB	VI
no	yes	PUNPCKHBW, PUNPCKHWD	VI
no	yes	PUNPCKLBW, PUNPCKLWD	VI
no	yes	PUNPCKHDQ, PUNPCKLDQ	VI
no	yes	PUNPCKLQDQ, PUNPCKHQDQ	VI
no	yes	PSHUFBW, PSHUFLW, PSHUFD	VI
no	yes	PMOVMASKB, MASKMOVDQU	VI
no	yes	PAND, PANDN, POR, PXOR	VI
no	yes	PINSRW, PEXTRW,	VI
CPUID.SSSE3			
no	yes	PALIGNR, PSHUFB	VI
CPUID.SSE4_1			
no	yes	EXTRACTPS, INSERTPS	Note 3
no	yes	PACKUSDW, PCMPEQQ	VI

**Table 14-7. Promotion of Legacy SIMD ISA to 128-bit Non-Arithmetic AVX instruction**

VEX.256 Encoding	VEX.128 Encoding	Instruction	Reason Not Promoted
no	yes	PBLENDVB, PBLENDW	VI
no	yes	PEXTRW, PEXTRB, PEXTRD, PEXTRQ	VI
no	yes	PINSRB, PINSRD, PINSRQ	VI

Description of Column “Reason not promoted?”

**MMX:** Instructions referencing MMX registers do not support VEX

**Scalar:** Scalar instructions are not promoted to 256-bit

**VI:** “Vector Integer” instructions are not promoted to 256-bit

**Note 1:** MOVLDP/PS and MOVHPD/PS are not promoted to 256-bit. The equivalent functionality are provided by VINSERTF128 and VEXTRACTF128 instructions as the existing instructions have no natural 256b extension

**Note 3:** It is expected that using 128-bit INSERTPS followed by a VINSERTF128 would be better than promoting INSERTPS to 256-bit (for example).

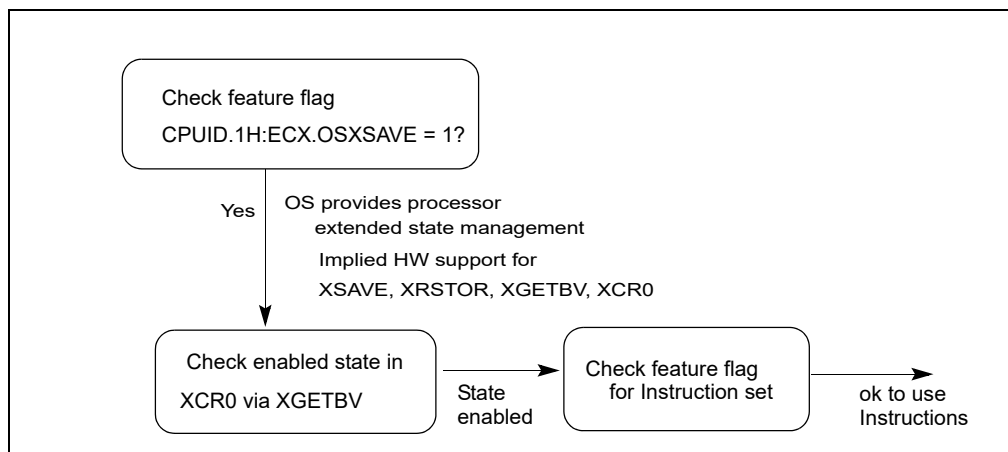
### 14.3 DETECTION OF AVX INSTRUCTIONS

Intel AVX instructions operate on the 256-bit YMM register state. Application detection of new instruction extensions operating on the YMM state follows the general procedural flow in Figure 14-2.

Prior to using AVX, the application must identify that the operating system supports the XGETBV instruction, the YMM register state, in addition to processor’s support for YMM state management using XSAVE/XRSTOR and AVX instructions. The following simplified sequence accomplishes both and is strongly recommended.

- 1) Detect CPUID.1:ECX.OSXSAVE[bit 27] = 1 (XGETBV enabled for application use<sup>1</sup>)
- 2) Issue XGETBV and verify that XCR0[2:1] = ‘11b’ (XMM state and YMM state are enabled by OS).
- 3) detect CPUID.1:ECX.AVX[bit 28] = 1 (AVX instructions supported).

(Step 3 can be done in any order relative to 1 and 2)



**Figure 14-2. General Procedural Flow of Application Detection of AVX**

1. If CPUID.01H:ECX.OSXSAVE reports 1, it also indirectly implies the processor supports XSAVE, XRSTOR, XGETBV, processor extended state bit vector XCR0. Thus an application may streamline the checking of CPUID feature flags for XSAVE and OSXSAVE. XSETBV is a privileged instruction.



The following pseudocode illustrates this recommended application AVX detection process:

**Example 14-1. Detection of AVX Instruction**

```

INT supports_AVX()
{
    mov     eax, 1
    cpuid
    and     ecx, 018000000H
    cmp     ecx, 018000000H; check both OSXSAVE and AVX feature flags
    jne     not_supported
    ; processor supports AVX instructions and XGETBV is enabled by OS
    mov     ecx, 0; specify 0 for XCRO register
    XGETBV     ; result in EDX:EAX
    and     eax, 06H
    cmp     eax, 06H; check OS has enabled both XMM and YMM state support
    jne     not_supported
    mov     eax, 1
    jmp     done
NOT_SUPPORTED:
    mov     eax, 0
done:
}

```

Note: It is unwise for an application to rely exclusively on CPUID.1:ECX.AVX[bit 28] or at all on CPUID.1:ECX.XSAVE[bit 26]: These indicate hardware support but not operating system support. If YMM state management is not enabled by an operating systems, AVX instructions will #UD regardless of CPUID.1:ECX.AVX[bit 28]. "CPUID.1:ECX.XSAVE[bit 26] = 1" does not guarantee the OS actually uses the XSAVE process for state management.

These steps above also apply to enhanced 128-bit SIMD floating-pointing instructions in AVX (using VEX prefix-encoding) that operate on the YMM states.

### 14.3.1 Detection of VEX-Encoded AES and VPCLMULQDQ

VAESDEC/VAESDECLAST/VAESENC/VAESENCLAST/VAESIMC/VAESKEYGENASSIST instructions operate on YMM states. The detection sequence must combine checking for CPUID.1:ECX.AES[bit 25] = 1 and the sequence for detection application support for AVX.

#### Example 14-2. Detection of VEX-Encoded AESNI Instructions

```

INT supports_VAESNI()
{
    mov     eax, 1
    cpuid
    and     ecx, 01A000000H
    cmp     ecx, 01A000000H; check OSXSAVE AVX and AESNI feature flags
    jne     not_supported
    ; processor supports AVX and VEX-encoded AESNI and XGETBV is enabled by OS
    mov     ecx, 0; specify 0 for XCRO register
    XGETBV     ; result in EDX:EAX
    and     eax, 06H
    cmp     eax, 06H; check OS has enabled both XMM and YMM state support
    jne     not_supported
    mov     eax, 1
    jmp     done
NOT_SUPPORTED:
    mov     eax, 0
done:

```

Similarly, the detection sequence for VPCLMULQDQ must combine checking for CPUID.1:ECX.PCLMULQDQ[bit 1] = 1 and the sequence for detection application support for AVX.

This is shown in the pseudocode:

#### Example 14-3. Detection of VEX-Encoded AESNI Instructions

```

INT supports_VPCLMULQDQ()
{
    mov     eax, 1
    cpuid
    and     ecx, 018000002H
    cmp     ecx, 018000002H; check OSXSAVE AVX and PCLMULQDQ feature flags
    jne     not_supported
    ; processor supports AVX and VEX-encoded PCLMULQDQ and XGETBV is enabled by OS
    mov     ecx, 0; specify 0 for XCRO register
    XGETBV     ; result in EDX:EAX
    and     eax, 06H
    cmp     eax, 06H; check OS has enabled both XMM and YMM state support
    jne     not_supported

    mov     eax, 1
    jmp     done
NOT_SUPPORTED:
    mov     eax, 0
done:

```

## 14.4 HALF-PRECISION FLOATING-POINT CONVERSION

VCVTPH2PS and VCVTPS2PH are two instructions supporting half-precision floating-point data type conversion to and from single-precision floating-point data types.

Half-precision floating-point values are not used by the processor directly for arithmetic operations. But the conversion operation are subject to SIMD floating-point exceptions.

Additionally, The conversion operations of VCVTPS2PH allow programmer to specify rounding control using control fields in an immediate byte. The effects of the immediate byte are listed in Table 14-8.

Rounding control can use Imm[2] to select an override RC field specified in Imm[1:0] or use MXCSR setting.

**Table 14-8. Immediate Byte Encoding for 16-bit Floating-Point Conversion Instructions**

Bits	Field Name/value	Description	Comment
Imm[1:0]	RC=00B	Round to nearest even	If Imm[2] = 0
	RC=01B	Round down	
	RC=10B	Round up	
	RC=11B	Truncate	
Imm[2]	MS1=0	Use imm[1:0] for rounding	Ignore MXCSR.RC
	MS1=1	Use MXCSR.RC for rounding	
Imm[7:3]	Ignored	Ignored by processor	

Specific SIMD floating-point exceptions that can occur in conversion operations are shown in Table 14-9 and Table 14-10.

**Table 14-9. Non-Numerical Behavior for VCVTPH2PS, VCVTPS2PH**

Source Operands	Masked Result	Unmasked Result
QNaN	QNaN <sup>1</sup>	QNaN <sup>1</sup> (not an exception)
SNaN	QNaN <sup>2</sup>	None

**NOTES:**

1. The half precision output QNaN1 is created from the single precision input QNaN as follows: the sign bit is preserved, the 8-bit exponent FFH is replaced by the 5-bit exponent 1FH, and the 24-bit significand is truncated to an 11-bit significand by removing its 14 least significant bits.
2. The half precision output QNaN1 is created from the single precision input SNaN as follows: the sign bit is preserved, the 8-bit exponent FFH is replaced by the 5-bit exponent 1FH, and the 24-bit significand is truncated to an 11-bit significand by removing its 14 least significant bits. The second most significant bit of the significand is changed from 0 to 1 to convert the signaling NaN into a quiet NaN.

**Table 14-10. Invalid Operation for VCVTPH2PS, VCVTPS2PH**

Instruction	Condition	Masked Result	Unmasked Result
VCVTPH2PS	SRC = NaN	See Table 14-9	#I=1
VCVTPS2PH	SRC = NaN	See Table 14-9	#I=1

VCVTPS2PH can cause denormal exceptions if the value of the source operand is denormal relative to the numerical range represented by the source format (see Table 14-11).

**Table 14-11. Denormal Condition Summary**

Instruction	Condition	Masked Result	Unmasked Result
VCVTPH2PS	SRC is denormal relative to input format	res = Result rounded to the destination precision and using the bounded exponent, but only if no unmasked post-computation exception occurs. #DE unchanged	Same as masked result.
VCVTPS2PH	SRC is denormal relative to input format	res = Result rounded to the destination precision and using the bounded exponent, but only if no unmasked post-computation exception occurs. #DE=1	#DE=1

VCVTPS2PH can cause an underflow exception if the result of the conversion is less than the underflow threshold for half-precision floating-point data type, i.e.  $|x| < 1.0 * 2^{-14}$ .

**Table 14-12. Underflow Condition for VCVTPS2PH**

Instruction	Condition	Masked Result <sup>1</sup>	Unmasked Result
VCVTPS2PH	Result < smallest destination precision final normal value <sup>2</sup>	Result = +0 or -0, denormal, normal. #UE = 1. #PE = 1 if the result is inexact.	#UE=1, #PE = 1 if the result is inexact.

**NOTES:**

1. Masked and unmasked results are shown in Table 14-11.
2. MXCSR.FTZ is ignored, the processor behaves as if MXCSR.FTZ = 0.

VCVTPS2PH can cause an overflow exception if the result of the conversion is greater than the maximum representable value for half-precision floating-point data type, i.e.  $|x| \geq 1.0 * 2^{16}$ .

**Table 14-13. Overflow Condition for VCVTPS2PH**

Instruction	Condition	Masked Result	Unmasked Result
VCVTPS2PH	Result $\geq$ largest destination precision final normal value <sup>1</sup>	Result = +Inf or -Inf. #OE=1.	#OE=1.

VCVTPS2PH can cause an inexact exception if the result of the conversion is not exactly representable in the destination format.

**Table 14-14. Inexact Condition for VCVTPS2PH**

Instruction	Condition	Masked Result <sup>1</sup>	Unmasked Result
VCVTPS2PH	The result is not representable in the destination format	res = Result rounded to the destination precision and using the bounded exponent, but only if no unmasked underflow or overflow conditions occur (this exception can occur in the presence of a masked underflow or overflow). #PE=1.	Only if no underflow/overflow condition occurred, or if the corresponding exceptions are masked: <ul style="list-style-type: none"> <li>▪ Set #OE if masked overflow and set result as described above for masked overflow.</li> <li>▪ Set #UE if masked underflow and set result as described above for masked underflow.</li> </ul> If neither underflow nor overflow, result equals the result rounded to the destination precision and using the bounded exponent set #PE = 1.

**NOTES:**

1. If a source is denormal relative to input format with DM masked and at least one of PM or UM unmasked, then an exception will be raised with DE, UE and PE set.

### 14.4.1 Detection of F16C Instructions

Application using float 16 instruction must follow a detection sequence similar to AVX to ensure:

- The OS has enabled YMM state management support,
- The processor support AVX as indicated by the CPUID feature flag, i.e. CPUID.01H:ECX.AVX[bit 28] = 1.
- The processor support 16-bit floating-point conversion instructions via a CPUID feature flag (CPUID.01H:ECX.F16C[bit 29] = 1).

Application detection of Float-16 conversion instructions follow the general procedural flow in Figure 14-3.

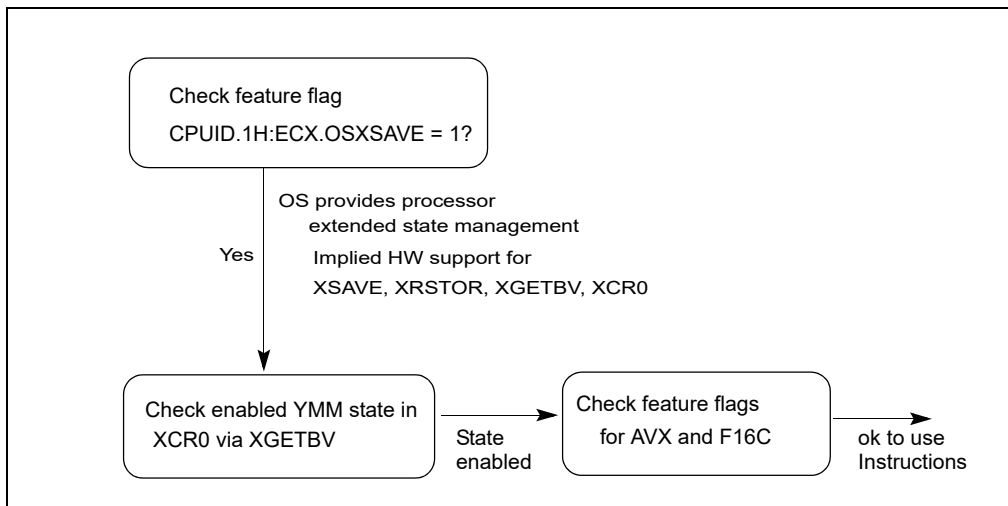


Figure 14-3. General Procedural Flow of Application Detection of Float-16

```

-----
INT supports_f16c()
{
    ; result in eax
    mov eax, 1
    cpuid
    and ecx, 038000000H
    cmp ecx, 038000000H; check OSXSAVE, AVX, F16C feature flags
    jne not_supported
    ; processor supports AVX,F16C instructions and XGETBV is enabled by OS
    mov ecx, 0; specify 0 for XCR0 register
    XGETBV; result in EDX:EAX
    and eax, 06H
    cmp eax, 06H; check OS has enabled both XMM and YMM state support
    jne not_supported
    mov eax, 1
    jmp done
NOT_SUPPORTED:
    mov eax, 0
done:
}
-----
  
```

## 14.5 FUSED-MULTIPLY-ADD (FMA) EXTENSIONS

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

FMA extensions also provide 60 128-bit floating-point instructions to process 128-bit vector and scalar data. The arithmetic operations cover fused multiply-add, fused multiply-subtract, signed-reversed multiply on fused multiply-add and multiply-subtract.

**Table 14-15. FMA Instructions**

Instruction	Description
VFMADD132PD/VFMADD213PD/VFMADD231PD xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Multiply-Add of Packed Double-Precision Floating-Point Values
VFMADD132PS/VFMADD213PS/VFMADD231PS xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Multiply-Add of Packed Single-Precision Floating-Point Values
VFMADD132SD/VFMADD213SD/VFMADD231SD xmm0, xmm1, xmm2/m64	Fused Multiply-Add of Scalar Double-Precision Floating-Point Values
VFMADD132SS/VFMADD213SS/VFMADD231SS xmm0, xmm1, xmm2/m32	Fused Multiply-Add of Scalar Single-Precision Floating-Point Values
VFMADDSUB132PD/VFMADDSUB213PD/VFMADDSUB231PD xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Multiply-Alternating Add/Subtract of Packed Double-Precision Floating-Point Values
VFMADDSUB132PS/VFMADDSUB213PS/VFMADDSUB231PS xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Multiply-Alternating Add/Subtract of Packed Single-Precision Floating-Point Values
VFMSUBADD132PD/VFMSUBADD213PD/VFMSUBADD231PD xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Multiply-Alternating Subtract/Add of Packed Double-Precision Floating-Point Values
VFMSUBADD132PS/VFMSUBADD213PS/VFMSUBADD231PS xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Multiply-Alternating Subtract/Add of Packed Single-Precision Floating-Point Values
VFMSUB132PD/VFMSUB213PD/VFMSUB231PD xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Multiply-Subtract of Packed Double-Precision Floating-Point Values
VFMSUB132PS/VFMSUB213PS/VFMSUB231PS xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Multiply-Subtract of Packed Single-Precision Floating-Point Values
VFMSUB132SD/VFMSUB213SD/VFMSUB231SD xmm0, xmm1, xmm2/m64	Fused Multiply-Subtract of Scalar Double-Precision Floating-Point Values
VFMSUB132SS/VFMSUB213SS/VFMSUB231SS xmm0, xmm1, xmm2/m32	Fused Multiply-Subtract of Scalar Single-Precision Floating-Point Values
VFNMADD132PD/VFNMADD213PD/VFNMADD231PD xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Negative Multiply-Add of Packed Double-Precision Floating-Point Values
VFNMADD132PS/VFNMADD213PS/VFNMADD231PS xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Negative Multiply-Add of Packed Single-Precision Floating-Point Values
VFNMADD132SD/VFNMADD213SD/VFNMADD231SD xmm0, xmm1, xmm2/m64	Fused Negative Multiply-Add of Scalar Double-Precision Floating-Point Values
VFNMADD132SS/VFNMADD213SS/VFNMADD231SS xmm0, xmm1, xmm2/m32	Fused Negative Multiply-Add of Scalar Single-Precision Floating-Point Values
VFNMSUB132PD/VFNMSUB213PD/VFNMSUB231PD xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Negative Multiply-Subtract of Packed Double-Precision Floating-Point Values
VFNMSUB132PS/VFNMSUB213PS/VFNMSUB231PS xmm0, xmm1, xmm2/m128; ymm0, ymm1, ymm2/m256	Fused Negative Multiply-Subtract of Packed Single-Precision Floating-Point Values

Table 14-15. FMA Instructions

Instruction	Description
VFNMSUB132SD/VFNMSUB213SD/VFNMSUB231SD xmm0, xmm1, xmm2/m64	Fused Negative Multiply-Subtract of Scalar Double-Precision Floating-Point Values
VFNMSUB132SS/VFNMSUB213SS/VFNMSUB231SS xmm0, xmm1, xmm2/m32	Fused Negative Multiply-Subtract of Scalar Single-Precision Floating-Point Values

### 14.5.1 FMA Instruction Operand Order and Arithmetic Behavior

FMA instruction mnemonics are defined explicitly with an ordered three digits, e.g. VFMADD132PD. The value of each digit refers to the ordering of the three source operand as defined by instruction encoding specification:

- '1': The first source operand (also the destination operand) in the syntactical order listed in this specification.
- '2': The second source operand in the syntactical order. This is a YMM/XMM register, encoded using VEX prefix.
- '3': The third source operand in the syntactical order. The first and third operand are encoded following ModR/M encoding rules.

The ordering of each digit within the mnemonic refers to the floating-point data listed on the right-hand side of the arithmetic equation of each FMA operation (see Table 14-17):

- The first position in the three digits of a FMA mnemonic refers to the operand position of the first FP data expressed in the arithmetic equation of FMA operation, the multiplicand.
- The second position in the three digits of a FMA mnemonic refers to the operand position of the second FP data expressed in the arithmetic equation of FMA operation, the multiplier.
- The third position in the three digits of a FMA mnemonic refers to the operand position of the FP data being added/subtracted to the multiplication result.

Note the non-numerical result of an FMA operation does not resemble the mathematically-defined commutative property between the multiplicand and the multiplier values (see Table 14-17). Consequently, software tools (such as an assembler) may support a complementary set of FMA mnemonics for each FMA instruction for ease of programming to take advantage of the mathematical property of commutative multiplications. For example, an assembler may optionally support the complementary mnemonic "VFMADD312PD" in addition to the true mnemonic "VFMADD132PD". The assembler will generate the same instruction opcode sequence corresponding to VFMADD132PD. The processor executes VFMADD132PD and report any NAN conditions based on the definition of VFMADD132PD. Similarly, if the complementary mnemonic VFMADD123PD is supported by an assembler at source level, it must generate the opcode sequence corresponding to VFMADD213PD; the complementary mnemonic VFMADD321PD must produce the opcode sequence defined by VFMADD231PD. In the absence of FMA operations reporting a NAN result, the numerical results of using either mnemonic with an assembler supporting both mnemonics will match the behavior defined in Table 14-17. Support for the complementary FMA mnemonics by software tools is optional.

### 14.5.2 Fused-Multiply-ADD (FMA) Numeric Behavior

FMA instructions can perform fused-multiply-add operations (including fused-multiply-subtract, and other varieties) on packed and scalar data elements in the instruction operands. Separate FMA instructions are provided to handle different types of arithmetic operations on the three source operands.

FMA instruction syntax is defined using three source operands and the first source operand is updated based on the result of the arithmetic operations of the data elements of 128-bit or 256-bit operands, i.e. The first source operand is also the destination operand.

The arithmetic FMA operation performed in an FMA instruction takes one of several forms,  $r=(x*y)+z$ ,  $r=(x*y)-z$ ,  $r=-(x*y)+z$ , or  $r=-(x*y)-z$ . Packed FMA instructions can perform eight single-precision FMA operations or four double-precision FMA operations with 256-bit vectors.

Scalar FMA instructions only perform one arithmetic operation on the low order data element. The content of the rest of the data elements in the lower 128-bits of the destination operand is preserved. the upper 128bits of the destination operand are filled with zero.

An arithmetic FMA operation of the form,  $r=(x*y)+z$ , takes two IEEE-754-2008 single (double) precision values and multiplies them to form an infinite precision intermediate value. This intermediate value is added to a third single (double) precision value (also at infinite precision) and rounded to produce a single (double) precision result.

Table 14-17 describes the numerical behavior of the FMA operation,  $r=(x*y)+z$ ,  $r=(x*y)-z$ ,  $r=-(x*y)+z$ ,  $r=-(x*y)-z$  for various input values. The input values can be 0, finite non-zero (F in Table 14-17), infinity of either sign (INF in Table 14-17), positive infinity (+INF in Table 14-17), negative infinity (-INF in Table 14-17), or NaN (including QNaN or SNaN). If any one of the input values is a NaN, the result of FMA operation, r, may be a quietized NaN. The result can be either Q(x), Q(y), or Q(z), see Table 14-17. If x is a NaN, then:

- $Q(x) = x$  if x is QNaN or
- $Q(x) =$  the quietized NaN obtained from x if x is SNaN

The notation for the output value in Table 14-17 are:

- “+INF”: positive infinity, “-INF”: negative infinity. When the result depends on a conditional expression, both values are listed in the result column and the condition is described in the comment column.
- QNaNIndefinite represents the QNaN which has the sign bit equal to 1, the most significant field equal to 1, and the remaining significant field bits equal to 0.
- The summation or subtraction of 0s or identical values in FMA operation can lead to the following situations shown in Table 14-16
- If the FMA computation represents an invalid operation (e.g. when adding two INF with opposite signs), the invalid exception is signaled, and the MXCSR.IE flag is set.

**Table 14-16. Rounding Behavior of Zero Result in FMA Operation**

$x*y$	$z$	$(x*y) + z$	$(x*y) - z$	$-(x*y) + z$	$-(x*y) - z$
(+0)	(+0)	+0 in all rounding modes	- 0 when rounding down, and +0 otherwise	- 0 when rounding down, and +0 otherwise	- 0 in all rounding modes
(+0)	(-0)	- 0 when rounding down, and +0 otherwise	+0 in all rounding modes	- 0 in all rounding modes	- 0 when rounding down, and +0 otherwise
(-0)	(+0)	- 0 when rounding down, and +0 otherwise	- 0 in all rounding modes	+ 0 in all rounding modes	- 0 when rounding down, and +0 otherwise
(-0)	(-0)	- 0 in all rounding modes	- 0 when rounding down, and +0 otherwise	- 0 when rounding down, and +0 otherwise	+ 0 in all rounding modes
F	-F	- 0 when rounding down, and +0 otherwise	$2^*F$	$-2^*F$	- 0 when rounding down, and +0 otherwise
F	F	$2^*F$	- 0 when rounding down, and +0 otherwise	- 0 when rounding down, and +0 otherwise	$-2^*F$

**Table 14-17. FMA Numeric Behavior**

$x$ (multiplicand)	$y$ (multiplier)	$z$	$r=(x*y)+z$	$r=(x*y)-z$	$r = -(x*y)+z$	$r= -(x*y)-z$	Comment
NaN	0, F, INF, NaN	0, F, INF, NaN	Q(x)	Q(x)	Q(x)	Q(x)	Signal invalid exception if x or y or z is SNaN
0, F, INF	NaN	0, F, INF, NaN	Q(y)	Q(y)	Q(y)	Q(y)	Signal invalid exception if y or z is SNaN
0, F, INF	0, F, INF	NaN	Q(z)	Q(z)	Q(z)	Q(z)	Signal invalid exception if z is SNaN
INF	F, INF	+INF F	+INF	QNaNIn definite	QNaNIndefinite	-INF	if $x*y$ and z have the same sign
			QNaNIn definite	-INF	+INF	QNaNIndefinite	if $x*y$ and z have opposite signs



x (multiplicand)	y (multiplier)	z	$r=(x*y)+z$	$r=(x*y)-z$	$r = -(x*y)+z$	$r = -(x*y)-z$	Comment
INF	F, INF	-INF	-INF	QNaNIn definite	QNaNInd efinite	+INF	if $x*y$ and $z$ have the same sign
			QNaNIn definite	+INF	-INF	QNaNInd efinite	if $x*y$ and $z$ have opposite signs
INF	F, INF	0, F	+INF	+INF	-INF	-INF	if $x$ and $y$ have the same sign
			-INF	-INF	+INF	+INF	if $x$ and $y$ have opposite signs
INF	0	0, F, INF	QNaNIn definite	QNaNIn definite	QNaNInd efinite	QNaNInd efinite	Signal invalid exception
0	INF	0, F, INF	QNaNIn definite	QNaNIn definite	QNaNInd efinite	QNaNInd efinite	Signal invalid exception
F	INF	+INF, F	+INF	QNaNIn definite	QNaNInd efinite	-INF	if $x*y$ and $z$ have the same sign
			QNaNIn definite	-INF	+INF	QNaNInd efinite	if $x*y$ and $z$ have opposite signs
F	INF	-INF	-INF	QNaNIn definite	QNaNInd efinite	+INF	if $x*y$ and $z$ have the same sign
			QNaNIn definite	+INF	-INF	QNaNInd efinite	if $x*y$ and $z$ have opposite signs
F	INF	0, F	+INF	+INF	-INF	-INF	if $x * y > 0$
			-INF	-INF	+INF	+INF	if $x * y < 0$
0, F	0, F	INF	+INF	-INF	+INF	-INF	if $z > 0$
			-INF	+INF	-INF	+INF	if $z < 0$
0	0	0	0	0	0	0	The sign of the result depends on the sign of the operands and on the rounding mode. The product $x*y$ is +0 or -0, depending on the signs of $x$ and $y$ . The summation/subtraction of the zero representing $(x*y)$ and the zero representing $z$ can lead to one of the four cases shown in Table 14-16.
0	F	0	0	0	0	0	
F	0	0	0	0	0	0	
F	F	0	0	0	0	0	
0	0	F	$z$	$-z$	$z$	$-z$	
0	F	F	$z$	$-z$	$z$	$-z$	
F	0	F	$z$	$-z$	$z$	$-z$	
F	F	0	$x*y$	$x*y$	$-x*y$	$-x*y$	Rounded to the destination precision, with bounded exponent
F	F	F	$(x*y)+z$	$(x*y)-z$	$-(x*y)+z$	$-(x*y)-z$	Rounded to the destination precision, with bounded exponent; however, if the exact values of $x*y$ and $z$ are equal in magnitude with signs resulting in the FMA operation producing 0, the rounding behavior described in Table 14-16.

If unmasked floating-point exceptions are signaled (invalid operation, denormal operand, overflow, underflow, or inexact result) the result register is left unchanged and a floating-point exception handler is invoked.

### 14.5.3 Detection of FMA

Hardware support for FMA is indicated by `CPUID.1:ECX.FMA[bit 12]=1`.

Application Software must identify that hardware supports AVX, after that it must also detect support for FMA by `CPUID.1:ECX.FMA[bit 12]`. The recommended pseudocode sequence for detection of FMA is:

```

-----
INT supports_fma()
{
    ; result in eax
    mov eax, 1
    cpuid
    and ecx, 018001000H
    cmp ecx, 018001000H; check OSXSAVE, AVX, FMA feature flags
    jne not_supported
    ; processor supports AVX,FMA instructions and XGETBV is enabled by OS
    mov ecx, 0; specify 0 for XCR0 register
    XGETBV; result in EDX:EAX
    and eax, 06H
    cmp eax, 06H; check OS has enabled both XMM and YMM state support
    jne not_supported
    mov eax, 1
    jmp done
NOT_SUPPORTED:
    mov eax, 0
done:
}
-----

```

Note that FMA comprises 256-bit and 128-bit SIMD instructions operating on YMM states.

## 14.6 OVERVIEW OF INTEL® ADVANCED VECTOR EXTENSIONS 2 (INTEL® AVX2)

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

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

### 14.6.1 AVX2 and 256-bit Vector Integer Processing

AVX2 promotes the vast majority of 128-bit integer SIMD instruction sets to operate with 256-bit wide YMM registers. AVX2 instructions are encoded using the VEX prefix and require the same operating system support as AVX. Generally, most of the promoted 256-bit vector integer instructions follow the 128-bit lane operation, similar to the promoted 256-bit floating-point SIMD instructions in AVX.

Newer functionalities in AVX2 generally fall into the following categories:

- Fetching non-contiguous data elements from memory using vector-index memory addressing. These “gather” instructions introduce a new memory-addressing form, consisting of a base register and multiple indices specified by a vector register (either XMM or YMM). Data elements sizes of 32 and 64-bits are supported, and data types for floating-point and integer elements are also supported.
- Cross-lane functionalities are provided with several new instructions for broadcast and permute operations. Some of the 256-bit vector integer instructions promoted from legacy SSE instruction sets also exhibit cross-lane behavior, e.g. VPMOVS/VPPOVS family.
- AVX2 complements the AVX instructions that are typed for floating-point operation with a full compliment of equivalent set for operating with 32/64-bit integer data elements.

- Vector shift instructions with per-element shift count. Data elements sizes of 32 and 64-bits are supported.

## 14.7 PROMOTED VECTOR INTEGER INSTRUCTIONS IN AVX2

In AVX2, most SSE/SSE2/SSE3/SSSE3/SSE4 vector integer instructions have been promoted to support VEX.256 encodings. Table 14-18 summarizes the promotion status for existing instructions. The column “VEX.128” indicates whether the instruction using VEX.128 prefix encoding is supported.

The column “VEX.256” indicates whether 256-bit vector form of the instruction using the VEX.256 prefix encoding is supported, and under which feature flag.

**Table 14-18. Promoted Vector Integer SIMD Instructions in AVX2**

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction
AVX2	AVX	YY OF 6X	PUNPCKLBW
AVX2	AVX		PUNPCKLWD
AVX2	AVX		PUNPCKLDQ
AVX2	AVX		PACKSSWB
AVX2	AVX		PCMPGTB
AVX2	AVX		PCMPGTW
AVX2	AVX		PCMPGTD
AVX2	AVX		PACKUSWB
AVX2	AVX		PUNPCKHBW
AVX2	AVX		PUNPCKHWD
AVX2	AVX		PUNPCKHDQ
AVX2	AVX		PACKSSDW
AVX2	AVX		PUNPCKLQDQ
AVX2	AVX		PUNPCKHQDQ
no	AVX		MOVD
no	AVX		MOVQ
AVX	AVX		MOVDQA
AVX	AVX		MOVDQU
AVX2	AVX	YY OF 7X	PSHUFD
AVX2	AVX		PSHUFW
AVX2	AVX		PSHUFLW
AVX2	AVX		PCMPEQB
AVX2	AVX		PCMPEQW
AVX2	AVX		PCMPEQD
AVX	AVX		MOVDQA
AVX	AVX		MOVDQU
no	AVX		PINSRW
no	AVX		PEXTRW
AVX2	AVX		PSRLW
AVX2	AVX		PSRLD

Table 14-18. Promoted Vector Integer SIMD Instructions in AVX2

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction
AVX2	AVX		PSRLQ
AVX2	AVX		PADDQ
AVX2	AVX		PMULLW
AVX2	AVX		PMOVBMSKB
AVX2	AVX		PSUBUSB
AVX2	AVX		PSUBUSW
AVX2	AVX		PMINUB
AVX2	AVX		PAND
AVX2	AVX		PADDUSB
AVX2	AVX		PADDUSW
AVX2	AVX		PMAXUB
AVX2	AVX		PANDN
AVX2	AVX	YY OF EX	PAVGB
AVX2	AVX		PSRAW
AVX2	AVX		PSRAD
AVX2	AVX		PAVGW
AVX2	AVX		PMULHUW
AVX2	AVX		PMULHW
AVX	AVX		MOVNTDQ
AVX2	AVX		PSUBSB
AVX2	AVX		PSUBSW
AVX2	AVX		PMINSW
AVX2	AVX		POR
AVX2	AVX		PADDSB
AVX2	AVX		PADDSW
AVX2	AVX		PMAXSW
AVX2	AVX		PXOR
AVX	AVX	YY OF FX	LDDQU
AVX2	AVX		PSLLW
AVX2	AVX		PSLLD
AVX2	AVX		PSLLQ
AVX2	AVX		PMULUDQ
AVX2	AVX		PMADDWD
AVX2	AVX		PSADBW
AVX2	AVX		PSUBB
AVX2	AVX		PSUBW
AVX2	AVX		PSUBD
AVX2	AVX		PSUBQ

**Table 14-18. Promoted Vector Integer SIMD Instructions in AVX2**

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction
AVX2	AVX		PADDB
AVX2	AVX		PADDW
AVX2	AVX		PADDD
AVX2	AVX	SSSE3	PHADDW
AVX2	AVX		PHADDSW
AVX2	AVX		PHADDD
AVX2	AVX		PHSUBW
AVX2	AVX		PHSUBSW
AVX2	AVX		PHSUBD
AVX2	AVX		PMADDUBSW
AVX2	AVX		PALIGNR
AVX2	AVX		PSHUFB
AVX2	AVX		PMULHRSW
AVX2	AVX		PSIGNB
AVX2	AVX		PSIGNW
AVX2	AVX		PSIGND
AVX2	AVX		PABSB
AVX2	AVX		PABSW
AVX2	AVX		PABSD
AVX2	AVX		MOVNTDQA
AVX2	AVX		MPSADBW
AVX2	AVX		PACKUSDW
AVX2	AVX		PBLENDVB
AVX2	AVX		PBLENDW
AVX2	AVX		PCMPEQQ
no	AVX		PEXTRD
no	AVX		PEXTRQ
no	AVX		PEXTRB
no	AVX		PEXTRW
no	AVX		PHMINPOSUW
no	AVX		PINSRB
no	AVX		PINSRD
no	AVX		PINSRQ
AVX2	AVX		PMAXSB
AVX2	AVX		PMAXSD
AVX2	AVX		PMAXUD
AVX2	AVX		PMAXUW
AVX2	AVX		PMINSB

**Table 14-18. Promoted Vector Integer SIMD Instructions in AVX2**

VEX.256 Encoding	VEX.128 Encoding	Group	Instruction
AVX2	AVX		PMINSD
AVX2	AVX		PMINUD
AVX2	AVX		PMINUW
AVX2	AVX		PMOVSXxx
AVX2	AVX		PMOVZXxx
AVX2	AVX		PMULDQ
AVX2	AVX		PMULLD
AVX	AVX		PTEST
AVX2	AVX	SSE4.2	PCMPGTQ
no	AVX		PCMPSTR
no	AVX		PCMPSTRM
no	AVX		PCMPISTR
no	AVX		PCMPISTRM
no	AVX	AESNI	AESDEC
no	AVX		AESDECLAST
no	AVX		AESENC
no	AVX		AESECNLAST
no	AVX		AESIMC
no	AVX		AESKEYGENASSIST
no	AVX	CLMUL	PCLMULQDQ

Table 14-19 compares complementary SIMD functionalities introduced in AVX and AVX2. instructions.

**Table 14-19. VEX-Only SIMD Instructions in AVX and AVX2**

AVX2	AVX	Comment
VBROADCASTI128	VBROADCASTF128	256-bit only
VBROADCASTSD ymm1, xmm	VBROADCASTSD ymm1, m64	256-bit only
VBROADCASTSS (from xmm)	VBROADCASTSS (from m32)	
VEXTRACTI128	VEXTRACTF128	256-bit only
VINSERTI128	VINSERTF128	256-bit only
VPMASKMOVD	VMASKMOVPS	
VPMASKMOVQ!	VMASKMOVPD	
	VPERMILPD	in-lane
	VPERMILPS	in-lane
VPERM2I128	VPERM2F128	256-bit only
VPERMD		cross-lane
VPERMPS		cross-lane
VPERMQ		cross-lane
VPERMPD		cross-lane

**Table 14-19. VEX-Only SIMD Instructions in AVX and AVX2**

AVX2	AVX	Comment
	VTESTPD	
	VTESTPS	
VPBLENDQ		
VPSLLVD/Q		
VPSRAVD		
VPSRLVD/Q		
VGATHERDPD/QPD		
VGATHERDPS/QPS		
VPGATHERDD/QD		
VPGATHERDQ/QQ		

**Table 14-20. New Primitive in AVX2 Instructions**

Instruction	Description
VPERMD ymm1, ymm2, ymm3/m256	Permute doublewords in ymm3/m256 using indexes in ymm2 and store the result in ymm1.
VPERMPD ymm1, ymm2/m256, imm8	Permute double-precision FP elements in ymm2/m256 using indexes in imm8 and store the result in ymm1.
VPERMPS ymm1, ymm2, ymm3/m256	Permute single-precision FP elements in ymm3/m256 using indexes in ymm2 and store the result in ymm1.
VPERMQ ymm1, ymm2/m256, imm8	Permute quadwords in ymm2/m256 using indexes in imm8 and store the result in ymm1.
VPSLLVD xmm1, xmm2, xmm3/m128	Shift doublewords in xmm2 left by amount specified in the corresponding element of xmm3/m128 while shifting in 0s.
VPSLLVQ xmm1, xmm2, xmm3/m128	Shift quadwords in xmm2 left by amount specified in the corresponding element of xmm3/m128 while shifting in 0s.
VPSLLVD ymm1, ymm2, ymm3/m256	Shift doublewords in ymm2 left by amount specified in the corresponding element of ymm3/m256 while shifting in 0s.
VPSLLVQ ymm1, ymm2, ymm3/m256	Shift quadwords in ymm2 left by amount specified in the corresponding element of ymm3/m256 while shifting in 0s.
VPSRAVD xmm1, xmm2, xmm3/m128	Shift doublewords in xmm2 right by amount specified in the corresponding element of xmm3/m128 while shifting in the sign bits.
VPSRLVD xmm1, xmm2, xmm3/m128	Shift doublewords in xmm2 right by amount specified in the corresponding element of xmm3/m128 while shifting in 0s.
VPSRLVQ xmm1, xmm2, xmm3/m128	Shift quadwords in xmm2 right by amount specified in the corresponding element of xmm3/m128 while shifting in 0s.
VPSRLVD ymm1, ymm2, ymm3/m256	Shift doublewords in ymm2 right by amount specified in the corresponding element of ymm3/m256 while shifting in 0s.
VPSRLVQ ymm1, ymm2, ymm3/m256	Shift quadwords in ymm2 right by amount specified in the corresponding element of ymm3/m256 while shifting in 0s.
VGATHERDD xmm1, vm32x, xmm2	Using dword indices specified in vm32x, gather dword values from memory conditioned on mask specified by xmm2. Conditionally gathered elements are merged into xmm1.
VGATHERQD xmm1, vm64x, xmm2	Using qword indices specified in vm64x, gather dword values from memory conditioned on mask specified by xmm2. Conditionally gathered elements are merged into xmm1.
VGATHERDD ymm1, vm32y, ymm2	Using dword indices specified in vm32y, gather dword values from memory conditioned on mask specified by ymm2. Conditionally gathered elements are merged into ymm1.
VGATHERQD ymm1, vm64y, ymm2	Using qword indices specified in vm64y, gather dword values from memory conditioned on mask specified by ymm2. Conditionally gathered elements are merged into ymm1.

Instruction	Description
VGATHERDPD xmm1, vm32x, xmm2	Using dword indices specified in vm32x, gather double-precision FP values from memory conditioned on mask specified by xmm2. Conditionally gathered elements are merged into xmm1.
VGATHERQPD xmm1, vm64x, xmm2	Using qword indices specified in vm64x, gather double-precision FP values from memory conditioned on mask specified by xmm2. Conditionally gathered elements are merged into xmm1.
VGATHERDPD ymm1, vm32x, ymm2	Using dword indices specified in vm32x, gather double-precision FP values from memory conditioned on mask specified by ymm2. Conditionally gathered elements are merged into ymm1.
VGATHERQPD ymm1, vm64y, ymm2	Using qword indices specified in vm64y, gather double-precision FP values from memory conditioned on mask specified by ymm2. Conditionally gathered elements are merged into ymm1.
VGATHERDPS xmm1, vm32x, xmm2	Using dword indices specified in vm32x, gather single-precision FP values from memory conditioned on mask specified by xmm2. Conditionally gathered elements are merged into xmm1.
VGATHERQPS xmm1, vm64x, xmm2	Using qword indices specified in vm64x, gather single-precision FP values from memory conditioned on mask specified by xmm2. Conditionally gathered elements are merged into xmm1.
VGATHERDPS ymm1, vm32y, ymm2	Using dword indices specified in vm32y, gather single-precision FP values from memory conditioned on mask specified by ymm2. Conditionally gathered elements are merged into ymm1.
VGATHERQPS ymm1, vm64y, ymm2	Using qword indices specified in vm64y, gather single-precision FP values from memory conditioned on mask specified by ymm2. Conditionally gathered elements are merged into ymm1.
VGATHERDQ xmm1, vm32x, xmm2	Using dword indices specified in vm32x, gather qword values from memory conditioned on mask specified by xmm2. Conditionally gathered elements are merged into xmm1.
VGATHERQQ xmm1, vm64x, xmm2	Using qword indices specified in vm64x, gather qword values from memory conditioned on mask specified by xmm2. Conditionally gathered elements are merged into xmm1.
VGATHERDQ ymm1, vm32x, ymm2	Using dword indices specified in vm32x, gather qword values from memory conditioned on mask specified by ymm2. Conditionally gathered elements are merged into ymm1.
VGATHERQQ ymm1, vm64y, ymm2	Using qword indices specified in vm64y, gather qword values from memory conditioned on mask specified by ymm2. Conditionally gathered elements are merged into ymm1.

### 14.7.1 Detection of AVX2

Hardware support for AVX2 is indicated by CPUID.(EAX=07H, ECX=0H):EBX.AVX2[bit 5]=1.

Application Software must identify that hardware supports AVX, after that it must also detect support for AVX2 by checking CPUID.(EAX=07H, ECX=0H):EBX.AVX2[bit 5]. The recommended pseudocode sequence for detection of AVX2 is:

```

-----
INT supports_avx2()
{
    ; result in eax
    mov eax, 1
    cpuid
    and ecx, 018000000H
    cmp ecx, 018000000H; check both OSXSAVE and AVX feature flags
    jne not_supported
    ; processor supports AVX instructions and XGETBV is enabled by OS
    mov eax, 7
}

```



```

    mov ecx, 0
    cpuid
    and ebx, 20H
    cmp ebx, 20H; check AVX2 feature flags
    jne not_supported
    mov ecx, 0; specify 0 for XCR0 register
    XGETBV; result in EDX:EAX
    and eax, 06H
    cmp eax, 06H; check OS has enabled both XMM and YMM state support
    jne not_supported
    mov eax, 1
    jmp done
NOT_SUPPORTED:
    mov eax, 0
done:
}

```

---

## 14.8 ACCESSING YMM REGISTERS

The lower 128 bits of a YMM register is aliased to the corresponding XMM register. Legacy SSE instructions (i.e. SIMD instructions operating on XMM state but not using the VEX prefix, also referred to non-VEX encoded SIMD instructions) will not access the upper bits (255:128) of the YMM registers. AVX and FMA instructions with a VEX prefix and vector length of 128-bits zeroes the upper 128 bits of the YMM register.

Upper bits of YMM registers (255:128) can be read and written by many instructions with a VEX.256 prefix.

XSAVE and XRSTOR may be used to save and restore the upper bits of the YMM registers.

## 14.9 MEMORY ALIGNMENT

Memory alignment requirements on VEX-encoded instruction differs from non-VEX-encoded instructions. Memory alignment applies to non-VEX-encoded SIMD instructions in three categories:

- Explicitly-aligned SIMD load and store instructions accessing 16 bytes of memory (e.g. MOVAPD, MOVAPS, MOVDQA, etc.). These instructions always require memory address to be aligned on 16-byte boundary.
- Explicitly-unaligned SIMD load and store instructions accessing 16 bytes or less of data from memory (e.g. MOVUPD, MOVUPS, MOVDQU, MOVQ, MOVD, etc.). These instructions do not require memory address to be aligned on 16-byte boundary.
- The vast majority of arithmetic and data processing instructions in legacy SSE instructions (non-VEX-encoded SIMD instructions) support memory access semantics. When these instructions access 16 bytes of data from memory, the memory address must be aligned on 16-byte boundary.

Most arithmetic and data processing instructions encoded using the VEX prefix and performing memory accesses have more flexible memory alignment requirements than instructions that are encoded without the VEX prefix. Specifically,

- With the exception of explicitly aligned 16 or 32 byte SIMD load/store instructions, most VEX-encoded, arithmetic and data processing instructions operate in a flexible environment regarding memory address alignment, i.e. VEX-encoded instruction with 32-byte or 16-byte load semantics will support unaligned load operation by default. Memory arguments for most instructions with VEX prefix operate normally without

causing #GP(0) on any byte-granularity alignment (unlike Legacy SSE instructions). The instructions that require explicit memory alignment requirements are listed in Table 14-22.

Software may see performance penalties when unaligned accesses cross cacheline boundaries, so reasonable attempts to align commonly used data sets should continue to be pursued.

Atomic memory operation in Intel 64 and IA-32 architecture is guaranteed only for a subset of memory operand sizes and alignment scenarios. The list of guaranteed atomic operations are described in Section 8.1.1 of *IA-32 Intel® Architecture Software Developer's Manual, Volumes 3A*. AVX and FMA instructions do not introduce any new guaranteed atomic memory operations.

AVX instructions can generate an #AC(0) fault on misaligned 4 or 8-byte memory references in Ring-3 when CR0.AM=1. 16 and 32-byte memory references will not generate #AC(0) fault. See Table 14-21 for details.

Certain AVX instructions always require 16- or 32-byte alignment (see the complete list of such instructions in Table 14-22). These instructions will #GP(0) if not aligned to 16-byte boundaries (for 16-byte granularity loads and stores) or 32-byte boundaries (for 32-byte loads and stores).

**Table 14-21. Alignment Faulting Conditions when Memory Access is Not Aligned**

		EFLAGS.AC==1 && Ring-3 && CR0.AM == 1	0	1
Instruction Type	AVX, FMA,	16- or 32-byte "explicitly unaligned" loads and stores (see Table 14-23)	no fault	no fault
		VEX op YMM, m256	no fault	no fault
		VEX op XMM, m128	no fault	no fault
		"explicitly aligned" loads and stores (see Table 14-22)	#GP(0)	#GP(0)
		2, 4, or 8-byte loads and stores	no fault	#AC(0)
	SSE	16 byte "explicitly unaligned" loads and stores (see Table 14-23)	no fault	no fault
		op XMM, m128	#GP(0)	#GP(0)
		"explicitly aligned" loads and stores (see Table 14-22)	#GP(0)	#GP(0)
		2, 4, or 8-byte loads and stores	no fault	#AC(0)

**Table 14-22. Instructions Requiring Explicitly Aligned Memory**

Require 16-byte alignment	Require 32-byte alignment
(V)MOVDQA xmm, m128	VMOVDQA ymm, m256
(V)MOVDQA m128, xmm	VMOVDQA m256, ymm
(V)MOVAPS xmm, m128	VMOVAPS ymm, m256
(V)MOVAPS m128, xmm	VMOVAPS m256, ymm
(V)MOVAPD xmm, m128	VMOVAPD ymm, m256
(V)MOVAPD m128, xmm	VMOVAPD m256, ymm
(V)MOVNTPS m128, xmm	VMOVNTPS m256, ymm
(V)MOVNTPD m128, xmm	VMOVNTPD m256, ymm
(V)MOVNTDQ m128, xmm	VMOVNTDQ m256, ymm
(V)MOVNTDQA xmm, m128	VMOVNTDQA ymm, m256

**Table 14-23. Instructions Not Requiring Explicit Memory Alignment**

(V)MOVDQU xmm, m128
(V)MOVDQU m128, m128
(V)MOVUPS xmm, m128
(V)MOVUPS m128, xmm
(V)MOVUPD xmm, m128
(V)MOVUPD m128, xmm
VMOVDQU ymm, m256
VMOVDQU m256, ymm
VMOVUPS ymm, m256
VMOVUPS m256, ymm
VMOVUPD ymm, m256
VMOVUPD m256, ymm

## 14.10 SIMD FLOATING-POINT EXCEPTIONS

AVX instructions can generate SIMD floating-point exceptions (#XM) and respond to exception masks in the same way as Legacy SSE instructions. When CR4.OSXMMEXCPT=0 any unmasked FP exceptions generate an Undefined Opcode exception (#UD).

AVX FP exceptions are created in a similar fashion (differing only in number of elements) to Legacy SSE and SSE2 instructions capable of generating SIMD floating-point exceptions.

AVX introduces no new arithmetic operations (AVX floating-point are analogues of existing Legacy SSE instructions).

F16C, FMA instructions can generate SIMD floating-point exceptions (#XM). The requirement that apply to AVX also apply to F16C and FMA.

The subset of AVX2 instructions that operate on floating-point data do not generate #XM.

The detailed exception conditions for AVX instructions and legacy SIMD instructions (excluding instructions that operates on MMX registers) are described in a number of exception class types, depending on the operand syntax and memory operation characteristics. The complete list of SIMD instruction exception class types are defined in Chapter 2, "Instruction Format," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

## 14.11 EMULATION

Setting the CR0.EMbit to 1 provides a technique to emulate Legacy SSE floating-point instruction sets in software. This technique is not supported with AVX instructions.

If an operating system wishes to emulate AVX instructions, set XCR0[2:1] to zero. This will cause AVX instructions to #UD. Emulation of F16C, AVX2, and FMA by operating system can be done similarly as with emulating AVX instructions.

## 14.12 WRITING AVX FLOATING-POINT EXCEPTION HANDLERS

AVX and FMA floating-point exceptions are handled in an entirely analogous way to Legacy SSE floating-point exceptions. To handle unmasked SIMD floating-point exceptions, the operating system or executive must provide an exception handler. The section titled "SSE and SSE2 SIMD Floating-Point Exceptions" in Chapter 11, "Programming with Streaming SIMD Extensions 2 (SSE2)," describes the SIMD floating-point exception classes and gives suggestions for writing an exception handler to handle them.

To indicate that the operating system provides a handler for SIMD floating-point exceptions (`#XM`), the `CR4.OSXMMEXCPT` flag (bit 10) must be set.

The guidelines for writing AVX floating-point exception handlers also apply to F16C and FMA.

## 14.13 GENERAL PURPOSE INSTRUCTION SET ENHANCEMENTS

Enhancements in the general-purpose instruction set consist of several categories:

- A rich collection of instructions to manipulate integer data at bit-granularity. Most of the bit-manipulation instructions employ VEX-prefix encoding to support three-operand syntax with non-destructive source operands. Two of the bit-manipulating instructions (`LZCNT`, `TZCNT`) are not encoded using VEX. The VEX-encoded bit-manipulation instructions include: `ANDN`, `BEXTR`, `BLSI`, `BLSMSK`, `BLSR`, `BZHI`, `PEXT`, `PDEP`, `SARX`, `SHLX`, `SHRX`, and `RORX`.
- Enhanced integer multiply instruction (`MULX`) in conjunction with some of the bit-manipulation instructions allow software to accelerate calculation of large integer numerics (wider than 128-bits).
- `INVPCID` instruction targets system software that manages processor context IDs.



#### 4. 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*.

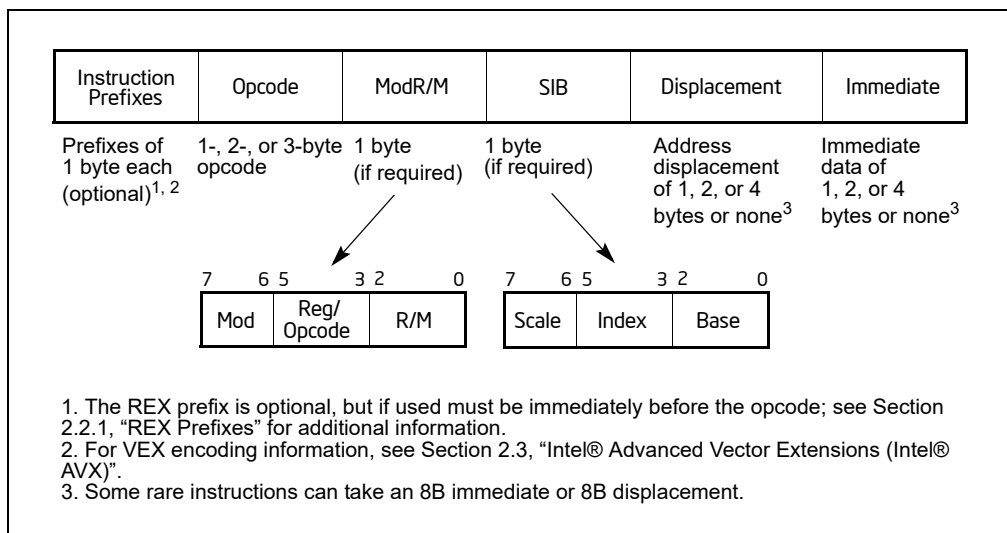
---

Change to this chapter: Updates to TupleType naming in Table 2-34 and Table 2-35. Updates to Table 2-39 “#UD Conditions of Operand-Encoding EVEX Prefix Bit Fields”, Table 2-40 “#UD Conditions of Opmask Related Encoding Field”, Table 2-46 “Type E2 Class Exception Conditions”, and Table 2-49 “Type E4 Class Exception Conditions”.

This chapter describes the instruction format for all Intel 64 and IA-32 processors. The instruction format for protected mode, real-address mode and virtual-8086 mode is described in Section 2.1. Increments provided for IA-32e mode and its sub-modes are described in Section 2.2.

## 2.1 INSTRUCTION FORMAT FOR PROTECTED MODE, REAL-ADDRESS MODE, AND VIRTUAL-8086 MODE

The Intel 64 and IA-32 architectures instruction encodings are subsets of the format shown in Figure 2-1. Instructions consist of optional instruction prefixes (in any order), primary opcode bytes (up to three bytes), an addressing-form specifier (if required) consisting of the ModR/M byte and sometimes the SIB (Scale-Index-Base) byte, a displacement (if required), and an immediate data field (if required).



**Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format**

### 2.1.1 Instruction Prefixes

Instruction prefixes are divided into four groups, each with a set of allowable prefix codes. For each instruction, it is only useful to include up to one prefix code from each of the four groups (Groups 1, 2, 3, 4). Groups 1 through 4 may be placed in any order relative to each other.

- Group 1
  - Lock and repeat prefixes:
    - LOCK prefix is encoded using F0H.
    - REPNE/REPZ prefix is encoded using F2H. Repeat-Not-Zero prefix applies only to string and input/output instructions. (F2H is also used as a mandatory prefix for some instructions.)
    - REP or REPE/REPZ is encoded using F3H. The repeat prefix applies only to string and input/output instructions. F3H is also used as a mandatory prefix for POPCNT, LZCNT and ADOX instructions.

## INSTRUCTION FORMAT

- BND prefix is encoded using F2H if the following conditions are true:
  - CPUID.(EAX=07H, ECX=0):EBX.MPX[bit 14] is set.
  - BNDCFGU.EN and/or IA32\_BNDCFGS.EN is set.
  - When the F2 prefix precedes a near CALL, a near RET, a near JMP, a short Jcc, or a near Jcc instruction (see Chapter 17, “Intel® MPX,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).
- Group 2
  - Segment override prefixes:
    - 2EH—CS segment override (use with any branch instruction is reserved).
    - 36H—SS segment override prefix (use with any branch instruction is reserved).
    - 3EH—DS segment override prefix (use with any branch instruction is reserved).
    - 26H—ES segment override prefix (use with any branch instruction is reserved).
    - 64H—FS segment override prefix (use with any branch instruction is reserved).
    - 65H—GS segment override prefix (use with any branch instruction is reserved).
  - Branch hints<sup>1</sup>:
    - 2EH—Branch not taken (used only with Jcc instructions).
    - 3EH—Branch taken (used only with Jcc instructions).
- Group 3
  - Operand-size override prefix is encoded using 66H (66H is also used as a mandatory prefix for some instructions).
- Group 4
  - 67H—Address-size override prefix.

The LOCK prefix (F0H) forces an operation that ensures exclusive use of shared memory in a multiprocessor environment. See “LOCK—Assert LOCK# Signal Prefix” in Chapter 3, “Instruction Set Reference, A-L,” for a description of this prefix.

Repeat prefixes (F2H, F3H) cause an instruction to be repeated for each element of a string. Use these prefixes only with string and I/O instructions (MOVS, CMPS, SCAS, LODS, STOS, INS, and OUTS). Use of repeat prefixes and/or undefined opcodes with other Intel 64 or IA-32 instructions is reserved; such use may cause unpredictable behavior.

Some instructions may use F2H,F3H as a mandatory prefix to express distinct functionality.

Branch hint prefixes (2EH, 3EH) allow a program to give a hint to the processor about the most likely code path for a branch. Use these prefixes only with conditional branch instructions (Jcc). Other use of branch hint prefixes and/or other undefined opcodes with Intel 64 or IA-32 instructions is reserved; such use may cause unpredictable behavior.

The operand-size override prefix allows a program to switch between 16- and 32-bit operand sizes. Either size can be the default; use of the prefix selects the non-default size.

Some SSE2/SSE3/SSSE3/SSE4 instructions and instructions using a three-byte sequence of primary opcode bytes may use 66H as a mandatory prefix to express distinct functionality.

Other use of the 66H prefix is reserved; such use may cause unpredictable behavior.

The address-size override prefix (67H) allows programs to switch between 16- and 32-bit addressing. Either size can be the default; the prefix selects the non-default size. Using this prefix and/or other undefined opcodes when operands for the instruction do not reside in memory is reserved; such use may cause unpredictable behavior.

---

1. Some earlier microarchitectures used these as branch hints, but recent generations have not and they are reserved for future hint usage.



## 2.1.2 Opcodes

A primary opcode can be 1, 2, or 3 bytes in length. An additional 3-bit opcode field is sometimes encoded in the ModR/M byte. Smaller fields can be defined within the primary opcode. Such fields define the direction of operation, size of displacements, register encoding, condition codes, or sign extension. Encoding fields used by an opcode vary depending on the class of operation.

Two-byte opcode formats for general-purpose and SIMD instructions consist of one of the following:

- An escape opcode byte 0FH as the primary opcode and a second opcode byte.
- A mandatory prefix (66H, F2H, or F3H), an escape opcode byte, and a second opcode byte (same as previous bullet).

For example, CVTQ2PD consists of the following sequence: F3 0F E6. The first byte is a mandatory prefix (it is not considered as a repeat prefix).

Three-byte opcode formats for general-purpose and SIMD instructions consist of one of the following:

- An escape opcode byte 0FH as the primary opcode, plus two additional opcode bytes.
- A mandatory prefix (66H, F2H, or F3H), an escape opcode byte, plus two additional opcode bytes (same as previous bullet).

For example, PHADDW for XMM registers consists of the following sequence: 66 0F 38 01. The first byte is the mandatory prefix.

Valid opcode expressions are defined in Appendix A and Appendix B.

## 2.1.3 ModR/M and SIB Bytes

Many instructions that refer to an operand in memory have an addressing-form specifier byte (called the ModR/M byte) following the primary opcode. The ModR/M byte contains three fields of information:

- The *mod* field combines with the *r/m* field to form 32 possible values: eight registers and 24 addressing modes.
- The *reg/opcode* field specifies either a register number or three more bits of opcode information. The purpose of the *reg/opcode* field is specified in the primary opcode.
- The *r/m* field can specify a register as an operand or it can be combined with the *mod* field to encode an addressing mode. Sometimes, certain combinations of the *mod* field and the *r/m* field are used to express opcode information for some instructions.

Certain encodings of the ModR/M byte require a second addressing byte (the SIB byte). The base-plus-index and scale-plus-index forms of 32-bit addressing require the SIB byte. The SIB byte includes the following fields:

- The *scale* field specifies the scale factor.
- The *index* field specifies the register number of the index register.
- The *base* field specifies the register number of the base register.

See Section 2.1.5 for the encodings of the ModR/M and SIB bytes.

## 2.1.4 Displacement and Immediate Bytes

Some addressing forms include a displacement immediately following the ModR/M byte (or the SIB byte if one is present). If a displacement is required, it can be 1, 2, or 4 bytes.

If an instruction specifies an immediate operand, the operand always follows any displacement bytes. An immediate operand can be 1, 2 or 4 bytes.

## 2.1.5 Addressing-Mode Encoding of ModR/M and SIB Bytes

The values and corresponding addressing forms of the ModR/M and SIB bytes are shown in Table 2-1 through Table 2-3: 16-bit addressing forms specified by the ModR/M byte are in Table 2-1 and 32-bit addressing forms are in Table 2-2. Table 2-3 shows 32-bit addressing forms specified by the SIB byte. In cases where the reg/opcode field in the ModR/M byte represents an extended opcode, valid encodings are shown in Appendix B.

In Table 2-1 and Table 2-2, the Effective Address column lists 32 effective addresses that can be assigned to the first operand of an instruction by using the Mod and R/M fields of the ModR/M byte. The first 24 options provide ways of specifying a memory location; the last eight (Mod = 11B) provide ways of specifying general-purpose, MMX technology and XMM registers.

The Mod and R/M columns in Table 2-1 and Table 2-2 give the binary encodings of the Mod and R/M fields required to obtain the effective address listed in the first column. For example: see the row indicated by Mod = 11B, R/M = 000B. The row identifies the general-purpose registers EAX, AX or AL; MMX technology register MM0; or XMM register XMM0. The register used is determined by the opcode byte and the operand-size attribute.

Now look at the seventh row in either table (labeled "REG ="). This row specifies the use of the 3-bit Reg/Opcode field when the field is used to give the location of a second operand. The second operand must be a general-purpose, MMX technology, or XMM register. Rows one through five list the registers that may correspond to the value in the table. Again, the register used is determined by the opcode byte along with the operand-size attribute. If the instruction does not require a second operand, then the Reg/Opcode field may be used as an opcode extension. This use is represented by the sixth row in the tables (labeled "/digit (Opcode)"). Note that values in row six are represented in decimal form.

The body of Table 2-1 and Table 2-2 (under the label "Value of ModR/M Byte (in Hexadecimal)") contains a 32 by 8 array that presents all of 256 values of the ModR/M byte (in hexadecimal). Bits 3, 4 and 5 are specified by the column of the table in which a byte resides. The row specifies bits 0, 1 and 2; and bits 6 and 7. The figure below demonstrates interpretation of one table value.

	Mod	11	
	RM		000
/digit (Opcode);	REG =	<b>001</b>	
	C8H	11	<b>001000</b>

**Figure 2-2. Table Interpretation of ModR/M Byte (C8H)**

Table 2-1. 16-Bit Addressing Forms with the ModR/M Byte

			AL AX EAX	CL CX ECX	DL DX EDX	BL BX EBX	AH SP ESP	CH BP <sup>1</sup> EBP	DH SI ESI	BH DI EDI
			MM0 XMM0	MM1 XMM1	MM2 XMM2	MM3 XMM3	MM4 XMM4	MM5 XMM5	MM6 XMM6	MM7 XMM7
			0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
Effective Address	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
[BX+SI] [BX+DI] [BP+SI] [BP+DI] [SI] [DI] disp16 <sup>2</sup> [BX]	00	000 001 010 011 100 101 110 111	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	10 11 12 13 14 15 16 17	18 19 1A 1B 1C 1D 1E 1F	20 21 22 23 24 25 26 27	28 29 2A 2B 2C 2D 2E 2F	30 31 32 33 34 35 36 37	38 39 3A 3B 3C 3D 3E 3F
[BX+SI]+disp8 <sup>3</sup> [BX+DI]+disp8 [BP+SI]+disp8 [BP+DI]+disp8 [SI]+disp8 [DI]+disp8 [BP]+disp8 [BX]+disp8	01	000 001 010 011 100 101 110 111	40 41 42 43 44 45 46 47	48 49 4A 4B 4C 4D 4E 4F	50 51 52 53 54 55 56 57	58 59 5A 5B 5C 5D 5E 5F	60 61 62 63 64 65 66 67	68 69 6A 6B 6C 6D 6E 6F	70 71 72 73 74 75 76 77	78 79 7A 7B 7C 7D 7E 7F
[BX+SI]+disp16 [BX+DI]+disp16 [BP+SI]+disp16 [BP+DI]+disp16 [SI]+disp16 [DI]+disp16 [BP]+disp16 [BX]+disp16	10	000 001 010 011 100 101 110 111	80 81 82 83 84 85 86 87	88 89 8A 8B 8C 8D 8E 8F	90 91 92 93 94 95 96 97	98 99 9A 9B 9C 9D 9E 9F	A0 A1 A2 A3 A4 A5 A6 A7	A8 A9 AA AB AC AD AE AF	B0 B1 B2 B3 B4 B5 B6 B7	B8 B9 BA BB BC BD BE BF
EAX/AX/AL/MM0/XMM0 ECX/CX/CL/MM1/XMM1 EDX/DX/DL/MM2/XMM2 EBX/BX/BL/MM3/XMM3 ESP/SP/AH/MM4/XMM4 EBP/BP/CH/MM5/XMM5 ESI/SI/DH/MM6/XMM6 EDI/DI/BH/MM7/XMM7	11	000 001 010 011 100 101 110 111	C0 C1 C2 C3 C4 C5 C6 C7	C8 C9 CA CB CC CD CE CF	D0 D1 D2 D3 D4 D5 D6 D7	D8 D9 DA DB DC DD DE DF	E0 E1 E2 E3 E4 E5 E6 E7	E8 E9 EA EB EC ED EE EF	F0 F1 F2 F3 F4 F5 F6 F7	F8 F9 FA FB FC FD FE FF

**NOTES:**

1. The default segment register is SS for the effective addresses containing a BP index, DS for other effective addresses.
2. The disp16 nomenclature denotes a 16-bit displacement that follows the ModR/M byte and that is added to the index.
3. The disp8 nomenclature denotes an 8-bit displacement that follows the ModR/M byte and that is sign-extended and added to the index.

Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG =	AL AX EAX	CL CX ECX	DL DX EDX	BL BX EBX	AH SP ESP	CH BP EBP	DH SI ESI	BH DI EDI		
	MM0 XMM0	MM1 XMM1	MM2 XMM2	MM3 XMM3	MM4 XMM4	MM5 XMM5	MM6 XMM6	MM7 XMM7		
	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111		
Effective Address	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
[EAX] [ECX] [EDX] [EBX] [--][--] <sup>1</sup> disp32 <sup>2</sup> [ESI] [EDI]	00	000 001 010 011 100 101 110 111	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	10 11 12 13 14 15 16 17	18 19 1A 1B 1C 1D 1E 1F	20 21 22 23 24 25 26 27	28 29 2A 2B 2C 2D 2E 2F	30 31 32 33 34 35 36 37	38 39 3A 3B 3C 3D 3E 3F
[EAX]+disp8 <sup>3</sup> [ECX]+disp8 [EDX]+disp8 [EBX]+disp8 [--][--]+disp8 [EBP]+disp8 [ESI]+disp8 [EDI]+disp8	01	000 001 010 011 100 101 110 111	40 41 42 43 44 45 46 47	48 49 4A 4B 4C 4D 4E 4F	50 51 52 53 54 55 56 57	58 59 5A 5B 5C 5D 5E 5F	60 61 62 63 64 65 66 67	68 69 6A 6B 6C 6D 6E 6F	70 71 72 73 74 75 76 77	78 79 7A 7B 7C 7D 7E 7F
[EAX]+disp32 [ECX]+disp32 [EDX]+disp32 [EBX]+disp32 [--][--]+disp32 [EBP]+disp32 [ESI]+disp32 [EDI]+disp32	10	000 001 010 011 100 101 110 111	80 81 82 83 84 85 86 87	88 89 8A 8B 8C 8D 8E 8F	90 91 92 93 94 95 96 97	98 99 9A 9B 9C 9D 9E 9F	A0 A1 A2 A3 A4 A5 A6 A7	A8 A9 AA AB AC AD AE AF	B0 B1 B2 B3 B4 B5 B6 B7	B8 B9 BA BB BC BD BE BF
EAX/AX/AL/MM0/XMM0 ECX/CX/CL/MM/XMM1 EDX/DX/DL/MM2/XMM2 EBX/BX/BL/MM3/XMM3 ESP/SP/AH/MM4/XMM4 EBP/BP/CH/MM5/XMM5 ESI/SI/DH/MM6/XMM6 EDI/DI/BH/MM7/XMM7	11	000 001 010 011 100 101 110 111	C0 C1 C2 C3 C4 C5 C6 C7	C8 C9 CA CB CC CD CE CF	D0 D1 D2 D3 D4 D5 D6 D7	D8 D9 DA DB DC DD DE DF	E0 E1 E2 E3 E4 E5 E6 E7	E8 E9 EA EB EC ED EE EF	F0 F1 F2 F3 F4 F5 F6 F7	F8 F9 FA FB FC FD FE FF

**NOTES:**

1. The [--][--] nomenclature means a SIB follows the ModR/M byte.
2. The disp32 nomenclature denotes a 32-bit displacement that follows the ModR/M byte (or the SIB byte if one is present) and that is added to the index.
3. The disp8 nomenclature denotes an 8-bit displacement that follows the ModR/M byte (or the SIB byte if one is present) and that is sign-extended and added to the index.

Table 2-3 is organized to give 256 possible values of the SIB byte (in hexadecimal). General purpose registers used as a base are indicated across the top of the table, along with corresponding values for the SIB byte’s base field. Table rows in the body of the table indicate the register used as the index (SIB byte bits 3, 4 and 5) and the scaling factor (determined by SIB byte bits 6 and 7).

Table 2-3. 32-Bit Addressing Forms with the SIB Byte

r32 (In decimal) Base = (In binary) Base =			EAX 0 000	ECX 1 001	EDX 2 010	EBX 3 011	ESP 4 100	[*] 5 101	ESI 6 110	EDI 7 111
Scaled Index	SS	Index	Value of SIB Byte (in Hexadecimal)							
[EAX] [ECX] [EDX] [EBX] none [EBP] [ESI] [EDI]	00	000 001 010 011 100 101 110 111	00 08 10 18 20 28 30 38	01 09 11 19 21 29 31 39	02 0A 12 1A 22 2A 32 3A	03 0B 13 1B 23 2B 33 3B	04 0C 14 1C 24 2C 34 3C	05 0D 15 1D 25 2D 35 3D	06 0E 16 1E 26 2E 36 3E	07 0F 17 1F 27 2F 37 3F
[EAX*2] [ECX*2] [EDX*2] [EBX*2] none [EBP*2] [ESI*2] [EDI*2]	01	000 001 010 011 100 101 110 111	40 48 50 58 60 68 70 78	41 49 51 59 61 69 71 79	42 4A 52 5A 62 6A 72 7A	43 4B 53 5B 63 6B 73 7B	44 4C 54 5C 64 6C 74 7C	45 4D 55 5D 65 6D 75 7D	46 4E 56 5E 66 6E 76 7E	47 4F 57 5F 67 6F 77 7F
[EAX*4] [ECX*4] [EDX*4] [EBX*4] none [EBP*4] [ESI*4] [EDI*4]	10	000 001 010 011 100 101 110 111	80 88 90 98 A0 A8 B0 B8	81 89 91 99 A1 A9 B1 B9	82 8A 92 9A A2 AA B2 BA	83 8B 93 9B A3 AB B3 BB	84 8C 94 9C A4 AC B4 BC	85 8D 95 9D A5 AD B5 BD	86 8E 96 9E A6 AE B6 BE	87 8F 97 9F A7 AF B7 BF
[EAX*8] [ECX*8] [EDX*8] [EBX*8] none [EBP*8] [ESI*8] [EDI*8]	11	000 001 010 011 100 101 110 111	C0 C8 D0 D8 E0 E8 F0 F8	C1 C9 D1 D9 E1 E9 F1 F9	C2 CA D2 DA E2 EA F2 FA	C3 CB D3 DB E3 EB F3 FB	C4 CC D4 DC E4 EC F4 FC	C5 CD D5 DD E5 ED F5 FD	C6 CE D6 DE E6 EE F6 FE	C7 CF D7 DF E7 EF F7 FF

**NOTES:**

- The [\*] nomenclature means a disp32 with no base if the MOD is 00B. Otherwise, [\*] means disp8 or disp32 + [EBP]. This provides the following address modes:

MOD bits    Effective Address

- |    |                                 |
|----|---------------------------------|
| 00 | [scaled index] + disp32         |
| 01 | [scaled index] + disp8 + [EBP]  |
| 10 | [scaled index] + disp32 + [EBP] |

## 2.2 IA-32E MODE

IA-32e mode has two sub-modes. These are:

- Compatibility Mode.** Enables a 64-bit operating system to run most legacy protected mode software unmodified.
- 64-Bit Mode.** Enables a 64-bit operating system to run applications written to access 64-bit address space.

## 2.2.1 REX Prefixes

REX prefixes are instruction-prefix bytes used in 64-bit mode. They do the following:

- Specify GPRs and SSE registers.
- Specify 64-bit operand size.
- Specify extended control registers.

Not all instructions require a REX prefix in 64-bit mode. A prefix is necessary only if an instruction references one of the extended registers or uses a 64-bit operand. If a REX prefix is used when it has no meaning, it is ignored.

Only one REX prefix is allowed per instruction. If used, the REX prefix byte must immediately precede the opcode byte or the escape opcode byte (0FH). When a REX prefix is used in conjunction with an instruction containing a mandatory prefix, the mandatory prefix must come before the REX so the REX prefix can be immediately preceding the opcode or the escape byte. For example, CVTDQ2PD with a REX prefix should have REX placed between F3 and 0F E6. Other placements are ignored. The instruction-size limit of 15 bytes still applies to instructions with a REX prefix. See Figure 2-3.

Legacy Prefixes	REX Prefix	Opcode	ModR/M	SIB	Displacement	Immediate
Grp 1, Grp 2, Grp 3, Grp 4 (optional)	(optional)	1-, 2-, or 3-byte opcode	1 byte (if required)	1 byte (if required)	Address displacement of 1, 2, or 4 bytes	Immediate data of 1, 2, or 4 bytes or none

Figure 2-3. Prefix Ordering in 64-bit Mode

### 2.2.1.1 Encoding

Intel 64 and IA-32 instruction formats specify up to three registers by using 3-bit fields in the encoding, depending on the format:

- ModR/M: the reg and r/m fields of the ModR/M byte.
- ModR/M with SIB: the reg field of the ModR/M byte, the base and index fields of the SIB (scale, index, base) byte.
- Instructions without ModR/M: the reg field of the opcode.

In 64-bit mode, these formats do not change. Bits needed to define fields in the 64-bit context are provided by the addition of REX prefixes.

### 2.2.1.2 More on REX Prefix Fields

REX prefixes are a set of 16 opcodes that span one row of the opcode map and occupy entries 40H to 4FH. These opcodes represent valid instructions (INC or DEC) in IA-32 operating modes and in compatibility mode. In 64-bit mode, the same opcodes represent the instruction prefix REX and are not treated as individual instructions.

The single-byte-opcode forms of the INC/DEC instructions are not available in 64-bit mode. INC/DEC functionality is still available using ModR/M forms of the same instructions (opcodes FF/0 and FF/1).

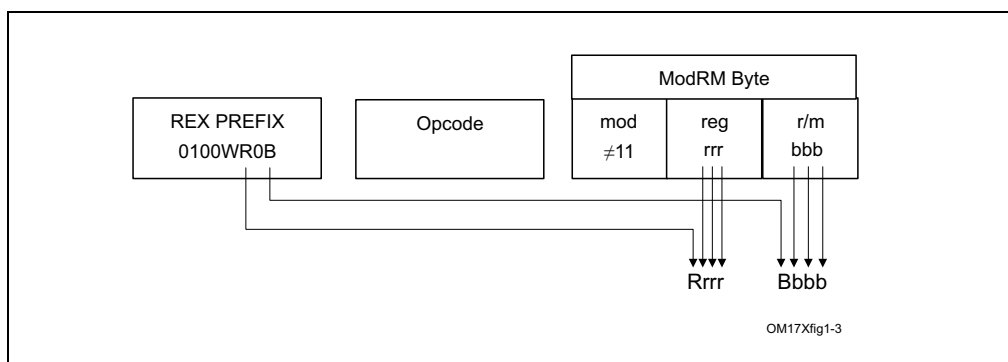
See Table 2-4 for a summary of the REX prefix format. Figure 2-4 through Figure 2-7 show examples of REX prefix fields in use. Some combinations of REX prefix fields are invalid. In such cases, the prefix is ignored. Some additional information follows:

- Setting REX.W can be used to determine the operand size but does not solely determine operand width. Like the 66H size prefix, 64-bit operand size override has no effect on byte-specific operations.
- For non-byte operations: if a 66H prefix is used with prefix (REX.W = 1), 66H is ignored.
- If a 66H override is used with REX and REX.W = 0, the operand size is 16 bits.

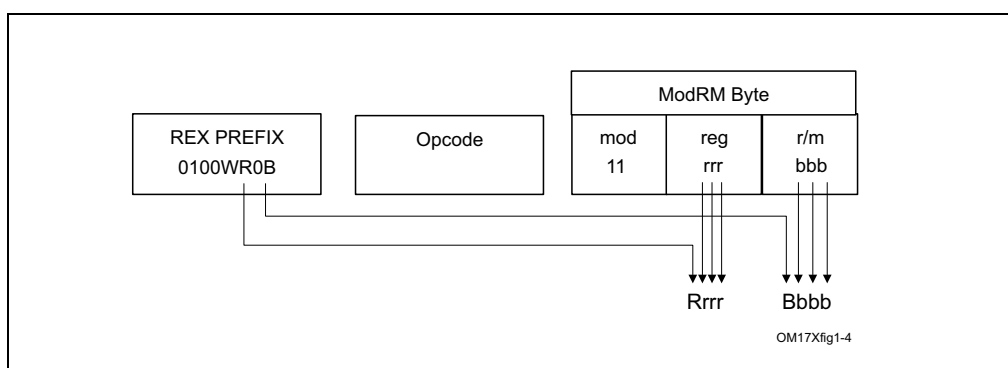
- REX.R modifies the ModR/M reg field when that field encodes a GPR, SSE, control or debug register. REX.R is ignored when ModR/M specifies other registers or defines an extended opcode.
- REX.X bit modifies the SIB index field.
- REX.B either modifies the base in the ModR/M r/m field or SIB base field; or it modifies the opcode reg field used for accessing GPRs.

**Table 2-4. REX Prefix Fields [BITS: 0100WRXB]**

Field Name	Bit Position	Definition
-	7:4	0100
W	3	0 = Operand size determined by CS.D 1 = 64 Bit Operand Size
R	2	Extension of the ModR/M reg field
X	1	Extension of the SIB index field
B	0	Extension of the ModR/M r/m field, SIB base field, or Opcode reg field



**Figure 2-4. Memory Addressing Without a SIB Byte; REX.X Not Used**



**Figure 2-5. Register-Register Addressing (No Memory Operand); REX.X Not Used**

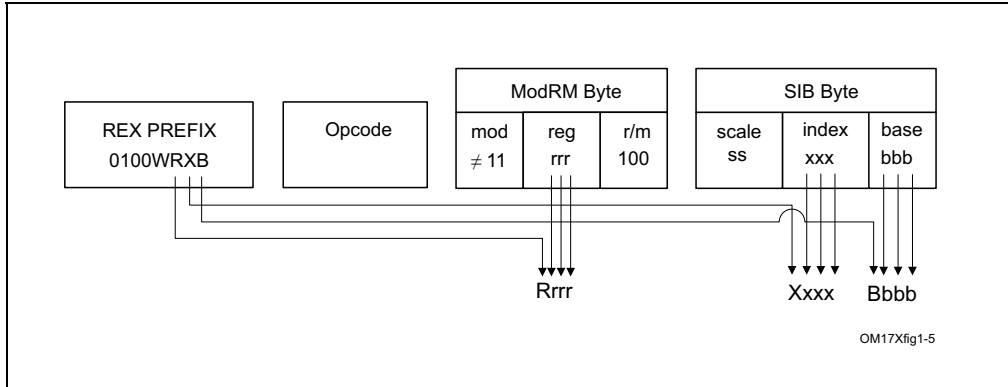


Figure 2-6. Memory Addressing With a SIB Byte

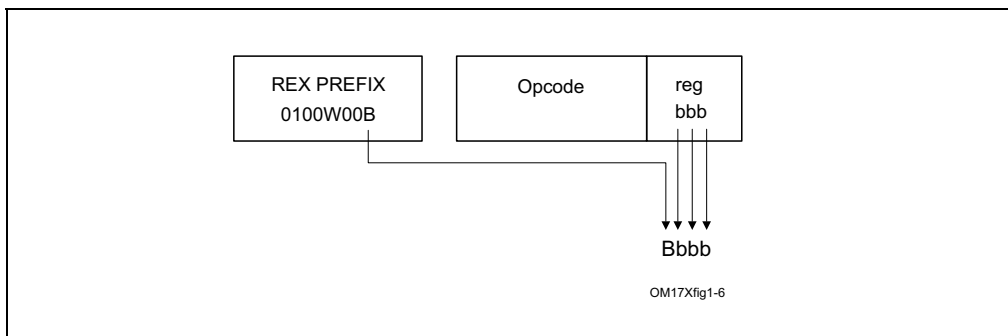


Figure 2-7. Register Operand Coded in Opcode Byte; REX.X & REX.R Not Used

In the IA-32 architecture, byte registers (AH, AL, BH, BL, CH, CL, DH, and DL) are encoded in the ModR/M byte’s reg field, the r/m field or the opcode reg field as registers 0 through 7. REX prefixes provide an additional addressing capability for byte-registers that makes the least-significant byte of GPRs available for byte operations. Certain combinations of the fields of the ModR/M byte and the SIB byte have special meaning for register encodings. For some combinations, fields expanded by the REX prefix are not decoded. Table 2-5 describes how each case behaves.



Table 2-5. Special Cases of REX Encodings

ModR/M or SIB	Sub-field Encodings	Compatibility Mode Operation	Compatibility Mode Implications	Additional Implications
ModR/M Byte	mod ≠ 11 r/m = b*100(ESP)	SIB byte present.	SIB byte required for ESP-based addressing.	REX prefix adds a fourth bit (b) which is not decoded (don't care). SIB byte also required for R12-based addressing.
ModR/M Byte	mod = 0 r/m = b*101(EBP)	Base register not used.	EBP without a displacement must be done using mod = 01 with displacement of 0.	REX prefix adds a fourth bit (b) which is not decoded (don't care). Using RBP or R13 without displacement must be done using mod = 01 with a displacement of 0.
SIB Byte	index = 0100(ESP)	Index register not used.	ESP cannot be used as an index register.	REX prefix adds a fourth bit (b) which is decoded. There are no additional implications. The expanded index field allows distinguishing RSP from R12, therefore R12 can be used as an index.
SIB Byte	base = 0101(EBP)	Base register is unused if mod = 0.	Base register depends on mod encoding.	REX prefix adds a fourth bit (b) which is not decoded. This requires explicit displacement to be used with EBP/RBP or R13.

**NOTES:**

\* Don't care about value of REX.B

### 2.2.1.3 Displacement

Addressing in 64-bit mode uses existing 32-bit ModR/M and SIB encodings. The ModR/M and SIB displacement sizes do not change. They remain 8 bits or 32 bits and are sign-extended to 64 bits.

### 2.2.1.4 Direct Memory-Offset MOVs

In 64-bit mode, direct memory-offset forms of the MOV instruction are extended to specify a 64-bit immediate absolute address. This address is called a moffset. No prefix is needed to specify this 64-bit memory offset. For these MOV instructions, the size of the memory offset follows the address-size default (64 bits in 64-bit mode). See Table 2-6.

Table 2-6. Direct Memory Offset Form of MOV

Opcode	Instruction
A0	MOV AL, moffset
A1	MOV EAX, moffset
A2	MOV moffset, AL
A3	MOV moffset, EAX

### 2.2.1.5 Immediates

In 64-bit mode, the typical size of immediate operands remains 32 bits. When the operand size is 64 bits, the processor sign-extends all immediates to 64 bits prior to their use.

Support for 64-bit immediate operands is accomplished by expanding the semantics of the existing move (MOV reg, imm16/32) instructions. These instructions (opcodes B8H – BFH) move 16-bits or 32-bits of immediate data (depending on the effective operand size) into a GPR. When the effective operand size is 64 bits, these instructions can be used to load an immediate into a GPR. A REX prefix is needed to override the 32-bit default operand size to a 64-bit operand size.

For example:

```
48 B8 8877665544332211 MOV RAX,1122334455667788H
```

### 2.2.1.6 RIP-Relative Addressing

A new addressing form, RIP-relative (relative instruction-pointer) addressing, is implemented in 64-bit mode. An effective address is formed by adding displacement to the 64-bit RIP of the next instruction.

In IA-32 architecture and compatibility mode, addressing relative to the instruction pointer is available only with control-transfer instructions. In 64-bit mode, instructions that use ModR/M addressing can use RIP-relative addressing. Without RIP-relative addressing, all ModR/M modes address memory relative to zero.

RIP-relative addressing allows specific ModR/M modes to address memory relative to the 64-bit RIP using a signed 32-bit displacement. This provides an offset range of  $\pm 2\text{GB}$  from the RIP. Table 2-7 shows the ModR/M and SIB encodings for RIP-relative addressing. Redundant forms of 32-bit displacement-addressing exist in the current ModR/M and SIB encodings. There is one ModR/M encoding and there are several SIB encodings. RIP-relative addressing is encoded using a redundant form.

In 64-bit mode, the ModR/M Disp32 (32-bit displacement) encoding is re-defined to be RIP+Disp32 rather than displacement-only. See Table 2-7.

**Table 2-7. RIP-Relative Addressing**

ModR/M and SIB Sub-field Encodings		Compatibility Mode Operation	64-bit Mode Operation	Additional Implications in 64-bit mode
ModR/M Byte	mod = 00	Disp32	RIP + Disp32	Must use SIB form with normal (zero-based) displacement addressing
	r/m = 101 (none)			
SIB Byte	base = 101 (none)	if mod = 00, Disp32	Same as legacy	None
	index = 100 (none)			
	scale = 0, 1, 2, 4			

The ModR/M encoding for RIP-relative addressing does not depend on using a prefix. Specifically, the r/m bit field encoding of 101B (used to select RIP-relative addressing) is not affected by the REX prefix. For example, selecting R13 (REX.B = 1, r/m = 101B) with mod = 00B still results in RIP-relative addressing. The 4-bit r/m field of REX.B combined with ModR/M is not fully decoded. In order to address R13 with no displacement, software must encode R13 + 0 using a 1-byte displacement of zero.

RIP-relative addressing is enabled by 64-bit mode, not by a 64-bit address-size. The use of the address-size prefix does not disable RIP-relative addressing. The effect of the address-size prefix is to truncate and zero-extend the computed effective address to 32 bits.

### 2.2.1.7 Default 64-Bit Operand Size

In 64-bit mode, two groups of instructions have a default operand size of 64 bits (do not need a REX prefix for this operand size). These are:

- Near branches.
- All instructions, except far branches, that implicitly reference the RSP.

## 2.2.2 Additional Encodings for Control and Debug Registers

In 64-bit mode, more encodings for control and debug registers are available. The REX.R bit is used to modify the ModR/M reg field when that field encodes a control or debug register (see Table 2-4). These encodings enable the processor to address CR8-CR15 and DR8-DR15. An additional control register (CR8) is defined in 64-bit mode. CR8 becomes the Task Priority Register (TPR).

In the first implementation of IA-32e mode, CR9-CR15 and DR8-DR15 are not implemented. Any attempt to access unimplemented registers results in an invalid-opcode exception (#UD).

## 2.3 INTEL® ADVANCED VECTOR EXTENSIONS (INTEL® AVX)

Intel AVX instructions are encoded using an encoding scheme that combines prefix bytes, opcode extension field, operand encoding fields, and vector length encoding capability into a new prefix, referred to as VEX. In the VEX encoding scheme, the VEX prefix may be two or three bytes long, depending on the instruction semantics. Despite the two-byte or three-byte length of the VEX prefix, the VEX encoding format provides a more compact representation/packing of the components of encoding an instruction in Intel 64 architecture. The VEX encoding scheme also allows more headroom for future growth of Intel 64 architecture.

### 2.3.1 Instruction Format

Instruction encoding using VEX prefix provides several advantages:

- Instruction syntax support for three operands and up-to four operands when necessary. For example, the third source register used by VBLENDVPD is encoded using bits 7:4 of the immediate byte.
- Encoding support for vector length of 128 bits (using XMM registers) and 256 bits (using YMM registers).
- Encoding support for instruction syntax of non-destructive source operands.
- Elimination of escape opcode byte (0FH), SIMD prefix byte (66H, F2H, F3H) via a compact bit field representation within the VEX prefix.
- Elimination of the need to use REX prefix to encode the extended half of general-purpose register sets (R8-R15) for direct register access, memory addressing, or accessing XMM8-XMM15 (including YMM8-YMM15).
- Flexible and more compact bit fields are provided in the VEX prefix to retain the full functionality provided by REX prefix. REX.W, REX.X, REX.B functionalities are provided in the three-byte VEX prefix only because only a subset of SIMD instructions need them.
- Extensibility for future instruction extensions without significant instruction length increase.

Figure 2-8 shows the Intel 64 instruction encoding format with VEX prefix support. Legacy instruction without a VEX prefix is fully supported and unchanged. The use of VEX prefix in an Intel 64 instruction is optional, but a VEX prefix is required for Intel 64 instructions that operate on YMM registers or support three and four operand syntax. VEX prefix is not a constant-valued, “single-purpose” byte like 0FH, 66H, F2H, F3H in legacy SSE instructions. VEX prefix provides substantially richer capability than the REX prefix.

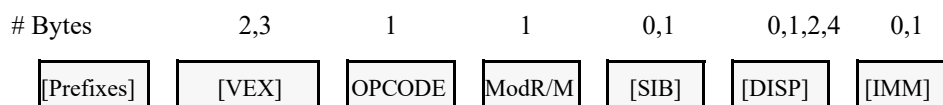


Figure 2-8. Instruction Encoding Format with VEX Prefix

### 2.3.2 VEX and the LOCK prefix

Any VEX-encoded instruction with a LOCK prefix preceding VEX will #UD.

### 2.3.3 VEX and the 66H, F2H, and F3H prefixes

Any VEX-encoded instruction with a 66H, F2H, or F3H prefix preceding VEX will #UD.

### 2.3.4 VEX and the REX prefix

Any VEX-encoded instruction with a REX prefix preceding VEX will #UD.

### 2.3.5 The VEX Prefix

The VEX prefix is encoded in either the two-byte form (the first byte must be C5H) or in the three-byte form (the first byte must be C4H). The two-byte VEX is used mainly for 128-bit, scalar, and the most common 256-bit AVX instructions; while the three-byte VEX provides a compact replacement of REX and 3-byte opcode instructions (including AVX and FMA instructions). Beyond the first byte of the VEX prefix, it consists of a number of bit fields providing specific capability, they are shown in Figure 2-9.

The bit fields of the VEX prefix can be summarized by its functional purposes:

- Non-destructive source register encoding (applicable to three and four operand syntax): This is the first source operand in the instruction syntax. It is represented by the notation, VEX.vvvv. This field is encoded using 1's complement form (inverted form), i.e. XMM0/YMM0/R0 is encoded as 1111B, XMM15/YMM15/R15 is encoded as 0000B.
- Vector length encoding: This 1-bit field represented by the notation VEX.L. L= 0 means vector length is 128 bits wide, L=1 means 256 bit vector. The value of this field is written as VEX.128 or VEX.256 in this document to distinguish encoded values of other VEX bit fields.
- REX prefix functionality: Full REX prefix functionality is provided in the three-byte form of VEX prefix. However the VEX bit fields providing REX functionality are encoded using 1's complement form, i.e. XMM0/YMM0/R0 is encoded as 1111B, XMM15/YMM15/R15 is encoded as 0000B.
  - Two-byte form of the VEX prefix only provides the equivalent functionality of REX.R, using 1's complement encoding. This is represented as VEX.R.
  - Three-byte form of the VEX prefix provides REX.R, REX.X, REX.B functionality using 1's complement encoding and three dedicated bit fields represented as VEX.R, VEX.X, VEX.B.
  - Three-byte form of the VEX prefix provides the functionality of REX.W only to specific instructions that need to override default 32-bit operand size for a general purpose register to 64-bit size in 64-bit mode. For those applicable instructions, VEX.W field provides the same functionality as REX.W. VEX.W field can provide completely different functionality for other instructions.

Consequently, the use of REX prefix with VEX encoded instructions is not allowed. However, the intent of the REX prefix for expanding register set is reserved for future instruction set extensions using VEX prefix encoding format.

- Compaction of SIMD prefix: Legacy SSE instructions effectively use SIMD prefixes (66H, F2H, F3H) as an opcode extension field. VEX prefix encoding allows the functional capability of such legacy SSE instructions (operating on XMM registers, bits 255:128 of corresponding YMM unmodified) to be encoded using the VEX.pp field without the presence of any SIMD prefix. The VEX-encoded 128-bit instruction will zero-out bits 255:128 of the destination register. VEX-encoded instruction may have 128 bit vector length or 256 bits length.
- Compaction of two-byte and three-byte opcode: More recently introduced legacy SSE instructions employ two and three-byte opcode. The one or two leading bytes are: 0FH, and 0FH 3AH/0FH 38H. The one-byte escape (0FH) and two-byte escape (0FH 3AH, 0FH 38H) can also be interpreted as an opcode extension field. The VEX.mmmmm field provides compaction to allow many legacy instruction to be encoded without the constant byte sequence, 0FH, 0FH 3AH, 0FH 38H. These VEX-encoded instruction may have 128 bit vector length or 256 bits length.

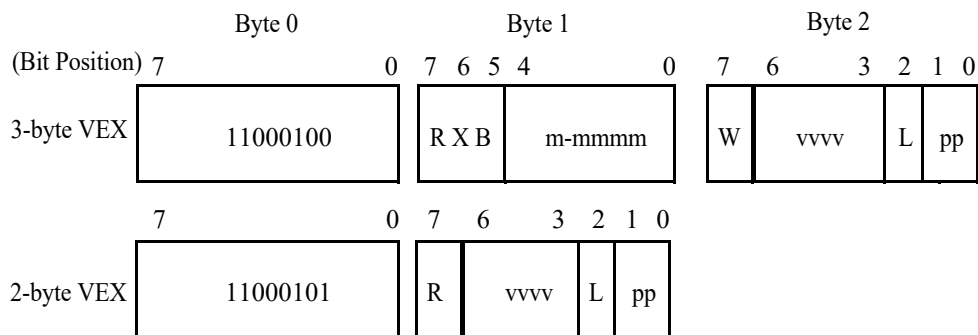
The VEX prefix is required to be the last prefix and immediately precedes the opcode bytes. It must follow any other prefixes. If VEX prefix is present a REX prefix is not supported.

The 3-byte VEX leaves room for future expansion with 3 reserved bits. REX and the 66h/F2h/F3h prefixes are reclaimed for future use.

VEX prefix has a two-byte form and a three byte form. If an instruction syntax can be encoded using the two-byte form, it can also be encoded using the three byte form of VEX. The latter increases the length of the instruction by one byte. This may be helpful in some situations for code alignment.

The VEX prefix supports 256-bit versions of floating-point SSE, SSE2, SSE3, and SSE4 instructions. Note, certain new instruction functionality can only be encoded with the VEX prefix.

The VEX prefix will #UD on any instruction containing MMX register sources or destinations.



R: REX.R in 1's complement (inverted) form  
 1: Same as REX.R=0 (must be 1 in 32-bit mode)  
 0: Same as REX.R=1 (64-bit mode only)

X: REX.X in 1's complement (inverted) form  
 1: Same as REX.X=0 (must be 1 in 32-bit mode)  
 0: Same as REX.X=1 (64-bit mode only)

B: REX.B in 1's complement (inverted) form  
 1: Same as REX.B=0 (Ignored in 32-bit mode).  
 0: Same as REX.B=1 (64-bit mode only)

W: opcode specific (use like REX.W, or used for opcode extension, or ignored, depending on the opcode byte)

m-mmmm:  
 00000: Reserved for future use (will #UD)  
 00001: implied 0F leading opcode byte  
 00010: implied 0F 38 leading opcode bytes  
 00011: implied 0F 3A leading opcode bytes  
 00100-11111: Reserved for future use (will #UD)

vvvv: a register specifier (in 1's complement form) or 1111 if unused.

L: Vector Length  
 0: scalar or 128-bit vector  
 1: 256-bit vector

pp: opcode extension providing equivalent functionality of a SIMD prefix  
 00: None  
 01: 66  
 10: F3  
 11: F2

**Figure 2-9. VEX bit fields**

The following subsections describe the various fields in two or three-byte VEX prefix.

### 2.3.5.1 VEX Byte 0, bits[7:0]

VEX Byte 0, bits [7:0] must contain the value 11000101b (C5h) or 11000100b (C4h). The 3-byte VEX uses the C4h first byte, while the 2-byte VEX uses the C5h first byte.

### 2.3.5.2 VEX Byte 1, bit [7] - 'R'

VEX Byte 1, bit [7] contains a bit analogous to a bit inverted REX.R. In protected and compatibility modes the bit must be set to '1' otherwise the instruction is LES or LDS.

This bit is present in both 2- and 3-byte VEX prefixes.

The usage of WRXB bits for legacy instructions is explained in detail section 2.2.1.2 of Intel 64 and IA-32 Architectures Software developer's manual, Volume 2A.

This bit is stored in bit inverted format.

### 2.3.5.3 3-byte VEX byte 1, bit[6] - 'X'

Bit[6] of the 3-byte VEX byte 1 encodes a bit analogous to a bit inverted REX.X. It is an extension of the SIB Index field in 64-bit modes. In 32-bit modes, this bit must be set to '1' otherwise the instruction is LES or LDS.

This bit is available only in the 3-byte VEX prefix.

This bit is stored in bit inverted format.

### 2.3.5.4 3-byte VEX byte 1, bit[5] - 'B'

Bit[5] of the 3-byte VEX byte 1 encodes a bit analogous to a bit inverted REX.B. In 64-bit modes, it is an extension of the ModR/M r/m field, or the SIB base field. In 32-bit modes, this bit is ignored.

This bit is available only in the 3-byte VEX prefix.

This bit is stored in bit inverted format.

### 2.3.5.5 3-byte VEX byte 2, bit[7] - 'W'

Bit[7] of the 3-byte VEX byte 2 is represented by the notation VEX.W. It can provide following functions, depending on the specific opcode.

- For AVX instructions that have equivalent legacy SSE instructions (typically these SSE instructions have a general-purpose register operand with its operand size attribute promotable by REX.W), if REX.W promotes the operand size attribute of the general-purpose register operand in legacy SSE instruction, VEX.W has same meaning in the corresponding AVX equivalent form. In 32-bit modes for these instructions, VEX.W is silently ignored.
- For AVX instructions that have equivalent legacy SSE instructions (typically these SSE instructions have operands with their operand size attribute fixed and not promotable by REX.W), if REX.W is don't care in legacy SSE instruction, VEX.W is ignored in the corresponding AVX equivalent form irrespective of mode.
- For new AVX instructions where VEX.W has no defined function (typically these meant the combination of the opcode byte and VEX.mmmmm did not have any equivalent SSE functions), VEX.W is reserved as zero and setting to other than zero will cause instruction to #UD.

### 2.3.5.6 2-byte VEX Byte 1, bits[6:3] and 3-byte VEX Byte 2, bits [6:3]- 'vvvv' the Source or Dest Register Specifier

In 32-bit mode the VEX first byte C4 and C5 alias onto the LES and LDS instructions. To maintain compatibility with existing programs the VEX 2nd byte, bits [7:6] must be 11b. To achieve this, the VEX payload bits are selected to place only inverted, 64-bit valid fields (extended register selectors) in these upper bits.

The 2-byte VEX Byte 1, bits [6:3] and the 3-byte VEX, Byte 2, bits [6:3] encode a field (shorthand VEX.vvvv) that for instructions with 2 or more source registers and an XMM or YMM or memory destination encodes the first source register specifier stored in inverted (1's complement) form.

VEX.vvvv is not used by the instructions with one source (except certain shifts, see below) or on instructions with no XMM or YMM or memory destination. If an instruction does not use VEX.vvvv then it should be set to 1111b otherwise instruction will #UD.

In 64-bit mode all 4 bits may be used. See Table 2-8 for the encoding of the XMM or YMM registers. In 32-bit and 16-bit modes bit 6 must be 1 (if bit 6 is not 1, the 2-byte VEX version will generate LDS instruction and the 3-byte VEX version will ignore this bit).

**Table 2-8. VEX.vvvv to register name mapping**

VEX.vvvv	Dest Register	Valid in Legacy/Compatibility 32-bit modes?
1111B	XMM0/YMM0	Valid
1110B	XMM1/YMM1	Valid
1101B	XMM2/YMM2	Valid
1100B	XMM3/YMM3	Valid
1011B	XMM4/YMM4	Valid
1010B	XMM5/YMM5	Valid
1001B	XMM6/YMM6	Valid
1000B	XMM7/YMM7	Valid
0111B	XMM8/YMM8	Invalid
0110B	XMM9/YMM9	Invalid
0101B	XMM10/YMM10	Invalid
0100B	XMM11/YMM11	Invalid
0011B	XMM12/YMM12	Invalid
0010B	XMM13/YMM13	Invalid
0001B	XMM14/YMM14	Invalid
0000B	XMM15/YMM15	Invalid

The VEX.vvvv field is encoded in bit inverted format for accessing a register operand.

### 2.3.6 Instruction Operand Encoding and VEX.vvvv, ModR/M

VEX-encoded instructions support three-operand and four-operand instruction syntax. Some VEX-encoded instructions have syntax with less than three operands, e.g. VEX-encoded pack shift instructions support one source operand and one destination operand).

The roles of VEX.vvvv, reg field of ModR/M byte (ModR/M.reg), r/m field of ModR/M byte (ModR/M.r/m) with respect to encoding destination and source operands vary with different type of instruction syntax.

The role of VEX.vvvv can be summarized to three situations:

- VEX.vvvv encodes the first source register operand, specified in inverted (1's complement) form and is valid for instructions with 2 or more source operands.
- VEX.vvvv encodes the destination register operand, specified in 1's complement form for certain vector shifts. The instructions where VEX.vvvv is used as a destination are listed in Table 2-9. The notation in the "Opcode" column in Table 2-9 is described in detail in section 3.1.1.
- VEX.vvvv does not encode any operand, the field is reserved and should contain 1111b.

**Table 2-9. Instructions with a VEX.vvvv destination**

Opcode	Instruction mnemonic
VEX.NDD.128.66.0F 73 /7 ib	VPSLLDQ xmm1, xmm2, imm8
VEX.NDD.128.66.0F 73 /3 ib	VPSRLDQ xmm1, xmm2, imm8
VEX.NDD.128.66.0F 71 /2 ib	VPSRLW xmm1, xmm2, imm8
VEX.NDD.128.66.0F 72 /2 ib	VPSRLD xmm1, xmm2, imm8
VEX.NDD.128.66.0F 73 /2 ib	VPSRLQ xmm1, xmm2, imm8
VEX.NDD.128.66.0F 71 /4 ib	VPSRAW xmm1, xmm2, imm8
VEX.NDD.128.66.0F 72 /4 ib	VPSRAD xmm1, xmm2, imm8
VEX.NDD.128.66.0F 71 /6 ib	VPSLLW xmm1, xmm2, imm8
VEX.NDD.128.66.0F 72 /6 ib	VPSLLD xmm1, xmm2, imm8
VEX.NDD.128.66.0F 73 /6 ib	VPSLLQ xmm1, xmm2, imm8

The role of ModR/M.r/m field can be summarized to two situations:

- ModR/M.r/m encodes the instruction operand that references a memory address.
- For some instructions that do not support memory addressing semantics, ModR/M.r/m encodes either the destination register operand or a source register operand.

The role of ModR/M.reg field can be summarized to two situations:

- ModR/M.reg encodes either the destination register operand or a source register operand.
- For some instructions, ModR/M.reg is treated as an opcode extension and not used to encode any instruction operand.

For instruction syntax that support four operands, VEX.vvvv, ModR/M.r/m, ModR/M.reg encodes three of the four operands. The role of bits 7:4 of the immediate byte serves the following situation:

- Imm8[7:4] encodes the third source register operand.

### 2.3.6.1 3-byte VEX byte 1, bits[4:0] - “m-mmmm”

Bits[4:0] of the 3-byte VEX byte 1 encode an implied leading opcode byte (0F, 0F 38, or 0F 3A). Several bits are reserved for future use and will #UD unless 0.

**Table 2-10. VEX.m-mmmm interpretation**

VEX.m-mmmm	Implied Leading Opcode Bytes
00000B	Reserved
00001B	0F
00010B	0F 38
00011B	0F 3A
00100-11111B	Reserved
(2-byte VEX)	0F

VEX.m-mmmm is only available on the 3-byte VEX. The 2-byte VEX implies a leading 0Fh opcode byte.

### 2.3.6.2 2-byte VEX byte 1, bit[2], and 3-byte VEX byte 2, bit [2]- “L”

The vector length field, VEX.L, is encoded in bit[2] of either the second byte of 2-byte VEX, or the third byte of 3-byte VEX. If “VEX.L = 1”, it indicates 256-bit vector operation. “VEX.L = 0” indicates scalar and 128-bit vector operations.

The instruction VZEROUPPER is a special case that is encoded with VEX.L = 0, although its operation zero’s bits 255:128 of all YMM registers accessible in the current operating mode.

See the following table.

**Table 2-11. VEX.L interpretation**

VEX.L	Vector Length
0	128-bit (or 32/64-bit scalar)
1	256-bit

### 2.3.6.3 2-byte VEX byte 1, bits[1:0], and 3-byte VEX byte 2, bits [1:0]- “pp”

Up to one implied prefix is encoded by bits[1:0] of either the 2-byte VEX byte 1 or the 3-byte VEX byte 2. The prefix behaves as if it was encoded prior to VEX, but after all other encoded prefixes.

See the following table.



Table 2-12. VEX.pp interpretation

pp	Implies this prefix after other prefixes but before VEX
00B	None
01B	66
10B	F3
11B	F2

### 2.3.7 The Opcode Byte

One (and only one) opcode byte follows the 2 or 3 byte VEX. Legal opcodes are specified in Appendix B, in color. Any instruction that uses illegal opcode will #UD.

### 2.3.8 The MODRM, SIB, and Displacement Bytes

The encodings are unchanged but the interpretation of reg\_field or rm\_field differs (see above).

### 2.3.9 The Third Source Operand (Immediate Byte)

VEX-encoded instructions can support instruction with a four operand syntax. VBLENDVPD, VBLENDVPS, and PBLENDVB use imm8[7:4] to encode one of the source registers.

### 2.3.10 AVX Instructions and the Upper 128-bits of YMM registers

If an instruction with a destination XMM register is encoded with a VEX prefix, the processor zeroes the upper bits (above bit 128) of the equivalent YMM register. Legacy SSE instructions without VEX preserve the upper bits.

#### 2.3.10.1 Vector Length Transition and Programming Considerations

An instruction encoded with a VEX.128 prefix that loads a YMM register operand operates as follows:

- Data is loaded into bits 127:0 of the register
- Bits above bit 127 in the register are cleared.

Thus, such an instruction clears bits 255:128 of a destination YMM register on processors with a maximum vector-register width of 256 bits. In the event that future processors extend the vector registers to greater widths, an instruction encoded with a VEX.128 or VEX.256 prefix will also clear any bits beyond bit 255. (This is in contrast with legacy SSE instructions, which have no VEX prefix; these modify only bits 127:0 of any destination register operand.)

Programmers should bear in mind that instructions encoded with VEX.128 and VEX.256 prefixes will clear any future extensions to the vector registers. A calling function that uses such extensions should save their state before calling legacy functions. This is not possible for involuntary calls (e.g., into an interrupt-service routine). It is recommended that software handling involuntary calls accommodate this by not executing instructions encoded with VEX.128 and VEX.256 prefixes. In the event that it is not possible or desirable to restrict these instructions, then software must take special care to avoid actions that would, on future processors, zero the upper bits of vector registers.

Processors that support further vector-register extensions (defining bits beyond bit 255) will also extend the XSAVE and XRSTOR instructions to save and restore these extensions. To ensure forward compatibility, software that handles involuntary calls and that uses instructions encoded with VEX.128 and VEX.256 prefixes should first save and then restore the vector registers (with any extensions) using the XSAVE and XRSTOR instructions with save/restore masks that set bits that correspond to all vector-register extensions. Ideally, software should rely on a mechanism that is cognizant of which bits to set. (E.g., an OS mechanism that sets the save/restore mask bits for all vector-register extensions that are enabled in XCR0.) Saving and restoring state with instructions other than XSAVE and XRSTOR will, on future processors with wider vector registers, corrupt the extended state of the vector registers - even if doing so functions correctly on processors supporting 256-bit vector registers. (The same is true

if XSAVE and XRSTOR are used with a save/restore mask that does not set bits corresponding to all supported extensions to the vector registers.)

### 2.3.11 AVX Instruction Length

The AVX instructions described in this document (including VEX and ignoring other prefixes) do not exceed 11 bytes in length, but may increase in the future. The maximum length of an Intel 64 and IA-32 instruction remains 15 bytes.

### 2.3.12 Vector SIB (VSIB) Memory Addressing

In Intel® Advanced Vector Extensions 2 (Intel® AVX2), an SIB byte that follows the ModR/M byte can support VSIB memory addressing to an array of linear addresses. VSIB addressing is only supported in a subset of Intel AVX2 instructions. VSIB memory addressing requires 32-bit or 64-bit effective address. In 32-bit mode, VSIB addressing is not supported when address size attribute is overridden to 16 bits. In 16-bit protected mode, VSIB memory addressing is permitted if address size attribute is overridden to 32 bits. Additionally, VSIB memory addressing is supported only with VEX prefix.

In VSIB memory addressing, the SIB byte consists of:

- The scale field (bit 7:6) specifies the scale factor.
- The index field (bits 5:3) specifies the register number of the vector index register, each element in the vector register specifies an index.
- The base field (bits 2:0) specifies the register number of the base register.

Table 2-3 shows the 32-bit VSIB addressing form. It is organized to give 256 possible values of the SIB byte (in hexadecimal). General purpose registers used as a base are indicated across the top of the table, along with corresponding values for the SIB byte's base field. The register names also include R8L-R15L applicable only in 64-bit mode (when address size override prefix is used, but the value of VEX.B is not shown in Table 2-3). In 32-bit mode, R8L-R15L does not apply.

Table rows in the body of the table indicate the vector index register used as the index field and each supported scaling factor shown separately. Vector registers used in the index field can be XMM or YMM registers. The left-most column includes vector registers VR8-VR15 (i.e. XMM8/YMM8-XMM15/YMM15), which are only available in 64-bit mode and does not apply if encoding in 32-bit mode.

**Table 2-13. 32-Bit VSIB Addressing Forms of the SIB Byte**

r32			EAX/ R8L	ECX/ R9L	EDX/ R10L	EBX/ R11L	ESP/ R12L	EBP/ R13L <sup>1</sup>	ESI/ R14L	EDI/ R15L	
(In decimal) Base =			0	1	2	3	4	5	6	7	
(In binary) Base =			000	001	010	011	100	101	110	111	
Scaled Index		SS	Index	Value of SIB Byte (in Hexadecimal)							
VR0/VR8	*1	00	000	00	01	02	03	04	05	06	07
VR1/VR9			001	08	09	0A	0B	0C	0D	0E	0F
VR2/VR10			010	10	11	12	13	14	15	16	17
VR3/VR11			011	18	19	1A	1B	1C	1D	1E	1F
VR4/VR12			100	20	21	22	23	24	25	26	27
VR5/VR13			101	28	29	2A	2B	2C	2D	2E	2F
VR6/VR14			110	30	31	32	33	34	35	36	37
VR7/VR15			111	38	39	3A	3B	3C	3D	3E	3F
VR0/VR8	*2	01	000	40	41	42	43	44	45	46	47
VR1/VR9			001	48	49	4A	4B	4C	4D	4E	4F
VR2/VR10			010	50	51	52	53	54	55	56	57
VR3/VR11			011	58	59	5A	5B	5C	5D	5E	5F
VR4/VR12			100	60	61	62	63	64	65	66	67
VR5/VR13			101	68	69	6A	6B	6C	6D	6E	6F
VR6/VR14			110	70	71	72	73	74	75	76	77
VR7/VR15			111	78	79	7A	7B	7C	7D	7E	7F

Table 2-13. 32-Bit VSIB Addressing Forms of the SIB Byte (Contd.)

VR0/VR8	*4	10	000	80	81	82	83	84	85	86	87
VR1/VR9			001	88	89	8A	8B	8C	8D	8E	8F
VR2/VR10			010	90	91	92	93	94	95	96	97
VR3/VR11			011	98	99	9A	9B	9C	9D	9E	9F
VR4/VR12			100	A0	A1	A2	A3	A4	A5	A6	A7
VR5/VR13			101	A8	A9	AA	AB	AC	AD	AE	AF
VR6/VR14			110	B0	B1	B2	B3	B4	B5	B6	B7
VR7/VR15			111	B8	B9	BA	BB	BC	BD	BE	BF
VR0/VR8	*8	11	000	C0	C1	C2	C3	C4	C5	C6	C7
VR1/VR9			001	C8	C9	CA	CB	CC	CD	CE	CF
VR2/VR10			010	D0	D1	D2	D3	D4	D5	D6	D7
VR3/VR11			011	D8	D9	DA	DB	DC	DD	DE	DF
VR4/VR12			100	E0	E1	E2	E3	E4	E5	E6	E7
VR5/VR13			101	E8	E9	EA	EB	EC	ED	EE	EF
VR6/VR14			110	F0	F1	F2	F3	F4	F5	F6	F7
VR7/VR15			111	F8	F9	FA	FB	FC	FD	FE	FF

**NOTES:**

1. If ModR/M.mod = 00b, the base address is zero, then effective address is computed as [scaled vector index] + disp32. Otherwise the base address is computed as [EBP/R13]+ disp, the displacement is either 8 bit or 32 bit depending on the value of ModR/M.mod:

MOD	Effective Address
00b	[Scaled Vector Register] + Disp32
01b	[Scaled Vector Register] + Disp8 + [EBP/R13]
10b	[Scaled Vector Register] + Disp32 + [EBP/R13]

**2.3.12.1 64-bit Mode VSIB Memory Addressing**

In 64-bit mode VSIB memory addressing uses the VEX.B field and the base field of the SIB byte to encode one of the 16 general-purpose register as the base register. The VEX.X field and the index field of the SIB byte encode one of the 16 vector registers as the vector index register.

In 64-bit mode the top row of Table 2-13 base register should be interpreted as the full 64-bit of each register.

**2.4 AVX AND SSE INSTRUCTION EXCEPTION SPECIFICATION**

To look up the exceptions of legacy 128-bit SIMD instruction, 128-bit VEX-encoded instructions, and 256-bit VEX-encoded instruction, Table 2-14 summarizes the exception behavior into separate classes, with detailed exception conditions defined in sub-sections 2.4.1 through 2.5.1. For example, ADDPS contains the entry:

“See Exceptions Type 2”

In this entry, “Type2” can be looked up in Table 2-14.

The instruction’s corresponding CPUID feature flag can be identified in the fourth column of the Instruction summary table.

Note: #UD on CPUID feature flags=0 is not guaranteed in a virtualized environment if the hardware supports the feature flag.

**NOTE**

Instructions that operate only with MMX, X87, or general-purpose registers are not covered by the exception classes defined in this section. For instructions that operate on MMX registers, see Section 22.25.3, “Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B*.

**Table 2-14. Exception class description**

Exception Class	Instruction set	Mem arg	Floating-Point Exceptions (#XM)
Type 1	AVX, Legacy SSE	16/32 byte explicitly aligned	None
Type 2	AVX, Legacy SSE	16/32 byte not explicitly aligned	Yes
Type 3	AVX, Legacy SSE	< 16 byte	Yes
Type 4	AVX, Legacy SSE	16/32 byte not explicitly aligned	No
Type 5	AVX, Legacy SSE	< 16 byte	No
Type 6	AVX (no Legacy SSE)	Varies	(At present, none do)
Type 7	AVX, Legacy SSE	None	None
Type 8	AVX	None	None
Type 11	F16C	8 or 16 byte, Not explicitly aligned, no AC#	Yes
Type 12	AVX2	Not explicitly aligned, no AC#	No

See Table 2-15 for lists of instructions in each exception class.

Table 2-15. Instructions in each Exception Class

Exception Class	Instruction
Type 1	(V)MOVAPD, (V)MOVAPS, (V)MOVDDQA, (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)CVTPS2DQ, (V)CVTTPD2DQ, (V)CVTTPS2DQ, (V)DIVPD, (V)DIVPS, (V)DPPD*, (V)DPPS*, (V)FMADD132PD, (V)FMADD213PD, (V)FMADD231PD, (V)FMADD132PS, (V)FMADD213PS, (V)FMADD231PS, (V)FMADDSUB132PD, (V)FMADDSUB213PD, (V)FMADDSUB231PD, (V)FMADDSUB132PS, (V)FMADDSUB213PS, (V)FMADDSUB231PS, (V)FMSUBADD132PD, (V)FMSUBADD213PD, (V)FMSUBADD231PD, (V)FMSUBADD132PS, (V)FMSUBADD213PS, (V)FMSUBADD231PS, (V)FMSUB132PD, (V)FMSUB213PD, (V)FMSUB231PD, (V)FMSUB132PS, (V)FMSUB213PS, (V)FMSUB231PS, (V)FNMADD132PD, (V)FNMADD213PD, (V)FNMADD231PD, (V)FNMADD132PS, (V)FNMADD213PS, (V)FNMADD231PS, (V)FNMMSUB132PD, (V)FNMMSUB213PD, (V)FNMMSUB231PD, (V)FNMMSUB132PS, (V)FNMMSUB213PS, (V)FNMMSUB231PS, (V)HADDPD, (V)HADDPs, (V)HSUBPD, (V)HSUBPS, (V)MAXPD, (V)MAXPS, (V)MINPD, (V)MINPS, (V)MULPD, (V)MULPS, (V)ROUNDPS, (V)SQRTPD, (V)SQRTPS, (V)SUBPD, (V)SUBPS
Type 3	(V)ADDS, (V)ADDSs, (V)CMPD, (V)CMPs, (V)COMISD, (V)COMISS, (V)CVTSD2PS, (V)CVTSD2SI, (V)CVTSD2SS, (V)CVTSD2SD, (V)CVTSD2SS, (V)CVTSS2SD, (V)CVTSS2SI, (V)CVTSS2SI, (V)CVTSS2SI, (V)DIVSD, (V)DIVSS, (V)FMADD132SD, (V)FMADD213SD, (V)FMADD231SD, (V)FMADD132SS, (V)FMADD213SS, (V)FMADD231SS, (V)FMSUB132SD, (V)FMSUB213SD, (V)FMSUB231SD, (V)FMSUB132SS, (V)FMSUB213SS, (V)FMSUB231SS, (V)FNMADD132SD, (V)FNMADD213SD, (V)FNMADD231SD, (V)FNMADD132SS, (V)FNMADD213SS, (V)FNMADD231SS, (V)FNMMSUB132SD, (V)FNMMSUB213SD, (V)FNMMSUB231SD, (V)FNMMSUB132SS, (V)FNMMSUB213SS, (V)FNMMSUB231SS, (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)MASKMOVDQU, (V)PTEST, (V)TESTPS, (V)TESTPD, (V)MOVDQU*, (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)PADDUSB, (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)PMADDUBSW, (V)PMASXB, (V)PMASXW, (V)PMASXD, (V)PMASXUB, (V)PMASXUW, (V)PMASXUD, (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)PSHUFW, (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, (V)BLEND, (V)PERMD, (V)PERMPS, (V)PERMPD, (V)PERMQ, (V)PSLLVD, (V)PSLLVQ, (V)PSRAVD, (V)PSRLVD, (V)PSRLVQ, (V)PERMILPD, (V)PERMILPS, (V)PERM2F128
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)RCPPS, (V)RSQRTSS, (V)PMOVSX/ZX, (V)LDMXCSR*, (V)STMXCSR
Type 6	VEXTRACTF128, VBROADCASTSS, VBROADCASTSD, VBROADCASTF128, VINSERTF128, VMASKMOVPS**, (V)MASKMOVPD**, (V)MASKMOVQ, (V)MASKMOVQ, VBROADCASTI128, VPBROADCASTB, VPBROADCASTD, VPBROADCASTW, VPBROADCASTQ, VEXTRACTI128, VINSERTI128, VPERM2I128
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
Type 11	VCVTPH2PS, VCVTPS2PH
Type 12	VGATHERDPS, VGATHERDPD, VGATHERQPS, VGATHERQPD, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

(\*) - 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, PCMPISTRI, PCMPISTRM and LDDQU instructions do not cause #GP if the memory operand is not aligned to 16-Byte boundary.

Table 2-15 classifies exception behaviors for AVX instructions. Within each class of exception conditions that are listed in Table 2-18 through Table 2-27, certain subsets of AVX instructions may be subject to #UD exception depending on the encoded value of the VEX.L field. Table 2-17 provides supplemental information of AVX instructions that may be subject to #UD exception if encoded with incorrect values in the VEX.W or VEX.L field.

**Table 2-16. #UD Exception and VEX.W=1 Encoding**

Exception Class	#UD If VEX.W = 1 in all modes	#UD If VEX.W = 1 in non-64-bit modes
Type 1		
Type 2		
Type 3		
Type 4	VBLENDVPD, VBLENDVPS, VPBLENDVB, VTESTPD, VTESTPS, VPBLEND, VPERMD, VPERMPS, VPERM2I128, VPSRAVD, VPERMILPD, VPERMILPS, VPERM2F128	
Type 5		
Type 6	VEXTRACTF128, VBROADCASTSS, VBROADCASTSD, VBROADCASTF128, VINSERTF128, VMASKMOVPS, VMASKMOVPD, VBROADCASTI128, VPBROADCASTB/W/D, VEXTRACTI128, VINSERTI128	
Type 7		
Type 8		
Type 11	VCVTPH2PS, VCVTPS2PH	
Type 12		

Table 2-17. #UD Exception and VEX.L Field Encoding

Exception Class	#UD If VEX.L = 0	#UD If (VEX.L = 1 && AVX2 not present && AVX present)	#UD If (VEX.L = 1 && AVX2 present)
Type 1		VMOVNTDQA	
Type 2		VDPPD	VDPPD
Type 3			
Type 4		VMASKMOVDQU, VMPSADBW, VPABSB/W/D, VPACKSSWB/DW, VPACKUSWB/DW, VPADDB/W/D, VPADDQ, VPADDSB/W, VPADDUSB/W, VPALIGNR, VPAND, VPANDN, VPAVGB/W, VPBLENDVB, VPBLENDW, VPCMP(E/I)STRI/M, VPCMPEQB/W/D/Q, VPCMPGTB/W/D/Q, VPHADDW/D, VPHADDSW, VPHMINPOSUW, VPHSUBD/W, VPHSUBSW, VPMADDWD, VPMADDUBSW, VPMAXSB/W/D, VPMAXUB/W/D, VPMINSB/W/D, VPMINUB/W/D, VPMULHUW, VPMULHRW, VPMULHW/LW, VPMULLD, VPMULLDQ, VPMULDQ, VPOR, VPSADBW, VPSHUF/D, VPSHUFHW/LW, VPSIGNB/W/D, VPSLLW/D/Q, VPSRAW/D, VPSRLW/D/Q, VPSUBB/W/D/Q, VPSUBSB/W, VPUNPCKHBW/W/D/DQ, VPUNPCKHQDQ, VPUNPCKLBW/W/D/DQ, VPUNPCKLQDQ, VPXOR	VPCMP(E/I)STRI/M, PHMINPOSUW
Type 5		VEXTRACTPS, VINSERTPS, VMOVD, VMOVQ, VMOVLPD, VMOVLPS, VMOVHPD, VMOVHPS, VPEXTRB, VPEXTRD, VPEXTRW, VPEXTRQ, VPINSRB, VPINSRD, VPINSRW, VPINSRQ, VPMOVSX/ZX, VLDMXCSR, VSTMXCSR	Same as column 3
Type 6	VEXTRACTF128, VPERM2F128, VBROADCASTSD, VBROADCASTF128, VINSERTF128,		
Type 7		VMOVLHPS, VMOVHLPS, VPMOVMASKB, VPSLLDQ, VPSRLDQ, VPSLLW, VPSLLD, VPSLLQ, VPSRAW, VPSRAD, VPSRLW, VPSRLD, VPSRLQ	VMOVLHPS, VMOVHLPS
Type 8			
Type 11			
Type 12			

## 2.4.1 Exceptions Type 1 (Aligned memory reference)

Table 2-18. Type 1 Class Exception Conditions

Exception	Real	Virtual-8086	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 CR0.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 CR0.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	VEX.256: Memory operand is not 32-byte aligned. VEX.128: Memory operand is not 16-byte aligned.
	X	X	X	X	Legacy SSE: Memory operand is not 16-byte aligned.
			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.



## 2.4.2 Exceptions Type 2 (>=16 Byte Memory Reference, Unaligned)

Table 2-19. Type 2 Class Exception Conditions

Exception	Real	Virtual 8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			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 (F0H).
			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.
			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.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1.

## 2.4.3 Exceptions Type 3 (<16 Byte memory argument)

Table 2-20. Type 3 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	VEX prefix: If XCRO[2:1] ≠ '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CR0.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 CR0.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		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.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 Bytes or less is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1.

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

Table 2-21. Type 4 Class Exception Conditions

Exception	Real	Virtual-8086	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. <sup>1</sup>
			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, PCMPISTRI, PCMPISTRM and LDDQU instructions do not cause #GP if the memory operand is not aligned to 16-Byte boundary.

## 2.4.5 Exceptions Type 5 (<16 Byte mem arg and no FP exceptions)

Table 2-22. Type 5 Class Exception Conditions

Exception	Real	Virtual-8086	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 (F0H).
			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		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.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

## 2.4.6 Exceptions Type 6 (VEX-Encoded Instructions Without Legacy SSE Analogues)

Note: At present, the AVX instructions in this category do not generate floating-point exceptions.

**Table 2-23. Type 6 Class Exception Conditions**

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	If XCRO[2:1] ≠ '11b'. If CR4.OSXSAVE[bit 18]=0.
			X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
			X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.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		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.
Page Fault #PF(fault-code)			X	X	For a page fault.
Alignment Check #AC(0)			X	X	For 4 or 8 byte memory references if alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

## 2.4.7 Exceptions Type 7 (No FP exceptions, no memory arg)

Table 2-24. Type 7 Class Exception Conditions

Exception	Real	Virtual-8086	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 CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			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	If CR0.TS[bit 3]=1.

## 2.4.8 Exceptions Type 8 (AVX and no memory argument)

Table 2-25. Type 8 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			Always in Real or Virtual-8086 mode.
			X	X	If XCRO[2:1] ≠ '11b'. If CR4.OSXSAVE[bit 18]=0. If CPUID.01H.ECX.AVX[bit 28]=0. If VEX.vvvv ≠ 1111B.
	X	X	X	X	If proceeded by a LOCK prefix (F0H).
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.

## 2.4.9 Exception Type 11 (VEX-only, mem arg no AC, floating-point exceptions)

Table 2-26. Type 11 Class Exception Conditions

Exception	Real	Virtual-8086	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	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 CR0.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		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.
SIMD Floating-Point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1.

## 2.4.10 Exception Type 12 (VEX-only, VSIB mem arg, no AC, no floating-point exceptions)

Table 2-27. Type 12 Class Exception Conditions

Exception	Real	Virtual-8086	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	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	NA	If address size attribute is 16 bit.
	X	X	X	X	If ModR/M.mod = '11b'.
	X	X	X	X	If ModR/M.rm ≠ '100b'.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
			X		For an illegal address in the SS segment.
Stack, SS(0)				X	If a memory address referencing the SS segment is in a non-canonical form.
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
General Protection, #GP(0)				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.

## 2.5 VEX ENCODING SUPPORT FOR GPR INSTRUCTIONS

VEX prefix may be used to encode instructions that operate on neither YMM nor XMM registers. VEX-encoded general-purpose-register instructions have the following properties:

- Instruction syntax support for three encodable operands.
- Encoding support for instruction syntax of non-destructive source operand, destination operand encoded via VEX.vvvv, and destructive three-operand syntax.
- Elimination of escape opcode byte (0FH), two-byte escape via a compact bit field representation within the VEX prefix.
- Elimination of the need to use REX prefix to encode the extended half of general-purpose register sets (R8-R15) for direct register access or memory addressing.
- Flexible and more compact bit fields are provided in the VEX prefix to retain the full functionality provided by REX prefix. REX.W, REX.X, REX.B functionalities are provided in the three-byte VEX prefix only.
- VEX-encoded GPR instructions are encoded with VEX.L=0.



Any VEX-encoded GPR instruction with a 66H, F2H, or F3H prefix preceding VEX will #UD.

Any VEX-encoded GPR instruction with a REX prefix proceeding VEX will #UD.

VEX-encoded GPR instructions are not supported in real and virtual 8086 modes.

## 2.5.1 Exception Conditions for VEX-Encoded GPR Instructions

The exception conditions applicable to VEX-encoded GPR instruction differs from those of legacy GPR instructions. Table 2-28 lists VEX-encoded GPR instructions. The exception conditions for VEX-encoded GPR instructions are found in Table 2-29 for those instructions which have a default operand size of 32 bits and 16-bit operand size is not encodable.

**Table 2-28. VEX-Encoded GPR Instructions**

Exception Class	Instruction
See Table 2-29	ANDN, BLSI, BLSMSK, BLSR, BZHI, MULX, PDEP, PEXT, RORX, SARX, SHLX, SHRX

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

**Table 2-29. Exception Definition (VEX-Encoded GPR Instructions)**

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If BMI1/BMI2 CPUID feature flag is '0'.
	X	X			If a VEX prefix is present.
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
Stack, SS(0)	X	X	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		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.
				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.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

## 2.6 INTEL® AVX-512 ENCODING

The majority of the Intel AVX-512 family of instructions (operating on 512/256/128-bit vector register operands) are encoded using a new prefix (called EVEX). Opmask instructions (operating on opmask register operands) are encoded using the VEX prefix. The EVEX prefix has some parts resembling the instruction encoding scheme using the VEX prefix, and many other capabilities not available with the VEX prefix.

The significant feature differences between EVEX and VEX are summarized below.

- EVEX is a 4-Byte prefix (the first byte must be 62H); VEX is either a 2-Byte (C5H is the first byte) or 3-Byte (C4H is the first byte) prefix.
- EVEX prefix can encode 32 vector registers (XMM/YMM/ZMM) in 64-bit mode.
- EVEX prefix can encode an opmask register for conditional processing or selection control in EVEX-encoded vector instructions. Opmask instructions, whose source/destination operands are opmask registers and treat the content of an opmask register as a single value, are encoded using the VEX prefix.
- EVEX memory addressing with disp8 form uses a compressed disp8 encoding scheme to improve the encoding density of the instruction byte stream.
- EVEX prefix can encode functionality that are specific to instruction classes (e.g., packed instruction with "load+op" semantic can support embedded broadcast functionality, floating-point instruction with rounding semantic can support static rounding functionality, floating-point instruction with non-rounding arithmetic semantic can support "suppress all exceptions" functionality).

### 2.6.1 Instruction Format and EVEX

The placement of the EVEX prefix in an IA instruction is represented in Figure 2-10.

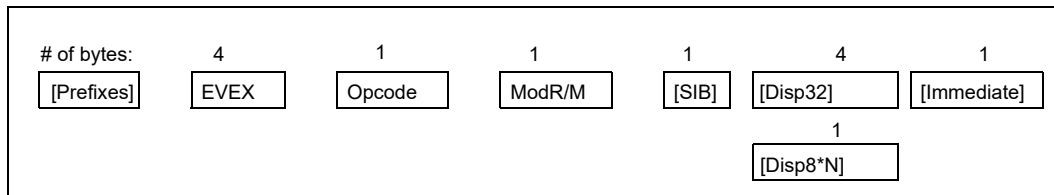


Figure 2-10. AVX-512 Instruction Format and the EVEX Prefix

The EVEX prefix is a 4-byte prefix, with the first two bytes derived from unused encoding form of the 32-bit-mode-only BOUND instruction. The layout of the EVEX prefix is shown in Figure 2-11. The first byte must be 62H, followed by three payload bytes, denoted as P0, P1, and P2 individually or collectively as P[23:0] (see Figure 2-11).

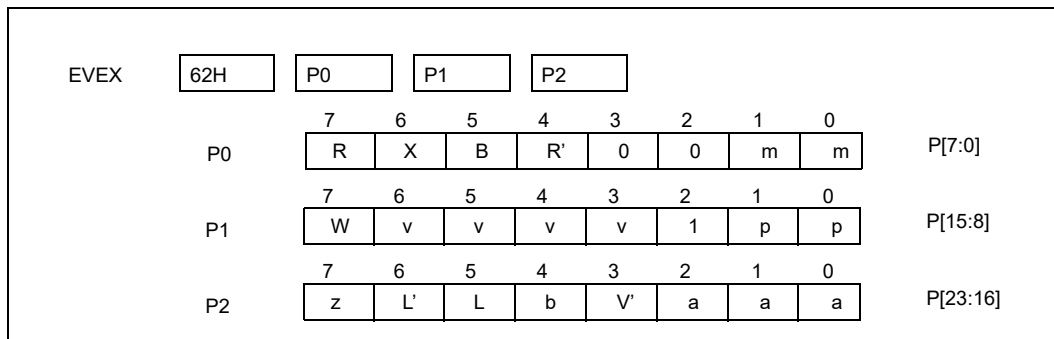


Figure 2-11. Bit Field Layout of the EVEX Prefix

Table 2-30. EVEX Prefix Bit Field Functional Grouping

Notation	Bit field Group	Position	Comment
--	Reserved	P[3 : 2]	Must be 0.
--	Fixed Value	P[10]	Must be 1.
EVEX.mm	Compressed legacy escape	P[1: 0]	Identical to low two bits of VEX.mmmmm.
EVEX.pp	Compressed legacy prefix	P[9 : 8]	Identical to VEX.pp.
EVEX.RXB	Next-8 register specifier modifier	P[7 : 5]	Combine with ModR/M.reg, ModR/M.rm (base, index/vidx).
EVEX.R'	High-16 register specifier modifier	P[4]	Combine with EVEX.R and ModR/M.reg.
EVEX.X	High-16 register specifier modifier	P[6]	Combine with EVEX.B and ModR/M.rm, when SIB/VSIB absent.
EVEX.vvvv	NDS register specifier	P[14 : 11]	Same as VEX.vvvv.
EVEX.V'	High-16 NDS/VIDX register specifier	P[19]	Combine with EVEX.vvvv or when VSIB present.
EVEX.aaa	Embedded opmask register specifier	P[18 : 16]	
EVEX.W	Osize promotion/Opcode extension	P[15]	
EVEX.z	Zeroing/Merging	P[23]	
EVEX.b	Broadcast/RC/SAE Context	P[20]	
EVEX.L'L	Vector length/RC	P[22 : 21]	

The bit fields in P[23:0] are divided into the following functional groups (Table 2-30 provides a tabular summary):

- Reserved bits: P[3:2] must be 0, otherwise #UD.
- Fixed-value bit: P[10] must be 1, otherwise #UD.
- Compressed legacy prefix/escape bytes: P[1:0] is identical to the lowest 2 bits of VEX.mmmmm; P[9:8] is identical to VEX.pp.
- Operand specifier modifier bits for vector register, general purpose register, memory addressing: P[7:5] allows access to the next set of 8 registers beyond the low 8 registers when combined with ModR/M register specifiers.
- Operand specifier modifier bit for vector register: P[4] (or EVEX.R') allows access to the high 16 vector register set when combined with P[7] and ModR/M.reg specifier; P[6] can also provide access to a high 16 vector register when SIB or VSIB addressing are not needed.
- Non-destructive source /vector index operand specifier: P[19] and P[14:11] encode the second source vector register operand in a non-destructive source syntax, vector index register operand can access an upper 16 vector register using P[19].
- Op-mask register specifiers: P[18:16] encodes op-mask register set k0-k7 in instructions operating on vector registers.
- EVEX.W: P[15] is similar to VEX.W which serves either as opcode extension bit or operand size promotion to 64-bit in 64-bit mode.
- Vector destination merging/zeroing: P[23] encodes the destination result behavior which either zeroes the masked elements or leave masked element unchanged.
- Broadcast/Static-rounding/SAE context bit: P[20] encodes multiple functionality, which differs across different classes of instructions and can affect the meaning of the remaining field (EVEX.L'L). The functionality for the following instruction classes are:
  - Broadcasting a single element across the destination vector register: this applies to the instruction class with Load+Op semantic where one of the source operand is from memory.
  - Redirect L'L field (P[22:21]) as static rounding control for floating-point instructions with rounding semantic. Static rounding control overrides MXCSR.RC field and implies "Suppress all exceptions" (SAE).
  - Enable SAE for floating -point instructions with arithmetic semantic that is not rounding.
  - For instruction classes outside of the afore-mentioned three classes, setting EVEX.b will cause #UD.

- Vector length/rounding control specifier: P[22:21] can serve one of three options.
  - Vector length information for packed vector instructions.
  - Ignored for instructions operating on vector register content as a single data element.
  - Rounding control for floating-point instructions that have a rounding semantic and whose source and destination operands are all vector registers.

## 2.6.2 Register Specifier Encoding and EVEX

EVEX-encoded instruction can access 8 opmask registers, 16 general-purpose registers and 32 vector registers in 64-bit mode (8 general-purpose registers and 8 vector registers in non-64-bit modes). EVEX-encoding can support instruction syntax that access up to 4 instruction operands. Normal memory addressing modes and VSIB memory addressing are supported with EVEX prefix encoding. The mapping of register operands used by various instruction syntax and memory addressing in 64-bit mode are shown in Table 2-31. Opmask register encoding is described in Section 2.6.3.

**Table 2-31. 32-Register Support in 64-bit Mode Using EVEX with Embedded REX Bits**

	4 <sup>1</sup>	3	[2:0]	Reg. Type	Common Usages
<b>REG</b>	EVEX.R'	REX.R	modrm.reg	GPR, Vector	Destination or Source
<b>NDS/NDD</b>	EVEX.V'	EVEX.vvvv		GPR, Vector	2ndSource or Destination
<b>RM</b>	EVEX.X	EVEX.B	modrm.r/m	GPR, Vector	1st Source or Destination
<b>BASE</b>	0	EVEX.B	modrm.r/m	GPR	memory addressing
<b>INDEX</b>	0	EVEX.X	sib.index	GPR	memory addressing
<b>VIDX</b>	EVEX.V'	EVEX.X	sib.index	Vector	VSIB memory addressing

### NOTES:

1. Not applicable for accessing general purpose registers.

The mapping of register operands used by various instruction syntax and memory addressing in 32-bit modes are shown in Table 2-32.

**Table 2-32. EVEX Encoding Register Specifiers in 32-bit Mode**

	[2:0]	Reg. Type	Common Usages
<b>REG</b>	modrm.reg	GPR, Vector	Destination or Source
<b>NDS/NDD</b>	EVEX.vvv	GPR, Vector	2nd Source or Destination
<b>RM</b>	modrm.r/m	GPR, Vector	1st Source or Destination
<b>BASE</b>	modrm.r/m	GPR	Memory Addressing
<b>INDEX</b>	sib.index	GPR	Memory Addressing
<b>VIDX</b>	sib.index	Vector	VSIB Memory Addressing

## 2.6.3 Opmask Register Encoding

There are eight opmask registers, k0-k7. Opmask register encoding falls into two categories:

- Opmask registers that are the source or destination operands of an instruction treating the content of opmask register as a scalar value, are encoded using the VEX prefix scheme. It can support up to three operands using standard modR/M byte's reg field and rm field and VEX.vvvv. Such a scalar opmask instruction does not support conditional update of the destination operand.
- An opmask register providing conditional processing and/or conditional update of the destination register of a vector instruction is encoded using EVEX.aaa field (see Section 2.6.4).

- An opmask register serving as the destination or source operand of a vector instruction is encoded using standard modR/M byte's reg field and rm fields.

Table 2-33. Opmask Register Specifier Encoding

	[2:0]	Register Access	Common Usages
REG	modrm.reg	k0-k7	Source
NDS	VEX.vvvv	k0-k7	2nd Source
RM	modrm.r/m	k0-7	1st Source
{k1}	EVEX.aaa	k0 <sup>1</sup> -k7	Opmask

**NOTES:**

- Instructions that overwrite the conditional mask in opmask do not permit using k0 as the embedded mask.

## 2.6.4 Masking Support in EVEX

EVEX can encode an opmask register to conditionally control per-element computational operation and updating of result of an instruction to the destination operand. The predicate operand is known as the opmask register. The EVEX.aaa field, P[18:16] of the EVEX prefix, is used to encode one out of a set of eight 64-bit architectural registers. Note that from this set of 8 architectural registers, only k1 through k7 can be addressed as predicate operands. k0 can be used as a regular source or destination but cannot be encoded as a predicate operand.

AVX-512 instructions support two types of masking with EVEX.z bit (P[23]) controlling the type of masking:

- Merging-masking, which is the default type of masking for EVEX-encoded vector instructions, preserves the old value of each element of the destination where the corresponding mask bit has a 0. It corresponds to the case of EVEX.z = 0.
- Zeroing-masking, is enabled by having the EVEX.z bit set to 1. In this case, an element of the destination is set to 0 when the corresponding mask bit has a 0 value.

AVX-512 Foundation instructions can be divided into the following groups:

- Instructions which support “zeroing-masking”.
  - Also allow merging-masking.
- Instructions which require aaa = 000.
  - Do not allow any form of masking.
- Instructions which allow merging-masking but do not allow zeroing-masking.
  - Require EVEX.z to be set to 0.
  - This group is mostly composed of instructions that write to memory.
- Instructions which require aaa <> 000 do not allow EVEX.z to be set to 1.
  - Allow merging-masking and do not allow zeroing-masking, e.g., gather instructions.

## 2.6.5 Compressed Displacement (disp8\*N) Support in EVEX

For memory addressing using disp8 form, EVEX-encoded instructions always use a compressed displacement scheme by multiplying disp8 in conjunction with a scaling factor N that is determined based on the vector length, the value of EVEX.b bit (embedded broadcast) and the input element size of the instruction. In general, the factor N corresponds to the number of bytes characterizing the internal memory operation of the input operand (e.g., 64 when the accessing a full 512-bit memory vector). The scale factor N is listed in Table 2-34 and Table 2-35 below, where EVEX encoded instructions are classified using the **tupletype** attribute. The scale factor N of each tupletype is listed based on the vector length (VL) and other factors affecting it.

Table 2-34 covers EVEX-encoded instructions which has a load semantic in conjunction with additional computational or data element movement operation, operating either on the full vector or half vector (due to conversion of

numerical precision from a wider format to narrower format). EVEX.b is supported for such instructions for data element sizes which are either dword or qword (see Section 2.6.11).

EVEX-encoded instruction that are pure load/store, and "Load+op" instruction semantic that operate on data element size less than dword do not support broadcasting using EVEX.b. These are listed in Table 2-35. Table 2-35 also includes many broadcast instructions which perform broadcast using a subset of data elements without using EVEX.b. These instructions and a few data element size conversion instructions are covered in Table 2-35. Instruction classified in Table 2-35 do not use EVEX.b and EVEX.b must be 0, otherwise #UD will occur.

The tuple type will be referenced in the instruction operand encoding table in the reference page of each instruction, providing the cross reference for the scaling factor N to encoding memory addressing operand.

Note that the disp8\*N rules still apply when using 16b addressing.

**Table 2-34. Compressed Displacement (DISP8\*N) Affected by Embedded Broadcast**

TupleType	EVEX.b	InputSize	EVEX.W	Broadcast	N (VL=128)	N (VL=256)	N (VL= 512)	Comment
Full	0	32bit	0	none	16	32	64	Load+Op (Full Vector Dword/Qword)
	1	32bit	0	{1tox}	4	4	4	
	0	64bit	1	none	16	32	64	
	1	64bit	1	{1tox}	8	8	8	
Half	0	32bit	0	none	8	16	32	Load+Op (Half Vector)
	1	32bit	0	{1tox}	4	4	4	

**Table 2-35. EVEX DISP8\*N for Instructions Not Affected by Embedded Broadcast**

TupleType	InputSize	EVEX.W	N (VL= 128)	N (VL= 256)	N (VL= 512)	Comment
Full Mem	N/A	N/A	16	32	64	Load/store or subDword full vector
Tuple1 Scalar	8bit	N/A	1	1	1	1 Tuple
	16bit	N/A	2	2	2	
	32bit	0	4	4	4	
	64bit	1	8	8	8	
Tuple1 Fixed	32bit	N/A	4	4	4	1 Tuple, memsize not affected by EVEX.W
	64bit	N/A	8	8	8	
Tuple2	32bit	0	8	8	8	Broadcast (2 elements)
	64bit	1	NA	16	16	
Tuple4	32bit	0	NA	16	16	Broadcast (4 elements)
	64bit	1	NA	NA	32	
Tuple8	32bit	0	NA	NA	32	Broadcast (8 elements)
Half Mem	N/A	N/A	8	16	32	SubQword Conversion
Quarter Mem	N/A	N/A	4	8	16	SubDword Conversion
Eighth Mem	N/A	N/A	2	4	8	SubWord Conversion
Mem128	N/A	N/A	16	16	16	Shift count from memory
MOVDDUP	N/A	N/A	8	32	64	VMOVDDUP

## 2.6.6 EVEX Encoding of Broadcast/Rounding/SAE Support

EVEX.b can provide three types of encoding context, depending on the instruction classes:

- Embedded broadcasting of one data element from a source memory operand to the destination for vector instructions with “load+op” semantic.
- Static rounding control overriding MXCSR.RC for floating-point instructions with rounding semantic.
- “Suppress All exceptions” (SAE) overriding MXCSR mask control for floating-point arithmetic instructions that do not have rounding semantic.

## 2.6.7 Embedded Broadcast Support in EVEX

EVEX encodes an embedded broadcast functionality that is supported on many vector instructions with 32-bit (double word or single-precision floating-point) and 64-bit data elements, and when the source operand is from memory. EVEX.b (P[20]) bit is used to enable broadcast on load-op instructions. When enabled, only one element is loaded from memory and broadcasted to all other elements instead of loading the full memory size.

The following instruction classes do not support embedded broadcasting:

- Instructions with only one scalar result is written to the vector destination.
- Instructions with explicit broadcast functionality provided by its opcode.
- Instruction semantic is a pure load or a pure store operation.

## 2.6.8 Static Rounding Support in EVEX

Static rounding control embedded in the EVEX encoding system applies only to register-to-register flavor of floating-point instructions with rounding semantic at two distinct vector lengths: (i) scalar, (ii) 512-bit. In both cases, the field EVEX.L'L expresses rounding mode control overriding MXCSR.RC if EVEX.b is set. When EVEX.b is set, “suppress all exceptions” is implied. The processor behaves as if all MXCSR masking controls are set.

## 2.6.9 SAE Support in EVEX

The EVEX encoding system allows arithmetic floating-point instructions without rounding semantic to be encoded with the SAE attribute. This capability applies to scalar and 512-bit vector lengths, register-to-register only, by setting EVEX.b. When EVEX.b is set, “suppress all exceptions” is implied. The processor behaves as if all MXCSR masking controls are set.

## 2.6.10 Vector Length Orthogonality

The architecture of EVEX encoding scheme can support SIMD instructions operating at multiple vector lengths. Many AVX-512 Foundation instructions operate at 512-bit vector length. The vector length of EVEX encoded vector instructions are generally determined using the L'L field in EVEX prefix, except for 512-bit floating-point, reg-reg instructions with rounding semantic. The table below shows the vector length corresponding to various values of the L'L bits. When EVEX is used to encode scalar instructions, L'L is generally ignored.

When EVEX.b bit is set for a register-register instructions with floating-point rounding semantic, the same two bits P2[6:5] specifies rounding mode for the instruction, with implied SAE behavior. The mapping of different instruction classes relative to the embedded broadcast/rounding/SAE control and the EVEX.L'L fields are summarized in Table 2-36.

**Table 2-36. EVEX Embedded Broadcast/Rounding/SAE and Vector Length on Vector Instructions**

Position	P2[4]	P2[6:5]	P2[6:5]
Broadcast/Rounding/SAE Context	EVEX.b	EVEX.L'L	EVEX.RC
Reg-reg, FP Instructions w/ rounding semantic	Enable static rounding control (SAE implied)	Vector length Implied (512 bit or scalar)	00b: SAE + RNE 01b: SAE + RD 10b: SAE + RU 11b: SAE + RZ
FP Instructions w/o rounding semantic, can cause #XF	SAE control	00b: 128-bit 01b: 256-bit 10b: 512-bit 11b: Reserved (#UD)	NA
Load+op Instructions w/ memory source	Broadcast Control		NA
Other Instructions ( Explicit Load/Store/Broadcast/Gather/Scatter)	Must be 0 (otherwise #UD)		NA

## 2.6.11 #UD Equations for EVEX

Instructions encoded using EVEX can face three types of UD conditions: state dependent, opcode independent and opcode dependent.

### 2.6.11.1 State Dependent #UD

In general, attempts to execute an instruction, which required OS support for incremental extended state component, will #UD if required state components were not enabled by OS. Table 2-37 lists instruction categories with respect to required processor state components. Attempts to execute a given category of instructions while enabled states were less than the required bit vector in XCR0 shown in Table 2-37 will cause #UD.

**Table 2-37. OS XSAVE Enabling Requirements of Instruction Categories**

Instruction Categories	Vector Register State Access	Required XCR0 Bit Vector [7:0]
Legacy SIMD prefix encoded Instructions (e.g SSE)	XMM	xxxxxx11b
VEX-encoded instructions operating on YMM	YMM	xxxxx111b
EVEX-encoded 128-bit instructions	ZMM	111xx111b
EVEX-encoded 256-bit instructions	ZMM	111xx111b
EVEX-encoded 512-bit instructions	ZMM	111xx111b
VEX-encoded instructions operating on opmask	k-reg	xx1xxx11b

### 2.6.11.2 Opcode Independent #UD

A number of bit fields in EVEX encoded instruction must obey mode-specific but opcode-independent patterns listed in Table 2-38.

**Table 2-38. Opcode Independent, State Dependent EVEX Bit Fields**

Position	Notation	64-bit #UD	Non-64-bit #UD
P[3 : 2]	--	if > 0	if > 0
P[10]	--	if 0	if 0
P[1 : 0]	EVEX.mm	if 00b	if 00b
P[7 : 6]	EVEX.RX	None (valid)	None (BOUND if EVEX.RX != 11b)



### 2.6.11.3 Opcode Dependent #UD

This section describes legal values for the rest of the EVEX bit fields. Table 2-39 lists the #UD conditions of EVEX prefix bit fields which encodes or modifies register operands.

**Table 2-39. #UD Conditions of Operand-Encoding EVEX Prefix Bit Fields**

Notation	Position	Operand Encoding	64-bit #UD	Non-64-bit #UD
EVEX.R	P[7]	ModRM.reg encodes k-reg	if EVEX.R = 0	None (BOUND if EVEX.RX != 11b)
		ModRM.reg is opcode extension	None (ignored)	
		ModRM.reg encodes all other registers	None (valid)	
EVEX.X	P[6]	ModRM.r/m encodes ZMM/YMM/XMM	None (valid)	
		ModRM.r/m encodes k-reg or GPR	None (ignored)	
		ModRM.r/m without SIB/VSIB	None (ignored)	
		ModRM.r/m with SIB/VSIB	None (valid)	
EVEX.B	P[5]	ModRM.r/m encodes k-reg	None (ignored)	None (ignored)
		ModRM.r/m encodes other registers	None (valid)	
		ModRM.r/m base present	None (valid)	
		ModRM.r/m base not present	None (ignored)	
EVEXR'	P[4]	ModRM.reg encodes k-reg or GPR	if 0	None (ignored)
		ModRM.reg is opcode extension	None (ignored)	
		ModRM.reg encodes ZMM/YMM/XMM	None (valid)	
EVEX.vvvv	P[14 : 11]	vvvv encodes ZMM/YMM/XMM	None (valid)	None (valid) P[14] ignored
		Otherwise	if != 1111b	if != 1111b
EVEXV'	P[19]	Encodes ZMM/YMM/XMM	None (valid)	None (ignored)
		Otherwise	if 0	None (ignored)

Table 2-40 lists the #UD conditions of instruction encoding of opmask register using EVEX.aaa and EVEX.z

**Table 2-40. #UD Conditions of Opmask Related Encoding Field**

Notation	Position	Operand Encoding	64-bit #UD	Non-64-bit #UD
EVEX.aaa	P[18 : 16]	Instructions do not use opmask for conditional processing <sup>1</sup> .	if aaa != 000b	if aaa != 000b
		Opmask used as conditional processing mask and updated at completion <sup>2</sup> .	if aaa = 000b	if aaa = 000b;
		Opmask used as conditional processing.	None (valid <sup>3</sup> )	None (valid <sup>1</sup> )
EVEX.z	P[23]	Vector instruction using opmask as source or destination <sup>4</sup> .	if EVEX.z != 0	if EVEX.z != 0
		Store instructions or gather/scatter instructions.	if EVEX.z != 0	if EVEX.z != 0
		Instruction supporting conditional processing mask with EVEX.aaa = 000b.	if EVEX.z != 0	if EVEX.z != 0
VEX.vvvv	Varies	K-reg are instruction operands not mask control.	if vvvv = 0xxx	None

#### NOTES:

1. E.g., VBROADCASTMxxx, VPMOVM2x, VPMOVx2M.

2. E.g., Gather/Scatter family.

3. aaa can take any value. A value of 000 indicates that there is no masking on the instruction; in this case, all elements will be processed as if there was a mask of 'all ones' regardless of the actual value in KO.

4. E.g., VFPCLASSPD/PS, VCMPPB/D/Q/W family, VPMOVM2x, VPMOVx2M.

Table 2-41 lists the #UD conditions of EVEX bit fields that depends on the context of EVEX.b.

**Table 2-41. #UD Conditions Dependent on EVEX.b Context**

Notation	Position	Operand Encoding	64-bit #UD	Non-64-bit #UD
EVEX.L'Lb	P[22 : 20]	Reg-reg, FP instructions with rounding semantic.	None (valid <sup>1</sup> )	None (valid <sup>1</sup> )
		Other reg-reg, FP instructions that can cause #XF.	None (valid <sup>2</sup> )	None (valid <sup>2</sup> )
		Other reg-mem instructions in Table 2-34.	None (valid <sup>3</sup> )	None (valid <sup>3</sup> )
		Other instruction classes <sup>4</sup> in Table 2-35.	If EVEX.b > 0	If EVEX.b > 0

**NOTES:**

1. L'L specifies rounding control, see Table 2-36, supports {er} syntax.
2. L'L specifies vector length, see Table 2-36, supports {sae} syntax.
3. L'L specifies vector length, see Table 2-36, supports embedded broadcast syntax
4. L'L specifies either vector length or ignored.

## 2.6.12 Device Not Available

EVEX-encoded instructions follow the same rules when it comes to generating #NM (Device Not Available) exception. In particular, it is generated when CR0.TS[bit 3]= 1.

## 2.6.13 Scalar Instructions

EVEX-encoded scalar SIMD instructions can access up to 32 registers in 64-bit mode. Scalar instructions support masking (using the least significant bit of the opmask register), but broadcasting is not supported.

## 2.7 EXCEPTION CLASSIFICATIONS OF EVEX-ENCODED INSTRUCTIONS

The exception behavior of EVEX-encoded instructions can be classified into the classes shown in the rest of this section. The classification of EVEX-encoded instructions follow a similar framework as those of AVX and AVX2 instructions using the VEX prefix. Exception types for EVEX-encoded instructions are named in the style of "E##" or with a suffix "E##XX". The "##" designation generally follows that of AVX/AVX2 instructions. The majority of EVEX encoded instruction with "Load+op" semantic supports memory fault suppression, which is represented by E##. The instructions with "Load+op" semantic but do not support fault suppression are named "E##NF". A summary table of exception classes by class names are shown below.

**Table 2-42. EVEX-Encoded Instruction Exception Class Summary**

Exception Class	Instruction set	Mem arg	(#XM)
Type E1	Vector Moves/Load/Stores	Explicitly aligned, w/ fault suppression	None
Type E1NF	Vector Non-temporal Stores	Explicitly aligned, no fault suppression	None
Type E2	FP Vector Load+op	Support fault suppression	Yes
Type E2NF	FP Vector Load+op	No fault suppression	Yes
Type E3	FP Scalar/Partial Vector, Load+Op	Support fault suppression	Yes
Type E3NF	FP Scalar/Partial Vector, Load+Op	No fault suppression	Yes
Type E4	Integer Vector Load+op	Support fault suppression	No
Type E4NF	Integer Vector Load+op	No fault suppression	No
Type E5	Legacy-like Promotion	Varies, Support fault suppression	No

**Table 2-42. EVEX-Encoded Instruction Exception Class Summary**

Exception Class	Instruction set	Mem arg	(#XM)
Type E5NF	Legacy-like Promotion	Varies, No fault suppression	No
Type E6	Post AVX Promotion	Varies, w/ fault suppression	No
Type E6NF	Post AVX Promotion	Varies, no fault suppression	No
Type E7NM	Register-to-register op	None	None
Type E9NF	Miscellaneous 128-bit	Vector-length Specific, no fault suppression	None
Type E10	Non-XF Scalar	Vector Length ignored, w/ fault suppression	None
Type E10NF	Non-XF Scalar	Vector Length ignored, no fault suppression	None
Type E11	VCVTPH2PS	Half Vector Length, w/ fault suppression	Yes
Type E11NF	VCVTPS2PH	Half Vector Length, no fault suppression	Yes
Type E12	Gather and Scatter Family	VSIB addressing, w/ fault suppression	None
Type E12NP	Gather and Scatter Prefetch Family	VSIB addressing, w/o page fault	None

Table 2-43 lists EVEX-encoded instruction mnemonic by exception classes.

**Table 2-43. EVEX Instructions in each Exception Class**

Exception Class	Instruction
Type E1	VMOVAPD, VMOVAPS, VMOVDQA32, VMOVDQA64
Type E1NF	VMOVNTDQ, VMOVNTDQA, VMOVNTPD, VMOVNTPS
Type E2	VADDPD, VADDPDS, VCMPPD, VCMPPS, VCVTDQ2PS, VCVTPD2DQ, VCVTPD2PS, VCVTPS2DQ, VCVTTPD2DQ, VCVTTPS2DQ, VDIVPD, VDIVPS, VFMADDxxxPD, VFMADDxxxPS, VFMSUBADDxxxPD, VFMSUBADDxxxPS, VFMSUBxxxPD, VFMSUBxxxPS, VFNMADDxxxPD, VFNMADDxxxPS, VFNMSUBxxxPD, VFNMSUBxxxPS, VMAXPD, VMAXPS, VMINPD, VMINPS, VMULPD, VMULPS, VSQRTPD, VSQRTPS, VSUBPD, VSUBPS VCVTPD2QQ, VCVTPD2UQQ, VCVTPD2UDQ, VCVTPS2UDQS, VCVTQQ2PD, VCVTQQ2PS, VCVTTPD2DQ, VCVTTPD2QQ, VCVTTPD2UDQ, VCVTTPD2UQQ, VCVTTPS2DQ, VCVTTPS2UDQ, VCVTUDQ2PS, VCVTUQQ2PD, VCVTUQQ2PS, VFIXUPIMMPD, VFIXUPIMMPS, VGETEXPPD, VGETEXPPS, VGETMANTPD, VGETMANTPS, VRANGEPD, VRANGEPS, VREDUCEPD, VREDUCEPS, VRNDSCALEPD, VRNDSCALEPS, VSCALEFPD, VSCALEFPS, VRCP28PD, VRCP28PS, VRSQRT28PD, VRSQRT28PS
Type E3	VADDS, VADDS, VCMPSD, VCMPS, VCVTPS2PD, VCVTSD2SS, VCVTSS2SD, VDIVSD, VDIVSS, VMAXSD, VMAXSS, VMINSD, VMINSS, VMULSD, VMULSS, VSQRTSD, VSQRTSS, VSUBSD, VSUBSS VCVTPS2QQ, VCVTPS2UQQ, VCVTTPS2QQ, VCVTTPS2UQQ, VFMADDxxxSD, VFMADDxxxSS, VFMSUBxxxSD, VFMSUBxxxSS, VFNMADDxxxSD, VFNMADDxxxSS, VFNMSUBxxxSD, VFNMSUBxxxSS, VFIXUPIMMSD, VFIXUPIMMSS, VGETEXPSD, VGETEXPSS, VGETMANTSD, VGETMANTSS, VRANGESD, VRANGESS, VREDUCESD, VREDUCESS, VRNDSCALESD, VRNDSCALESS, VSCALEFSD, VSCALEFSS, VRCP28SD, VRCP28SS, VRSQRT28SD, VRSQRT28SS
Type E3NF	VCOMISD, VCOMISS, VCVTSD2SI, VCVTSI2SD, VCVTSI2SS, VCVTSS2SI, VCVTSS2SD, VCVTSS2SI, VUCOMISD, VUCOMISS VCVTSD2USI, VCVTSS2USI, VCVTSS2USI, VCVTSS2USI, VCVTUSI2SD, VCVTUSI2SS

Table 2-43. EVEX Instructions in each Exception Class (Contd.)

Exception Class	Instruction
Type E4	VANDPD, VANDPS, VANDNPD, VANDNPS, VORPD, VORPS, VPABSD, VPABSQ, VPADDD, VPADDQ, VPANDD, VPANDQ, VPANDND, VPANDNQ, VPCMPEQD, VPCMPEQQ, VPCMPGTD, VPCMPGTQ, VPMAXSD, VPMAXSQ, VPMAXUD, VPMAXUQ, VPMINSD, VPMINSQ, VPMINUD, VPMINUQ, VPMULLD, VPMULLQ, VPMULUDQ, VPMULDQ, VPORD, VPORQ, VPSUBD, VPSUBQ, VPXORD, VPXORQ, VXORPD, VXORPS, VPSLLVD, VPSLLVQ, VBLENDMPD, VBLENDMPS, VPBLENDMD, VPBLENDMQ, VFPCCLASSPD, VFPCCLASSPS, VPCMPD, VPCMPQ, VPCMPUD, VPCMPUQ, VPLZCNTD, VPLZCNTQ, VPROLD, VPROLQ, (VPSLLD, VPSLLQ, VPSRAD, VPSRAQ, VPSRLD, VPSRLQ) <sup>1</sup> , VPTERNLOGD, VPTERNLOGQ, VPTESTMD, VPTESTMQ, VPTESTNMD, VPTESTNMQ, VRCP14PD, VRCP14PS, VRSQRT14PD, VRSQRT14PS, VPCONFLICTD, VPCONFLICTQ, VPSRAVW, VPSRAVD, VPSRAVW, VPSRAVQ, VPMADD52LUQ, VPMADD52HUQ
E4.nb <sup>2</sup>	VMOVUPD, VMOVUPS, VMOVDQU8, VMOVDQU16, VMOVDQU32, VMOVDQU64, VPCMPB, VPCMPW, VPCMPUB, VPCMPUW, VEXPANDPD, VEXPANDPS, VPCOMPRESSD, VPCOMPRESSQ, VPEXPANDD, VPEXPANDQ, VCOMPRESSPD, VCOMPRESSPS, VPABSB, VPABSW, VPADDB, VPADDW, VPADDSB, VPADDSW, VPADDUSB, VPADDUSW, VPAVGB, VPAVGW, VPCMPEQB, VPCMPEQW, VPCMPGTB, VPCMPGTW, VPMAXSB, VPMAXSW, VPMAXUB, VPMAXUW, VPMINSB, VPMINSW, VPMINUB, VPMINUW, VPMULHRW, VPMULHUW, VPMULHW, VPMULLW, VPSUBB, VPSUBW, VPSUBSB, VPSUBSW, VPTESTMB, VPTESTMW, VPTESTNMB, VPTESTNMW, VPSLLW, VPSRAW, VPSRLW, VPSLLW, VPSRLW
Type E4NF	VPACKSSDW, VPACKUSDW, VPSHUFD, VPUNPCKHDQ, VPUNPCKHQDQ, VPUNPCKLDQ, VPUNPCKLQDQ, VSHUFPD, VSHUFPS, VUNPCKHPD, VUNPCKHPS, VUNPCKLPD, VUNPCKLPS, VPERMD, VPERMPS, VPERMPD, VPERMQ, VALIGND, VALIGNQ, VPERMI2D, VPERMI2PS, VPERMI2PD, VPERMI2Q, VPERMT2D, VPERMT2PS, VPERMT2Q, VPERMT2PD, VPERMILPD, VPERMILPS, VSHUFI32X4, VSHUFI64X2, VSHUFF32X4, VSHUFF64X2, VPMULTISHIFTQB
E4NF.nb <sup>2</sup>	VDBPSADBW, VPACKSSWB, VPACKUSWB, VPALIGNR, VPMADDWD, VPMADDUBSW, VMOVSHDUP, VMOVSLDUP, VPSADBW, VPSHUFB, VPSHUFBW, VPSHUFLW, VPSLLDQ, VPSRLDQ, VPSLLW, VPSRAW, VPSRLW, (VPSLLD, VPSLLQ, VPSRAD, VPSRAQ, VPSRLD, VPSRLQ) <sup>3</sup> , VPUNPCKHBW, VPUNPCKHWD, VPUNPCKLBW, VPUNPCKLWD, VPERMW, VPERMI2W, VPERMT2W
Type E5	VCVTDQ2PD, PMOVSBW, PMOVSBW, PMOVXBD, PMOVXBQ, PMOVXWD, PMOVXWQ, PMOVXQD, PMOVZXBW, PMOVZXBW, PMOVZXBQ, PMOVZXWD, PMOVZXWQ, PMOVZXDQ, VCVTUDQ2PD
Type E5NF	VMOVDDUP
Type E6	VBROADCASTSS, VBROADCASTSD, VBROADCASTF32X4, VBROADCASTI32X4, VPBROADCASTB, VPBROADCASTD, VPBROADCASTW, VPBROADCASTQ, VBROADCASTF32X2, VBROADCASTF32X4, VBROADCASTF64X2, VBROADCASTF32X8, VBROADCASTF64X4, VBROADCASTI32X2, VBROADCASTI32X4, VBROADCASTI64X2, VBROADCASTI32X8, VBROADCASTI64X4, VFPCCLASSD, VFPCCLASSSS, VPMOVQB, VPMOVQB, VPMOVUSQB, VPMOVQW, VPMOVSQW, VPMOVUSQW, VPMOVQD, VPMOVSD, VPMOVUSQD, VPMOVDB, VPMOVSD, VPMOVUSDB, VPMOVDW, VPMOVSDW, VPMOVUSDW
Type E6NF	VEXTRACTF32X4, VEXTRACTF64X2, VEXTRACTF32X8, VINSERTF32X4, VINSERTF64X2, VINSERTF64X4, VINSERTF32X8, VINSERTI32X4, VINSERTI64X2, VINSERTI64X4, VINSERTI32X8, VEXTRACTI32X4, VEXTRACTI64X2, VEXTRACTI32X8, VEXTRACTI64X4, VPBROADCASTMB2Q, VPBROADCASTMW2D, VPMOVWB, VPMOVSWB, VPMOVUSWB
Type E7NM.128 <sup>4</sup>	VMOVLHPS, VMOVHPS
Type E7NM.	(VPBROADCASTD, VPBROADCASTQ, VPBROADCASTB, VPBROADCASTW) <sup>5</sup> , VPMOVM2B, VPMOVM2D, VPMOVM2Q, VPMOVM2W, VPMOVB2M, VPMOVD2M, VPMOVQ2M, VPMOVW2M

Table 2-43. EVEX Instructions in each Exception Class (Contd.)

Exception Class	Instruction
Type E9NF	VEXTRACTPS, VINSERTPS, VMOVHPD, VMOVHPS, VMOVLPD, VMOVLPS, VMOVD, VMOVQ, VPEXTRB, VPEXTRD, VPEXTRW, VPEXTRQ, VPINSRB, VPINSRD, VPINSRW, VPINSRQ
Type E10	VMOVSD, VMOVSS, VRCP14SD, VRCP14SS, VRSQRT14SD, VRSQRT14SS,
Type E10NF	(VCVTSI2SD, VCVTUSI2SD) <sup>6</sup>
Type E11	VCVTPH2PS, VCVTPS2PH
Type E12	VGATHERDPS, VGATHERDPD, VGATHERQPS, VGATHERQPD, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ, VPSCATTERDD, VPSCATTERDQ, VPSCATTERQD, VPSCATTERQQ, VSCATTERDPD, VSCATTERDPS, VSCATTERQPD, VSCATTERQPS
Type E12NP	VGATHERPFODPD, VGATHERPFODPS, VGATHERPFOQPD, VGATHERPFOQPS, VGATHERPF1DPD, VGATHERPF1DPS, VGATHERPF1QPD, VGATHERPF1QPS, VSCATTERPFODPD, VSCATTERPFODPS, VSCATTERPFOQPD, VSCATTERPFOQPS, VSCATTERPF1DPD, VSCATTERPF1DPS, VSCATTERPF1QPD, VSCATTERPF1QPS

**NOTES:**

1. Operand encoding Full tupletype with immediate.
2. Embedded broadcast is not supported with the “.nb” suffix.
3. Operand encoding Mem128 tupletype.
4. #UD raised if EVEX.L'L !=00b (VL=128).
5. The source operand is a general purpose register.
6. W0 encoding only.

## 2.7.1 Exceptions Type E1 and E1NF of EVEX-Encoded Instructions

EVEX-encoded instructions with memory alignment restrictions, and supporting memory fault suppression follow exception class E1.

**Table 2-44. Type E1 Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>State requirement, Table 2-37 not met.</li> <li>Opcode independent #UD condition in Table 2-38.</li> <li>Operand encoding #UD conditions in Table 2-39.</li> <li>Opmask encoding #UD condition of Table 2-40.</li> <li>If EVEX.b != 0.</li> <li>If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X	X	EVEX.512: Memory operand is not 64-byte aligned. EVEX.256: Memory operand is not 32-byte aligned. EVEX.128: Memory operand is not 16-byte aligned.
			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.

EVEX-encoded instructions with memory alignment restrictions, but do not support memory fault suppression follow exception class E1NF.

**Table 2-45. Type E1NF Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>State requirement, Table 2-37 not met.</li> <li>Opcode independent #UD condition in Table 2-38.</li> <li>Operand encoding #UD conditions in Table 2-39.</li> <li>Opmask encoding #UD condition of Table 2-40.</li> <li>If EVEX.b != 0.</li> <li>If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.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	EVEX.512: Memory operand is not 64-byte aligned. EVEX.256: Memory operand is not 32-byte aligned. EVEX.128: Memory operand is not 16-byte aligned.
			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.

## 2.7.2 Exceptions Type E2 of EVEX-Encoded Instructions

EVEX-encoded vector instructions with arithmetic semantic follow exception class E2.

**Table 2-46. Type E2 Class Exception Conditions**

Exception	Real	Virtual 8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	If EVEX.B=1, alignment checking is enabled, and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception, {sae} or {er} not set, and CR4.OSXMMEXCPT[bit 10] = 1.



## 2.7.3 Exceptions Type E3 and E3NF of EVEX-Encoded Instructions

EVEX-encoded scalar instructions with arithmetic semantic that support memory fault suppression follow exception class E3.

**Table 2-47. Type E3 Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception, {sae} or {er} not set, and CR4.OSXMMEXCPT[bit 10] = 1.

EVEX-encoded scalar instructions with arithmetic semantic that do not support memory fault suppression follow exception class E3NF.

**Table 2-48. Type E3NF Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			EVEX prefix.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.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		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.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception, {sae} or {er} not set, and CR4.OSXMMEXCPT[bit 10] = 1.

## 2.7.4 Exceptions Type E4 and E4NF of EVEX-Encoded Instructions

EVEX-encoded vector instructions that cause no SIMD FP exception and support memory fault suppression follow exception class E4.

**Table 2-49. Type E4 Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>State requirement, Table 2-37 not met.</li> <li>Opcode independent #UD condition in Table 2-38.</li> <li>Operand encoding #UD conditions in Table 2-39.</li> <li>Opmask encoding #UD condition of Table 2-40.</li> <li>If EVEX.b != 0 and in E4.nb subclass (see E4.nb entries in Table 2-43).</li> <li>If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	If EVEX.B=1, alignment checking is enabled, and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.

EVEX-encoded vector instructions that do not cause SIMD FP exception nor support memory fault suppression follow exception class E4NF.

**Table 2-50. Type E4NF Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0 and in E4NF.nb subclass (see E4NF.nb entries in Table 2-43).</li> <li>▪ If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.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		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.

## 2.7.5 Exceptions Type E5 and E5NF

EVEX-encoded scalar/partial-vector instructions that cause no SIMD FP exception and support memory fault suppression follow exception class E5.

**Table 2-51. Type E5 Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> <li>▪ If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.

EVEX-encoded scalar/partial vector instructions that do not cause SIMD FP exception nor support memory fault suppression follow exception class E5NF.

**Table 2-52. Type E5NF Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> <li>▪ If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If 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		If 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.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.

## 2.7.6 Exceptions Type E6 and E6NF

Table 2-53. Type E6 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> <li>▪ If EVEX.L'L != 10b (VL=512).</li> </ul>
			X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
			X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
Page Fault #PF(fault-code)			X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)			X	X	For 4 or 8 byte memory references if alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.

EVEX-encoded instructions that do not cause SIMD FP exception nor support memory fault suppression follow exception class E6NF.

**Table 2-54. Type E6NF Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> <li>▪ If EVEX.L'L != 10b (VL=512).</li> </ul>
			X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
			X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.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		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.
Page Fault #PF(fault-code)			X	X	For a page fault.
Alignment Check #AC(0)			X	X	For 4 or 8 byte memory references if alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.



## 2.7.7 Exceptions Type E7NM

EVEX-encoded instructions that cause no SIMD FP exception and do not reference memory follow exception class E7NM.

**Table 2-55. Type E7NM Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> <li>▪ Instruction specific EVEX.L'L restriction not met.</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.

## 2.7.8 Exceptions Type E9 and E9NF

EVEX-encoded vector or partial-vector instructions that do not cause no SIMD FP exception and support memory fault suppression follow exception class E9.

**Table 2-56. Type E9 Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>State requirement, Table 2-37 not met.</li> <li>Opcode independent #UD condition in Table 2-38.</li> <li>Operand encoding #UD conditions in Table 2-39.</li> <li>Opmask encoding #UD condition of Table 2-40.</li> <li>If EVEX.b != 0.</li> <li>If EVEX.L'L != 00b (VL=128).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.

EVEX-encoded vector or partial-vector instructions that must be encoded with VEX.L'L = 0, do not cause SIMD FP exception nor support memory fault suppression follow exception class E9NF.

**Table 2-57. Type E9NF Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> <li>▪ If EVEX.L'L != 00b (VL=128).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If 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		If 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.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

## 2.7.9 Exceptions Type E10

EVEX-encoded scalar instructions that ignore EVEX.L'L vector length encoding and do not cause no SIMD FP exception, support memory fault suppression follow exception class E10.

**Table 2-58. Type E10 Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>State requirement, Table 2-37 not met.</li> <li>Opcode independent #UD condition in Table 2-38.</li> <li>Operand encoding #UD conditions in Table 2-39.</li> <li>Opmask encoding #UD condition of Table 2-40.</li> <li>If EVEX.b != 0.</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.

EVEX-encoded scalar instructions that must be encoded with VEX.L'L = 0, do not cause SIMD FP exception nor support memory fault suppression follow exception class E10NF.

**Table 2-59. Type E10NF Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.

### 2.7.10 Exception Type E11 (EVEX-only, mem arg no AC, floating-point exceptions)

EVEX-encoded instructions that can cause SIMD FP exception, memory operand support fault suppression but do not cause #AC follow exception class E11.

**Table 2-60. Type E11 Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> <li>▪ If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF (fault-code)		X	X	X	If fault suppression not set, and a page fault.
SIMD Floating-Point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception, {sae} not set, and CR4.OSXMMEX-CPT[bit 10] = 1.

## 2.7.11 Exception Type E12 and E12NP (VSIB mem arg, no AC, no floating-point exceptions)

Table 2-61. Type E12 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>State requirement, Table 2-37 not met.</li> <li>Opcode independent #UD condition in Table 2-38.</li> <li>Operand encoding #UD conditions in Table 2-39.</li> <li>Opmask encoding #UD condition of Table 2-40.</li> <li>If EVEX.b != 0.</li> <li>If EVEX.L'L != 10b (VL=512).</li> <li>If vvvv != 1111b.</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	NA	If address size attribute is 16 bit.
	X	X	X	X	If ModR/M.mod = '11b'.
	X	X	X	X	If ModR/M.rm != '100b'.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
	X	X	X	X	If k0 is used (gather or scatter operation).
X	X	X	X	If index = destination register (gather operation).	
Device Not Available, #NM	X	X	X	X	If CR0.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		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.

EVEX-encoded prefetch instructions that do not cause #PF follow exception class E12NP.

**Table 2-62. Type E12NP Class Exception Conditions**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> <li>▪ Opmask encoding #UD condition of Table 2-40.</li> <li>▪ If EVEX.b != 0.</li> <li>▪ If EVEX.L'L != 10b (VL=512).</li> </ul>
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	NA	If address size attribute is 16 bit.
	X	X	X	X	If ModR/M.mod = '11b'.
	X	X	X	X	If ModR/M.rm != '100b'.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
X	X	X	X	If k0 is used (gather or scatter operation).	
Device Not Available, #NM	X	X	X	X	If CR0.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		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.



## 2.8 EXCEPTION CLASSIFICATIONS OF OPMASK INSTRUCTIONS

The exception behavior of VEX-encoded opmask instructions are listed below.

Exception conditions of Opmask instructions that do not address memory are listed as Type K20.

**Table 2-63. TYPE K20 Exception Definition (VEX-Encoded OpMask Instructions w/o Memory Arg)**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If relevant CPUID feature flag is '0'.
	X	X			If a VEX prefix is present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> </ul>
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
			X	X	If ModRM:[7:6] != 11b.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.

Exception conditions of Opmask instructions that address memory are listed as Type K21.

**Table 2-64. TYPE K21 Exception Definition (VEX-Encoded OpMask Instructions Addressing Memory)**

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If relevant CPUID feature flag is '0'.
	X	X			If a VEX prefix is present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: <ul style="list-style-type: none"> <li>▪ State requirement, Table 2-37 not met.</li> <li>▪ Opcode independent #UD condition in Table 2-38.</li> <li>▪ Operand encoding #UD conditions in Table 2-39.</li> </ul>
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
Stack, SS(0)	X	X	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		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.
				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.
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference of 8 bytes or less is made while the current privilege level is 3.

## 5. 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*.

-----  
Change to this chapter: Updates to tuple type naming throughout chapter; these updates are not covered in this document as they are considered minor in nature and extensive in page count. See changes listed in chapter 2 of Volume 2A for details on the minor adjustments to naming.

Updates to the following instructions are covered here with change bars: CALL, INT n/INTO/INT 3, IRET/IRETD, KMOVW/KMOVB/KMOVQ/KMOVD, and LSL.

## CALL—Call Procedure

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
E8 <i>cw</i>	CALL <i>rel16</i>	M	N.S.	Valid	Call near, relative, displacement relative to next instruction.
E8 <i>cd</i>	CALL <i>rel32</i>	M	Valid	Valid	Call near, relative, displacement relative to next instruction. 32-bit displacement sign extended to 64-bits in 64-bit mode.
FF /2	CALL <i>r/m16</i>	M	N.E.	Valid	Call near, absolute indirect, address given in <i>r/m16</i> .
FF /2	CALL <i>r/m32</i>	M	N.E.	Valid	Call near, absolute indirect, address given in <i>r/m32</i> .
FF /2	CALL <i>r/m64</i>	M	Valid	N.E.	Call near, absolute indirect, address given in <i>r/m64</i> .
9A <i>cd</i>	CALL <i>ptr16:16</i>	D	Invalid	Valid	Call far, absolute, address given in operand.
9A <i>cp</i>	CALL <i>ptr16:32</i>	D	Invalid	Valid	Call far, absolute, address given in operand.
FF /3	CALL <i>m16:16</i>	M	Valid	Valid	Call far, absolute indirect address given in <i>m16:16</i> . In 32-bit mode: if selector points to a gate, then RIP = 32-bit zero extended displacement taken from gate; else RIP = zero extended 16-bit offset from far pointer referenced in the instruction.
FF /3	CALL <i>m16:32</i>	M	Valid	Valid	In 64-bit mode: If selector points to a gate, then RIP = 64-bit displacement taken from gate; else RIP = zero extended 32-bit offset from far pointer referenced in the instruction.
REX.W + FF /3	CALL <i>m16:64</i>	M	Valid	N.E.	In 64-bit mode: If selector points to a gate, then RIP = 64-bit displacement taken from gate; else RIP = 64-bit offset from far pointer referenced in the instruction.

## Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
D	Offset	NA	NA	NA
M	ModRM:r/m ( <i>r</i> )	NA	NA	NA

## Description

Saves procedure linking information on the stack and branches to the called procedure specified using the target operand. The target operand specifies the address of the first instruction in the called procedure. The operand can be an immediate value, a general-purpose register, or a memory location.

This instruction can be used to execute four types of calls:

- **Near Call** — A call to a procedure in the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intra-segment call.
- **Far Call** — A call to a procedure located in a different segment than the current code segment, sometimes referred to as an inter-segment call.
- **Inter-privilege-level far call** — A far call to a procedure in a segment at a different privilege level than that of the currently executing program or procedure.
- **Task switch** — A call to a procedure located in a different task.

The latter two call types (inter-privilege-level call and task switch) can only be executed in protected mode. See “Calling Procedures Using Call and RET” in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*, for additional information on near, far, and inter-privilege-level calls. See Chapter 7, “Task Management,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*, for information on performing task switches with the CALL instruction.

**Near Call.** When executing a near call, the processor pushes the value of the EIP register (which contains the offset of the instruction following the CALL instruction) on the stack (for use later as a return-instruction pointer). The processor then branches to the address in the current code segment specified by the target operand. The target operand specifies either an absolute offset in the code segment (an offset from the base of the code segment) or a relative offset (a signed displacement relative to the current value of the instruction pointer in the EIP register; this value points to the instruction following the CALL instruction). The CS register is not changed on near calls.

For a near call absolute, an absolute offset is specified indirectly in a general-purpose register or a memory location (*r/m16*, *r/m32*, or *r/m64*). The operand-size attribute determines the size of the target operand (16, 32 or 64 bits). When in 64-bit mode, the operand size for near call (and all near branches) is forced to 64-bits. Absolute offsets are loaded directly into the EIP(RIP) register. If the operand size attribute is 16, the upper two bytes of the EIP register are cleared, resulting in a maximum instruction pointer size of 16 bits. When accessing an absolute offset indirectly using the stack pointer [ESP] as the base register, the base value used is the value of the ESP before the instruction executes.

A relative offset (*rel16* or *rel32*) is generally specified as a label in assembly code. But at the machine code level, it is encoded as a signed, 16- or 32-bit immediate value. This value is added to the value in the EIP(RIP) register. In 64-bit mode the relative offset is always a 32-bit immediate value which is sign extended to 64-bits before it is added to the value in the RIP register for the target calculation. As with absolute offsets, the operand-size attribute determines the size of the target operand (16, 32, or 64 bits). In 64-bit mode the target operand will always be 64-bits because the operand size is forced to 64-bits for near branches.

**Far Calls in Real-Address or Virtual-8086 Mode.** When executing a far call in real- address or virtual-8086 mode, the processor pushes the current value of both the CS and EIP registers on the stack for use as a return-instruction pointer. The processor then performs a “far branch” to the code segment and offset specified with the target operand for the called procedure. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). With the pointer method, the segment and offset of the called procedure is encoded in the instruction using a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address immediate. With the indirect method, the target operand specifies a memory location that contains a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address. The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The far address is loaded directly into the CS and EIP registers. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared.

**Far Calls in Protected Mode.** When the processor is operating in protected mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level
- Far call to a different privilege level (inter-privilege level call)
- Task switch (far call to another task)

In protected mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate, task gate, or TSS) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in protected mode is very similar to one carried out in real-address or virtual-8086 mode. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The new code segment selector and its descriptor are loaded into CS register; the offset from the instruction is loaded into the EIP register.

A call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. Using this mechanism provides an extra level of indirection and is the preferred method of making calls between 16-bit and 32-bit code segments.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a call gate. The segment selector specified by the target operand identifies the call gate. The target operand can specify the call gate segment selector either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch. (Note that when using a call gate to perform a far call to a segment at the same privilege level, no stack switch occurs.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack, an optional set of parameters from the calling procedure's stack, and the segment selector and instruction pointer for the calling procedure's code segment. (A value in the call gate descriptor determines how many parameters to copy to the new stack.) Finally, the processor branches to the address of the procedure being called within the new code segment.

Executing a task switch with the CALL instruction is similar to executing a call through a call gate. The target operand specifies the segment selector of the task gate for the new task activated by the switch (the offset in the target operand is ignored). The task gate in turn points to the TSS for the new task, which contains the segment selectors for the task's code and stack segments. Note that the TSS also contains the EIP value for the next instruction that was to be executed before the calling task was suspended. This instruction pointer value is loaded into the EIP register to re-start the calling task.

The CALL instruction can also specify the segment selector of the TSS directly, which eliminates the indirection of the task gate. See Chapter 7, "Task Management," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for information on the mechanics of a task switch.

When you execute a task switch with a CALL instruction, the nested task flag (NT) is set in the EFLAGS register and the new TSS's previous task link field is loaded with the old task's TSS selector. Code is expected to suspend this nested task by executing an IRET instruction which, because the NT flag is set, automatically uses the previous task link to return to the calling task. (See "Task Linking" in Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for information on nested tasks.) Switching tasks with the CALL instruction differs in this regard from JMP instruction. JMP does not set the NT flag and therefore does not expect an IRET instruction to suspend the task.

**Mixing 16-Bit and 32-Bit Calls.** When making far calls between 16-bit and 32-bit code segments, use a call gate. If the far call is from a 32-bit code segment to a 16-bit code segment, the call should be made from the first 64 KBytes of the 32-bit code segment. This is because the operand-size attribute of the instruction is set to 16, so only a 16-bit return address offset can be saved. Also, the call should be made using a 16-bit call gate so that 16-bit values can be pushed on the stack. See Chapter 21, "Mixing 16-Bit and 32-Bit Code," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, for more information.

**Far Calls in Compatibility Mode.** When the processor is operating in compatibility mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level, remaining in compatibility mode
- Far call to the same privilege level, transitioning to 64-bit mode
- Far call to a different privilege level (inter-privilege level call), transitioning to 64-bit mode

Note that a CALL instruction can not be used to cause a task switch in compatibility mode since task switches are not supported in IA-32e mode.

In compatibility mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in compatibility mode is very similar to one carried out in protected mode. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The new code segment selector and its descriptor are loaded into CS register and the offset from the instruction is loaded into the EIP register. The difference is that 64-bit mode may be entered. This is specified by the L bit in the new code segment descriptor.

Note that a 64-bit call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. However, using this mechanism requires that the target code segment descriptor have the L bit set, causing an entry to 64-bit mode.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a 64-bit call gate. The segment selector specified by the target operand identifies the call gate. The target

operand can specify the call gate segment selector either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the 16-byte call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is set to NULL. The new stack pointer is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch. (Note that when using a call gate to perform a far call to a segment at the same privilege level, an implicit stack switch occurs as a result of entering 64-bit mode. The SS selector is unchanged, but stack segment accesses use a segment base of 0x0, the limit is ignored, and the default stack size is 64-bits. The full value of RSP is used for the offset, of which the upper 32-bits are undefined.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack and the segment selector and instruction pointer for the calling procedure's code segment. (Parameter copy is not supported in IA-32e mode.) Finally, the processor branches to the address of the procedure being called within the new code segment.

**Near(Far) Calls in 64-bit Mode.** When the processor is operating in 64-bit mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level, transitioning to compatibility mode
- Far call to the same privilege level, remaining in 64-bit mode
- Far call to a different privilege level (inter-privilege level call), remaining in 64-bit mode

Note that in this mode the CALL instruction can not be used to cause a task switch in 64-bit mode since task switches are not supported in IA-32e mode.

In 64-bit mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in 64-bit mode is very similar to one carried out in compatibility mode. The target operand specifies an absolute far address indirectly with a memory location (*m16:16*, *m16:32* or *m16:64*). The form of CALL with a direct specification of absolute far address is not defined in 64-bit mode. The operand-size attribute determines the size of the offset (16, 32, or 64 bits) in the far address. The new code segment selector and its descriptor are loaded into the CS register; the offset from the instruction is loaded into the EIP register. The new code segment may specify entry either into compatibility or 64-bit mode, based on the L bit value.

A 64-bit call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. However, using this mechanism requires that the target code segment descriptor have the L bit set.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a 64-bit call gate. The segment selector specified by the target operand identifies the call gate. The target operand can only specify the call gate segment selector indirectly with a memory location (*m16:16*, *m16:32* or *m16:64*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the 16-byte call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is set to NULL. The new stack pointer is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch.

Note that when using a call gate to perform a far call to a segment at the same privilege level, an implicit stack switch occurs as a result of entering 64-bit mode. The SS selector is unchanged, but stack segment accesses use a segment base of 0x0, the limit is ignored, and the default stack size is 64-bits. (The full value of RSP is used for the offset.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack and the segment selector and instruction pointer for the calling procedure's code segment. (Parameter copy is not supported in IA-32e mode.) Finally, the processor branches to the address of the procedure being called within the new code segment.

**Operation**

```

IF near call
  THEN IF near relative call
    THEN
      IF OperandSize = 64
        THEN
          tempDEST ← SignExtend(DEST); (* DEST is rel32 *)
          tempRIP ← RIP + tempDEST;
          IF stack not large enough for a 8-byte return address
            THEN #SS(0); FI;
          Push(RIP);
          RIP ← tempRIP;
        FI;
      IF OperandSize = 32
        THEN
          tempEIP ← EIP + DEST; (* DEST is rel32 *)
          IF tempEIP is not within code segment limit THEN #GP(0); FI;
          IF stack not large enough for a 4-byte return address
            THEN #SS(0); FI;
          Push(EIP);
          EIP ← tempEIP;
        FI;
      IF OperandSize = 16
        THEN
          tempEIP ← (EIP + DEST) AND 0000FFFFH; (* DEST is rel16 *)
          IF tempEIP is not within code segment limit THEN #GP(0); FI;
          IF stack not large enough for a 2-byte return address
            THEN #SS(0); FI;
          Push(IP);
          EIP ← tempEIP;
        FI;
    ELSE (* Near absolute call *)
      IF OperandSize = 64
        THEN
          tempRIP ← DEST; (* DEST is r/m64 *)
          IF stack not large enough for a 8-byte return address
            THEN #SS(0); FI;
          Push(RIP);
          RIP ← tempRIP;
        FI;
      IF OperandSize = 32
        THEN
          tempEIP ← DEST; (* DEST is r/m32 *)
          IF tempEIP is not within code segment limit THEN #GP(0); FI;
          IF stack not large enough for a 4-byte return address
            THEN #SS(0); FI;
          Push(EIP);
          EIP ← tempEIP;
        FI;
      IF OperandSize = 16
        THEN
          tempEIP ← DEST AND 0000FFFFH; (* DEST is r/m16 *)
          IF tempEIP is not within code segment limit THEN #GP(0); FI;

```



```

        IF stack not large enough for a 2-byte return address
            THEN #SS(0); FI;
        Push(IP);
        EIP ← tempEIP;
    FI;
FI;rel/abs
FI; near

IF far call and (PE = 0 or (PE = 1 and VM = 1)) (* Real-address or virtual-8086 mode *)
    THEN
        IF OperandSize = 32
            THEN
                IF stack not large enough for a 6-byte return address
                    THEN #SS(0); FI;
                IF DEST[31:16] is not zero THEN #GP(0); FI;
                Push(CS); (* Padded with 16 high-order bits *)
                Push(EIP);
                CS ← DEST[47:32]; (* DEST is ptr16:32 or [m16:32] *)
                EIP ← DEST[31:0]; (* DEST is ptr16:32 or [m16:32] *)
            ELSE (* OperandSize = 16 *)
                IF stack not large enough for a 4-byte return address
                    THEN #SS(0); FI;
                Push(CS);
                Push(IP);
                CS ← DEST[31:16]; (* DEST is ptr16:16 or [m16:16] *)
                EIP ← DEST[15:0]; (* DEST is ptr16:16 or [m16:16]; clear upper 16 bits *)
            FI;
    FI;
FI;

IF far call and (PE = 1 and VM = 0) (* Protected mode or IA-32e Mode, not virtual-8086 mode*)
    THEN
        IF segment selector in target operand NULL
            THEN #GP(0); FI;
        IF segment selector index not within descriptor table limits
            THEN #GP(new code segment selector); FI;
        Read type and access rights of selected segment descriptor;
        IF IA32_EFER.LMA = 0
            THEN
                IF segment type is not a conforming or nonconforming code segment, call
                gate, task gate, or TSS
                    THEN #GP(segment selector); FI;
            ELSE
                IF segment type is not a conforming or nonconforming code segment or
                64-bit call gate,
                    THEN #GP(segment selector); FI;
        FI;
        Depending on type and access rights:
        GO TO CONFORMING-CODE-SEGMENT;
        GO TO NONCONFORMING-CODE-SEGMENT;
        GO TO CALL-GATE;
        GO TO TASK-GATE;
        GO TO TASK-STATE-SEGMENT;
    FI;
FI;

```

## CONFORMING-CODE-SEGMENT:

```

IF L bit = 1 and D bit = 1 and IA32_EFER.LMA = 1
    THEN GP(new code segment selector); FI;
IF DPL > CPL
    THEN #GP(new code segment selector); FI;
IF segment not present
    THEN #NP(new code segment selector); FI;
IF stack not large enough for return address
    THEN #SS(0); FI;
tempEIP ← DEST(Offset);
IF target mode = Compatibility mode
    THEN tempEIP ← tempEIP AND 00000000_FFFFFFFFH; FI;
IF OperandSize = 16
    THEN
        tempEIP ← tempEIP AND 0000FFFFH; FI; (* Clear upper 16 bits *)
IF (EFER.LMA = 0 or target mode = Compatibility mode) and (tempEIP outside new code
segment limit)
    THEN #GP(0); FI;
IF tempEIP is non-canonical
    THEN #GP(0); FI;
IF OperandSize = 32
    THEN
        Push(CS); (* Padded with 16 high-order bits *)
        Push(EIP);
        CS ← DEST(CodeSegmentSelector);
        (* Segment descriptor information also loaded *)
        CS(RPL) ← CPL;
        EIP ← tempEIP;
    ELSE
        IF OperandSize = 16
            THEN
                Push(CS);
                Push(IP);
                CS ← DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) ← CPL;
                EIP ← tempEIP;
            ELSE (* OperandSize = 64 *)
                Push(CS); (* Padded with 48 high-order bits *)
                Push(RIP);
                CS ← DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) ← CPL;
                RIP ← tempEIP;
        FI;
    FI;
END;

```

## NONCONFORMING-CODE-SEGMENT:

```

IF L-Bit = 1 and D-BIT = 1 and IA32_EFER.LMA = 1
    THEN GP(new code segment selector); FI;
IF (RPL > CPL) or (DPL ≠ CPL)
    THEN #GP(new code segment selector); FI;
IF segment not present

```

```

    THEN #NP(new code segment selector); FI;
IF stack not large enough for return address
    THEN #SS(0); FI;
tempEIP ← DEST(Offset);
IF target mode = Compatibility mode
    THEN tempEIP ← tempEIP AND 00000000_FFFFFFFFH; FI;
IF OperandSize = 16
    THEN tempEIP ← tempEIP AND 0000FFFFH; FI; (* Clear upper 16 bits *)
IF (EFER.LMA = 0 or target mode = Compatibility mode) and (tempEIP outside new code
segment limit)
    THEN #GP(0); FI;
IF tempEIP is non-canonical
    THEN #GP(0); FI;
IF OperandSize = 32
    THEN
        Push(CS); (* Padded with 16 high-order bits *)
        Push(EIP);
        CS ← DEST(CodeSegmentSelector);
        (* Segment descriptor information also loaded *)
        CS(RPL) ← CPL;
        EIP ← tempEIP;
    ELSE
        IF OperandSize = 16
            THEN
                Push(CS);
                Push(IP);
                CS ← DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) ← CPL;
                EIP ← tempEIP;
            ELSE (* OperandSize = 64 *)
                Push(CS); (* Padded with 48 high-order bits *)
                Push(RIP);
                CS ← DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) ← CPL;
                RIP ← tempEIP;
            FI;
        FI;
END;

```

## CALL-GATE:

```

IF call gate (DPL < CPL) or (RPL > DPL)
    THEN #GP(call-gate selector); FI;
IF call gate not present
    THEN #NP(call-gate selector); FI;
IF call-gate code-segment selector is NULL
    THEN #GP(0); FI;
IF call-gate code-segment selector index is outside descriptor table limits
    THEN #GP(call-gate code-segment selector); FI;
Read call-gate code-segment descriptor;
IF call-gate code-segment descriptor does not indicate a code segment
or call-gate code-segment descriptor DPL > CPL
    THEN #GP(call-gate code-segment selector); FI;

```

```

IF IA32_EFER.LMA = 1 AND (call-gate code-segment descriptor is
not a 64-bit code segment or call-gate code-segment descriptor has both L-bit and D-bit set)
    THEN #GP(call-gate code-segment selector); FI;
IF call-gate code segment not present
    THEN #NP(call-gate code-segment selector); FI;
IF call-gate code segment is non-conforming and DPL < CPL
    THEN go to MORE-PRIVILEGE;
    ELSE go to SAME-PRIVILEGE;
FI;
END;

```

MORE-PRIVILEGE:

```

IF current TSS is 32-bit
    THEN
        TSSstackAddress ← (new code-segment DPL * 8) + 4;
        IF (TSSstackAddress + 5) > current TSS limit
            THEN #TS(current TSS selector); FI;
        NewSS ← 2 bytes loaded from (TSS base + TSSstackAddress + 4);
        NewESP ← 4 bytes loaded from (TSS base + TSSstackAddress);
    ELSE
        IF current TSS is 16-bit
            THEN
                TSSstackAddress ← (new code-segment DPL * 4) + 2
                IF (TSSstackAddress + 3) > current TSS limit
                    THEN #TS(current TSS selector); FI;
                NewSS ← 2 bytes loaded from (TSS base + TSSstackAddress + 2);
                NewESP ← 2 bytes loaded from (TSS base + TSSstackAddress);
            ELSE (* current TSS is 64-bit *)
                TSSstackAddress ← (new code-segment DPL * 8) + 4;
                IF (TSSstackAddress + 7) > current TSS limit
                    THEN #TS(current TSS selector); FI;
                NewSS ← new code-segment DPL; (* NULL selector with RPL = new CPL *)
                NewRSP ← 8 bytes loaded from (current TSS base + TSSstackAddress);
        FI;
FI;
IF IA32_EFER.LMA = 0 and NewSS is NULL
    THEN #TS(NewSS); FI;
Read new stack-segment descriptor;
IF IA32_EFER.LMA = 0 and (NewSS RPL ≠ new code-segment DPL
or new stack-segment DPL ≠ new code-segment DPL or new stack segment is not a
writable data segment)
    THEN #TS(NewSS); FI;
IF IA32_EFER.LMA = 0 and new stack segment not present
    THEN #SS(NewSS); FI;
IF CallGateSize = 32
    THEN
        IF new stack does not have room for parameters plus 16 bytes
            THEN #SS(NewSS); FI;
        IF CallGate(InstructionPointer) not within new code-segment limit
            THEN #GP(0); FI;
        SS ← newSS; (* Segment descriptor information also loaded *)
        ESP ← newESP;
        CS:EIP ← CallGate(CS:InstructionPointer);
        (* Segment descriptor information also loaded *)

```

```

    Push(oldSS:oldESP); (* From calling procedure *)
    temp ← parameter count from call gate, masked to 5 bits;
    Push(parameters from calling procedure's stack, temp)
    Push(oldCS:oldEIP); (* Return address to calling procedure *)
ELSE
    IF CallGateSize = 16
        THEN
            IF new stack does not have room for parameters plus 8 bytes
                THEN #SS(NewSS); FI;
            IF (CallGate(InstructionPointer) AND FFFFH) not in new code-segment limit
                THEN #GP(0); FI;
            SS ← newSS; (* Segment descriptor information also loaded *)
            ESP ← newESP;
            CS:IP ← CallGate(CS:InstructionPointer);
            (* Segment descriptor information also loaded *)
            Push(oldSS:oldESP); (* From calling procedure *)
            temp ← parameter count from call gate, masked to 5 bits;
            Push(parameters from calling procedure's stack, temp)
            Push(oldCS:oldEIP); (* Return address to calling procedure *)
        ELSE (* CallGateSize = 64 *)
            IF pushing 32 bytes on the stack would use a non-canonical address
                THEN #SS(NewSS); FI;
            IF (CallGate(InstructionPointer) is non-canonical)
                THEN #GP(0); FI;
            SS ← NewSS; (* NewSS is NULL)
            RSP ← NewESP;
            CS:IP ← CallGate(CS:InstructionPointer);
            (* Segment descriptor information also loaded *)
            Push(oldSS:oldESP); (* From calling procedure *)
            Push(oldCS:oldEIP); (* Return address to calling procedure *)
        FI;
    FI;
    CPL ← CodeSegment(DPL)
    CS(RPL) ← CPL
END;

SAME-PRIVILEGE:
    IF CallGateSize = 32
        THEN
            IF stack does not have room for 8 bytes
                THEN #SS(0); FI;
            IF CallGate(InstructionPointer) not within code segment limit
                THEN #GP(0); FI;
            CS:EIP ← CallGate(CS:EIP) (* Segment descriptor information also loaded *)
            Push(oldCS:oldEIP); (* Return address to calling procedure *)
        ELSE
            If CallGateSize = 16
                THEN
                    IF stack does not have room for 4 bytes
                        THEN #SS(0); FI;
                    IF CallGate(InstructionPointer) not within code segment limit
                        THEN #GP(0); FI;
                    CS:IP ← CallGate(CS:instruction pointer);
                    (* Segment descriptor information also loaded *)
                FI;
        FI;

```

```

        Push(oldCS:oldIP); (* Return address to calling procedure *)
    ELSE (* CallGateSize = 64)
        IF pushing 16 bytes on the stack touches non-canonical addresses
            THEN #SS(0); FI;
        IF RIP non-canonical
            THEN #GP(0); FI;
        CS:IP ← CallGate(CS:instruction pointer);
        (* Segment descriptor information also loaded *)
        Push(oldCS:oldIP); (* Return address to calling procedure *)
    FI;
FI;
CS(RPL) ← CPL
END;

```

**TASK-GATE:**

```

    IF task gate DPL < CPL or RPL
        THEN #GP(task gate selector); FI;
    IF task gate not present
        THEN #NP(task gate selector); FI;
    Read the TSS segment selector in the task-gate descriptor;
    IF TSS segment selector local/global bit is set to local
    or index not within GDT limits
        THEN #GP(TSS selector); FI;
    Access TSS descriptor in GDT;
    IF descriptor is not a TSS segment
        THEN #GP(TSS selector); FI;
    IF TSS descriptor specifies that the TSS is busy
        THEN #GP(TSS selector); FI;
    IF TSS not present
        THEN #NP(TSS selector); FI;
    SWITCH-TASKS (with nesting) to TSS;
    IF EIP not within code segment limit
        THEN #GP(0); FI;
END;

```

**TASK-STATE-SEGMENT:**

```

    IF TSS DPL < CPL or RPL
    or TSS descriptor indicates TSS not available
        THEN #GP(TSS selector); FI;
    IF TSS is not present
        THEN #NP(TSS selector); FI;
    SWITCH-TASKS (with nesting) to TSS;
    IF EIP not within code segment limit
        THEN #GP(0); FI;
END;

```

**Flags Affected**

All flags are affected if a task switch occurs; no flags are affected if a task switch does not occur.

**Protected Mode Exceptions**

#GP(0)	<p>If the target offset in destination operand is beyond the new code segment limit.</p> <p>If the segment selector in the destination operand is NULL.</p> <p>If the code segment selector in the gate is NULL.</p> <p>If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.</p> <p>If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.</p>
#GP(selector)	<p>If a code segment or gate or TSS selector index is outside descriptor table limits.</p> <p>If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or task state segment.</p> <p>If the DPL for a nonconforming-code segment is not equal to the CPL or the RPL for the segment's segment selector is greater than the CPL.</p> <p>If the DPL for a conforming-code segment is greater than the CPL.</p> <p>If the DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL or than the RPL of the call-gate, task-gate, or TSS's segment selector.</p> <p>If the segment descriptor for a segment selector from a call gate does not indicate it is a code segment.</p> <p>If the segment selector from a call gate is beyond the descriptor table limits.</p> <p>If the DPL for a code-segment obtained from a call gate is greater than the CPL.</p> <p>If the segment selector for a TSS has its local/global bit set for local.</p> <p>If a TSS segment descriptor specifies that the TSS is busy or not available.</p>
#SS(0)	<p>If pushing the return address, parameters, or stack segment pointer onto the stack exceeds the bounds of the stack segment, when no stack switch occurs.</p> <p>If a memory operand effective address is outside the SS segment limit.</p>
#SS(selector)	<p>If pushing the return address, parameters, or stack segment pointer onto the stack exceeds the bounds of the stack segment, when a stack switch occurs.</p> <p>If the SS register is being loaded as part of a stack switch and the segment pointed to is marked not present.</p> <p>If stack segment does not have room for the return address, parameters, or stack segment pointer, when stack switch occurs.</p>
#NP(selector)	<p>If a code segment, data segment, call gate, task gate, or TSS is not present.</p>
#TS(selector)	<p>If the new stack segment selector and ESP are beyond the end of the TSS.</p> <p>If the new stack segment selector is NULL.</p> <p>If the RPL of the new stack segment selector in the TSS is not equal to the DPL of the code segment being accessed.</p> <p>If DPL of the stack segment descriptor for the new stack segment is not equal to the DPL of the code segment descriptor.</p> <p>If the new stack segment is not a writable data segment.</p> <p>If segment-selector index for stack segment is outside descriptor table limits.</p>
#PF(fault-code)	<p>If a page fault occurs.</p>
#AC(0)	<p>If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.</p>
#UD	<p>If the LOCK prefix is used.</p>

**Real-Address Mode Exceptions**

#GP	<p>If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.</p> <p>If the target offset is beyond the code segment limit.</p>
#UD	<p>If the LOCK prefix is used.</p>

**Virtual-8086 Mode Exceptions**

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

**Compatibility Mode Exceptions**

Same exceptions as in protected mode.

#GP(selector)	If a memory address accessed by the selector is in non-canonical space.
#GP(0)	If the target offset in the destination operand is non-canonical.

**64-Bit Mode Exceptions**

#GP(0)	If a memory address is non-canonical. If target offset in destination operand is non-canonical. If the segment selector in the destination operand is NULL. If the code segment selector in the 64-bit gate is NULL.
#GP(selector)	If code segment or 64-bit call gate is outside descriptor table limits. If code segment or 64-bit call gate overlaps non-canonical space. If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, or 64-bit call gate. If the segment descriptor pointed to by the segment selector in the destination operand is a code segment and has both the D-bit and the L-bit set. If the DPL for a nonconforming-code segment is not equal to the CPL, or the RPL for the segment's segment selector is greater than the CPL. If the DPL for a conforming-code segment is greater than the CPL. If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate. If the upper type field of a 64-bit call gate is not 0x0. If the segment selector from a 64-bit call gate is beyond the descriptor table limits. If the DPL for a code-segment obtained from a 64-bit call gate is greater than the CPL. If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear. If the segment descriptor for a segment selector from the 64-bit call gate does not indicate it is a code segment.
#SS(0)	If pushing the return offset or CS selector onto the stack exceeds the bounds of the stack segment when no stack switch occurs. If a memory operand effective address is outside the SS segment limit. If the stack address is in a non-canonical form.
#SS(selector)	If pushing the old values of SS selector, stack pointer, EFLAGS, CS selector, offset, or error code onto the stack violates the canonical boundary when a stack switch occurs.
#NP(selector)	If a code segment or 64-bit call gate is not present.
#TS(selector)	If the load of the new RSP exceeds the limit of the TSS.
#UD	(64-bit mode only) If a far call is direct to an absolute address in memory. If the LOCK prefix is used.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.



## INT *n*/INTO/INT 3—Call to Interrupt Procedure

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
CC	INT 3	Z0	Valid	Valid	Interrupt 3—trap to debugger.
CD <i>ib</i>	INT <i>imm8</i>	I	Valid	Valid	Interrupt vector specified by immediate byte.
CE	INTO	Z0	Invalid	Valid	Interrupt 4—if overflow flag is 1.

### Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	NA	NA	NA	NA
I	imm8	NA	NA	NA

### Description

The INT *n* instruction generates a call to the interrupt or exception handler specified with the destination operand (see the section titled “Interrupts and Exceptions” in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*). The destination operand specifies a vector from 0 to 255, encoded as an 8-bit unsigned intermediate value. Each vector provides an index to a gate descriptor in the IDT. The first 32 vectors are reserved by Intel for system use. Some of these vectors are used for internally generated exceptions.

The INT *n* instruction is the general mnemonic for executing a software-generated call to an interrupt handler. The INTO instruction is a special mnemonic for calling overflow exception (#OF), exception 4. The overflow interrupt checks the OF flag in the EFLAGS register and calls the overflow interrupt handler if the OF flag is set to 1. (The INTO instruction cannot be used in 64-bit mode.)

The INT 3 instruction generates a special one byte opcode (CC) that is intended for calling the debug exception handler. (This one byte form is valuable because it can be used to replace the first byte of any instruction with a breakpoint, including other one byte instructions, without over-writing other code).

An interrupt generated by INTO or INT3 (CC) differs from one generated by INT *n* in the following ways:

- The normal IOPL checks do not occur in virtual-8086 mode. The interrupt is taken (without fault) with any IOPL value.
- The interrupt redirection enabled by the virtual-8086 mode extensions (VME) does not occur. The interrupt is always handled by a protected-mode handler.

(These features do not pertain to CD03, the “normal” 2-byte opcode for INT 3. Intel and Microsoft assemblers will not generate the CD03 opcode from any mnemonic, but this opcode can be created by direct numeric code definition or by self-modifying code.)

The action of the INT *n* instruction (including the INTO and INT 3 instructions) is similar to that of a far call made with the CALL instruction. The primary difference is that with the INT *n* instruction, the EFLAGS register is pushed onto the stack before the return address. (The return address is a far address consisting of the current values of the CS and EIP registers.) Returns from interrupt procedures are handled with the IRET instruction, which pops the EFLAGS information and return address from the stack.

The vector specifies an interrupt descriptor in the interrupt descriptor table (IDT); that is, it provides index into the IDT. The selected interrupt descriptor in turn contains a pointer to an interrupt or exception handler procedure. In protected mode, the IDT contains an array of 8-byte descriptors, each of which is an interrupt gate, trap gate, or task gate. In real-address mode, the IDT is an array of 4-byte far pointers (2-byte code segment selector and a 2-byte instruction pointer), each of which point directly to a procedure in the selected segment. (Note that in real-address mode, the IDT is called the **interrupt vector table**, and its pointers are called interrupt vectors.)

The following decision table indicates which action in the lower portion of the table is taken given the conditions in the upper portion of the table. Each Y in the lower section of the decision table represents a procedure defined in the “Operation” section for this instruction (except #GP).

Table 3-51. Decision Table

PE	0	1	1	1	1	1	1	1
VM	-	-	-	-	-	0	1	1
IOPL	-	-	-	-	-	-	<3	=3
DPL/CPL RELATIONSHIP	-	DPL < CPL	-	DPL > CPL	DPL = CPL or C	DPL < CPL & NC	-	-
INTERRUPT TYPE	-	S/W	-	-	-	-	-	-
GATE TYPE	-	-	Task	Trap or Interrupt	Trap or Interrupt	Trap or Interrupt	Trap or Interrupt	Trap or Interrupt
REAL-ADDRESS-MODE	Y							
PROTECTED-MODE		Y	Y	Y	Y	Y	Y	Y
TRAP-OR-INTERRUPT-GATE				Y	Y	Y	Y	Y
INTER-PRIVILEGE-LEVEL-INTERRUPT						Y		
INTRA-PRIVILEGE-LEVEL-INTERRUPT					Y			
INTERRUPT-FROM-VIRTUAL-8086-MODE								Y
TASK-GATE			Y					
#GP		Y		Y			Y	

**NOTES:**

- Don't Care.
- Y Yes, action taken.
- Blank Action not taken.

When the processor is executing in virtual-8086 mode, the IOPL determines the action of the INT *n* instruction. If the IOPL is less than 3, the processor generates a #GP(selector) exception; if the IOPL is 3, the processor executes a protected mode interrupt to privilege level 0. The interrupt gate's DPL must be set to 3 and the target CPL of the interrupt handler procedure must be 0 to execute the protected mode interrupt to privilege level 0.

The interrupt descriptor table register (IDTR) specifies the base linear address and limit of the IDT. The initial base address value of the IDTR after the processor is powered up or reset is 0.

**Operation**

The following operational description applies not only to the INT *n* and INTO instructions, but also to external interrupts, nonmaskable interrupts (NMIs), and exceptions. Some of these events push onto the stack an error code.

The operational description specifies numerous checks whose failure may result in delivery of a nested exception. In these cases, the original event is not delivered.

The operational description specifies the error code delivered by any nested exception. In some cases, the error code is specified with a pseudofunction `error_code(num, idt, ext)`, where `idt` and `ext` are bit values. The pseudofunction produces an error code as follows: (1) if `idt` is 0, the error code is  $(num \& FCH) \mid ext$ ; (2) if `idt` is 1, the error code is  $(num \ll 3) \mid 2 \mid ext$ .

In many cases, the pseudofunction `error_code` is invoked with a pseudovisible `EXT`. The value of `EXT` depends on the nature of the event whose delivery encountered a nested exception: if that event is a software interrupt, `EXT` is 0; otherwise, `EXT` is 1.

```

IF PE = 0
  THEN
    GOTO REAL-ADDRESS-MODE;
  ELSE (* PE = 1 *)
    IF (EFLAGS.VM = 1 AND CR4.VME = 0 AND IOPL < 3 AND INT n)
      THEN
        #GP(0); (* Bit 0 of error code is 0 because INT n *)
      ELSE
        IF (EFLAGS.VM = 1 AND CR4.VME = 1 AND INT n)
          THEN
            Consult bit n of the software interrupt redirection bit map in the TSS;
            IF bit n is clear
              THEN (* redirect interrupt to 8086 program interrupt handler *)
                Push EFLAGS[15:0]; (* if IOPL < 3, save VIF in IF position and save IOPL position as 3 *)
                Push CS;
                Push IP;
                IF IOPL = 3
                  THEN IF ← 0; (* Clear interrupt flag *)
                  ELSE VIF ← 0; (* Clear virtual interrupt flag *)
                FI;
                TF ← 0; (* Clear trap flag *)
                load CS and EIP (lower 16 bits only) from entry n in interrupt vector table referenced from TSS;
              ELSE
                IF IOPL = 3
                  THEN GOTO PROTECTED-MODE;
                  ELSE #GP(0); (* Bit 0 of error code is 0 because INT n *)
                FI;
            FI;
          ELSE (* Protected mode, IA-32e mode, or virtual-8086 mode interrupt *)
            IF (IA32_EFER.LMA = 0)
              THEN (* Protected mode, or virtual-8086 mode interrupt *)
                GOTO PROTECTED-MODE;
              ELSE (* IA-32e mode interrupt *)
                GOTO IA-32e-MODE;
            FI;
          FI;
        FI;
      FI;
    FI;
  REAL-ADDRESS-MODE:
    IF ((vector_number << 2) + 3) is not within IDT limit
      THEN #GP; FI;
    IF stack not large enough for a 6-byte return information
      THEN #SS; FI;
    Push (EFLAGS[15:0]);
    IF ← 0; (* Clear interrupt flag *)
    TF ← 0; (* Clear trap flag *)
    AC ← 0; (* Clear AC flag *)
    Push(CS);
    Push(IP);
    (* No error codes are pushed in real-address mode*)
    CS ← IDT(Descriptor (vector_number << 2), selector);
    EIP ← IDT(Descriptor (vector_number << 2), offset); (* 16 bit offset AND 0000FFFFH *)
  END;
  PROTECTED-MODE:

```

## INSTRUCTION SET REFERENCE, A-L

```

IF ((vector_number << 3) + 7) is not within IDT limits
or selected IDT descriptor is not an interrupt-, trap-, or task-gate type
    THEN #GP(error_code(vector_number,1,EXT)); FI;
    (* idt operand to error_code set because vector is used *)
IF software interrupt (* Generated by INT n, INT3, or INTO *)
    THEN
    IF gate DPL < CPL (* PE = 1, DPL < CPL, software interrupt *)
        THEN #GP(error_code(vector_number,1,0)); FI;
        (* idt operand to error_code set because vector is used *)
        (* ext operand to error_code is 0 because INT n, INT3, or INTO*)
    FI;
IF gate not present
    THEN #NP(error_code(vector_number,1,EXT)); FI;
    (* idt operand to error_code set because vector is used *)
IF task gate (* Specified in the selected interrupt table descriptor *)
    THEN GOTO TASK-GATE;
    ELSE GOTO TRAP-OR-INTERRUPT-GATE; (* PE = 1, trap/interrupt gate *)
FI;
END;
IA-32e-MODE:
IF INTO and CS.L = 1 (64-bit mode)
    THEN #UD;
FI;
IF ((vector_number << 4) + 15) is not in IDT limits
or selected IDT descriptor is not an interrupt-, or trap-gate type
    THEN #GP(error_code(vector_number,1,EXT));
    (* idt operand to error_code set because vector is used *)
FI;
IF software interrupt (* Generated by INT n, INT 3, or INTO *)
    THEN
    IF gate DPL < CPL (* PE = 1, DPL < CPL, software interrupt *)
        THEN #GP(error_code(vector_number,1,0));
        (* idt operand to error_code set because vector is used *)
        (* ext operand to error_code is 0 because INT n, INT3, or INTO*)
    FI;
    IF gate not present
        THEN #NP(error_code(vector_number,1,EXT));
        (* idt operand to error_code set because vector is used *)
    FI;
    GOTO TRAP-OR-INTERRUPT-GATE; (* Trap/interrupt gate *)
END;
TASK-GATE: (* PE = 1, task gate *)
    Read TSS selector in task gate (IDT descriptor);
    IF local/global bit is set to local or index not within GDT limits
        THEN #GP(error_code(TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    Access TSS descriptor in GDT;
    IF TSS descriptor specifies that the TSS is busy (low-order 5 bits set to 00001)
        THEN #GP(TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    IF TSS not present
        THEN #NP(TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)

```

```

SWITCH-TASKS (with nesting) to TSS;
IF interrupt caused by fault with error code
  THEN
    IF stack limit does not allow push of error code
      THEN #SS(EXT); FI;
    Push(error code);
  FI;
IF EIP not within code segment limit
  THEN #GP(EXT); FI;
END;
TRAP-OR-INTERRUPT-GATE:
  Read new code-segment selector for trap or interrupt gate (IDT descriptor);
  IF new code-segment selector is NULL
    THEN #GP(EXT); FI; (* Error code contains NULL selector *)
  IF new code-segment selector is not within its descriptor table limits
    THEN #GP(error_code(new code-segment selector,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
  Read descriptor referenced by new code-segment selector;
  IF descriptor does not indicate a code segment or new code-segment DPL > CPL
    THEN #GP(error_code(new code-segment selector,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
  IF new code-segment descriptor is not present,
    THEN #NP(error_code(new code-segment selector,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
  IF new code segment is non-conforming with DPL < CPL
    THEN
      IF VM = 0
        THEN
          GOTO INTER-PRIVILEGE-LEVEL-INTERRUPT;
          (* PE = 1, VM = 0, interrupt or trap gate, nonconforming code segment,
          DPL < CPL *)
        ELSE (* VM = 1 *)
          IF new code-segment DPL ≠ 0
            THEN #GP(error_code(new code-segment selector,0,EXT));
            (* idt operand to error_code is 0 because selector is used *)
          GOTO INTERRUPT-FROM-VIRTUAL-8086-MODE; FI;
          (* PE = 1, interrupt or trap gate, DPL < CPL, VM = 1 *)
        FI;
      ELSE (* PE = 1, interrupt or trap gate, DPL ≥ CPL *)
        IF VM = 1
          THEN #GP(error_code(new code-segment selector,0,EXT));
          (* idt operand to error_code is 0 because selector is used *)
        IF new code segment is conforming or new code-segment DPL = CPL
          THEN
            GOTO INTRA-PRIVILEGE-LEVEL-INTERRUPT;
          ELSE (* PE = 1, interrupt or trap gate, nonconforming code segment, DPL > CPL *)
            #GP(error_code(new code-segment selector,0,EXT));
            (* idt operand to error_code is 0 because selector is used *)
          FI;
        FI;
      FI;
    END;
INTER-PRIVILEGE-LEVEL-INTERRUPT:
  (* PE = 1, interrupt or trap gate, non-conforming code segment, DPL < CPL *)
  IF (IA32_EFER.LMA = 0) (* Not IA-32e mode *)

```

```

THEN
(* Identify stack-segment selector for new privilege level in current TSS *)
  IF current TSS is 32-bit
    THEN
      TSSstackAddress ← (new code-segment DPL << 3) + 4;
      IF (TSSstackAddress + 5) > current TSS limit
        THEN #TS(error_code(current TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
      NewSS ← 2 bytes loaded from (TSS base + TSSstackAddress + 4);
      NewESP ← 4 bytes loaded from (TSS base + TSSstackAddress);
    ELSE (* current TSS is 16-bit *)
      TSSstackAddress ← (new code-segment DPL << 2) + 2
      IF (TSSstackAddress + 3) > current TSS limit
        THEN #TS(error_code(current TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
      NewSS ← 2 bytes loaded from (TSS base + TSSstackAddress + 2);
      NewESP ← 2 bytes loaded from (TSS base + TSSstackAddress);
    FI;
  IF NewSS is NULL
    THEN #TS(EXT); FI;
  IF NewSS index is not within its descriptor-table limits
  or NewSS RPL ≠ new code-segment DPL
    THEN #TS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
  Read new stack-segment descriptor for NewSS in GDT or LDT;
  IF new stack-segment DPL ≠ new code-segment DPL
  or new stack-segment Type does not indicate writable data segment
    THEN #TS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
  IF NewSS is not present
    THEN #SS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
ELSE (* IA-32e mode *)
  IF IDT-gate IST = 0
    THEN TSSstackAddress ← (new code-segment DPL << 3) + 4;
    ELSE TSSstackAddress ← (IDT gate IST << 3) + 28;
  FI;
  IF (TSSstackAddress + 7) > current TSS limit
    THEN #TS(error_code(current TSS selector,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
  NewRSP ← 8 bytes loaded from (current TSS base + TSSstackAddress);
  NewSS ← new code-segment DPL; (* NULL selector with RPL = new CPL *)
FI;
IF IDT gate is 32-bit
  THEN
    IF new stack does not have room for 24 bytes (error code pushed)
    or 20 bytes (no error code pushed)
      THEN #SS(error_code(NewSS,0,EXT)); FI;
      (* idt operand to error_code is 0 because selector is used *)
    FI
  ELSE
    IF IDT gate is 16-bit
      THEN
        IF new stack does not have room for 12 bytes (error code pushed)

```

```

        or 10 bytes (no error code pushed);
        THEN #SS(error_code(NewSS,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
ELSE (* 64-bit IDT gate*)
    IF StackAddress is non-canonical
        THEN #SS(EXT); FI; (* Error code contains NULL selector *)
FI;
FI;
IF (IA32_EFER.LMA = 0) (* Not IA-32e mode *)
    THEN
        IF instruction pointer from IDT gate is not within new code-segment limits
            THEN #GP(EXT); FI; (* Error code contains NULL selector *)
        ESP ← NewESP;
        SS ← NewSS; (* Segment descriptor information also loaded *)
    ELSE (* IA-32e mode *)
        IF instruction pointer from IDT gate contains a non-canonical address
            THEN #GP(EXT); FI; (* Error code contains NULL selector *)
        RSP ← NewRSP & FFFFFFFF0H;
        SS ← NewSS;
FI;
IF IDT gate is 32-bit
    THEN
        CS:EIP ← Gate(CS:EIP); (* Segment descriptor information also loaded *)
    ELSE
        IF IDT gate 16-bit
            THEN
                CS:IP ← Gate(CS:IP);
                (* Segment descriptor information also loaded *)
            ELSE (* 64-bit IDT gate *)
                CS:RIP ← Gate(CS:RIP);
                (* Segment descriptor information also loaded *)
        FI;
FI;
IF IDT gate is 32-bit
    THEN
        Push(far pointer to old stack);
        (* Old SS and ESP, 3 words padded to 4 *)
        Push(EFLAGS);
        Push(far pointer to return instruction);
        (* Old CS and EIP, 3 words padded to 4 *)
        Push(ErrorCode); (* If needed, 4 bytes *)
    ELSE
        IF IDT gate 16-bit
            THEN
                Push(far pointer to old stack);
                (* Old SS and SP, 2 words *)
                Push(EFLAGS(15-0));
                Push(far pointer to return instruction);
                (* Old CS and IP, 2 words *)
                Push(ErrorCode); (* If needed, 2 bytes *)
            ELSE (* 64-bit IDT gate *)
                Push(far pointer to old stack);
                (* Old SS and SP, each an 8-byte push *)
                Push(RFLAGS); (* 8-byte push *)

```

```

        Push(far pointer to return instruction);
        (* Old CS and RIP, each an 8-byte push *)
        Push(ErrorCode); (* If needed, 8-bytes *)
    FI;
FI;
CPL ← new code-segment DPL;
CS(RPL) ← CPL;
IF IDT gate is interrupt gate
    THEN IF ← 0 (* Interrupt flag set to 0, interrupts disabled *); FI;
TF ← 0;
VM ← 0;
RF ← 0;
NT ← 0;
END;
INTERRUPT-FROM-VIRTUAL-8086-MODE:
(* Identify stack-segment selector for privilege level 0 in current TSS *)
IF current TSS is 32-bit
    THEN
        IF TSS limit < 9
            THEN #TS(error_code(current TSS selector,0,EXT)); FI;
            (* idt operand to error_code is 0 because selector is used *)
            NewSS ← 2 bytes loaded from (current TSS base + 8);
            NewESP ← 4 bytes loaded from (current TSS base + 4);
        ELSE (* current TSS is 16-bit *)
            IF TSS limit < 5
                THEN #TS(error_code(current TSS selector,0,EXT)); FI;
                (* idt operand to error_code is 0 because selector is used *)
                NewSS ← 2 bytes loaded from (current TSS base + 4);
                NewESP ← 2 bytes loaded from (current TSS base + 2);
        FI;
IF NewSS is NULL
    THEN #TS(EXT); FI; (* Error code contains NULL selector *)
IF NewSS index is not within its descriptor table limits
or NewSS RPL ≠ 0
    THEN #TS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
Read new stack-segment descriptor for NewSS in GDT or LDT;
IF new stack-segment DPL ≠ 0 or stack segment does not indicate writable data segment
    THEN #TS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
IF new stack segment not present
    THEN #SS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
IF IDT gate is 32-bit
    THEN
        IF new stack does not have room for 40 bytes (error code pushed)
        or 36 bytes (no error code pushed)
            THEN #SS(error_code(NewSS,0,EXT)); FI;
            (* idt operand to error_code is 0 because selector is used *)
        ELSE (* IDT gate is 16-bit *)
            IF new stack does not have room for 20 bytes (error code pushed)
            or 18 bytes (no error code pushed)
                THEN #SS(error_code(NewSS,0,EXT)); FI;
                (* idt operand to error_code is 0 because selector is used *)

```



```

FI;
IF instruction pointer from IDT gate is not within new code-segment limits
    THEN #GP(EXT); FI; (* Error code contains NULL selector *)
tempEFLAGS ← EFLAGS;
VM ← 0;
TF ← 0;
RF ← 0;
NT ← 0;
IF service through interrupt gate
    THEN IF = 0; FI;
TempSS ← SS;
TempESP ← ESP;
SS ← NewSS;
ESP ← NewESP;
(* Following pushes are 16 bits for 16-bit IDT gates and 32 bits for 32-bit IDT gates;
Segment selector pushes in 32-bit mode are padded to two words *)
Push(GS);
Push(FS);
Push(DS);
Push(ES);
Push(TempSS);
Push(TempESP);
Push(TempEFlags);
Push(CS);
Push(EIP);
GS ← 0; (* Segment registers made NULL, invalid for use in protected mode *)
FS ← 0;
DS ← 0;
ES ← 0;
CS ← Gate(CS); (* Segment descriptor information also loaded *)
CS(RPL) ← 0;
CPL ← 0;
IF IDT gate is 32-bit
    THEN
        EIP ← Gate(instruction pointer);
    ELSE (* IDT gate is 16-bit *)
        EIP ← Gate(instruction pointer) AND 0000FFFFH;
FI;
(* Start execution of new routine in Protected Mode *)
END;
INTRA-PRIVILEGE-LEVEL-INTERRUPT:
(* PE = 1, DPL = CPL or conforming segment *)
IF IA32_EFER.LMA = 1 (* IA-32e mode *)
    IF IDT-descriptor IST ≠ 0
        THEN
            TSSstackAddress ← (IDT-descriptor IST « 3) + 28;
            IF (TSSstackAddress + 7) > TSS limit
                THEN #TS(error_code(current TSS selector,0,EXT)); FI;
                (* idt operand to error_code is 0 because selector is used *)
                NewRSP ← 8 bytes loaded from (current TSS base + TSSstackAddress);
            ELSE NewRSP ← RSP;
        FI;
    FI;
    IF 32-bit gate (* implies IA32_EFER.LMA = 0 *)

```

```

THEN
    IF current stack does not have room for 16 bytes (error code pushed)
    or 12 bytes (no error code pushed)
        THEN #SS(EXT); FI; (* Error code contains NULL selector *)
    ELSE IF 16-bit gate (* implies IA32_EFER.LMA = 0 *)
        IF current stack does not have room for 8 bytes (error code pushed)
        or 6 bytes (no error code pushed)
            THEN #SS(EXT); FI; (* Error code contains NULL selector *)
        ELSE (* IA32_EFER.LMA = 1, 64-bit gate*)
            IF NewRSP contains a non-canonical address
                THEN #SS(EXT); (* Error code contains NULL selector *)
        FI;
    FI;
FI;
IF (IA32_EFER.LMA = 0) (* Not IA-32e mode *)
    THEN
        IF instruction pointer from IDT gate is not within new code-segment limit
            THEN #GP(EXT); FI; (* Error code contains NULL selector *)
        ELSE
            IF instruction pointer from IDT gate contains a non-canonical address
                THEN #GP(EXT); FI; (* Error code contains NULL selector *)
            RSP ← NewRSP & FFFFFFFFOH;
        FI;
    FI;
IF IDT gate is 32-bit (* implies IA32_EFER.LMA = 0 *)
    THEN
        Push (EFLAGS);
        Push (far pointer to return instruction); (* 3 words padded to 4 *)
        CS:EIP ← Gate(CS:EIP); (* Segment descriptor information also loaded *)
        Push (ErrorCode); (* If any *)
    ELSE
        IF IDT gate is 16-bit (* implies IA32_EFER.LMA = 0 *)
            THEN
                Push (FLAGS);
                Push (far pointer to return location); (* 2 words *)
                CS:IP ← Gate(CS:IP);
                (* Segment descriptor information also loaded *)
                Push (ErrorCode); (* If any *)
            ELSE (* IA32_EFER.LMA = 1, 64-bit gate*)
                Push(far pointer to old stack);
                (* Old SS and SP, each an 8-byte push *)
                Push(RFLAGS); (* 8-byte push *)
                Push(far pointer to return instruction);
                (* Old CS and RIP, each an 8-byte push *)
                Push(ErrorCode); (* If needed, 8 bytes *)
                CS:RIP ← GATE(CS:RIP);
                (* Segment descriptor information also loaded *)
            FI;
        FI;
    FI;
CS(RPL) ← CPL;
IF IDT gate is interrupt gate
    THEN IF ← 0; FI; (* Interrupt flag set to 0; interrupts disabled *)
TF ← 0;
NT ← 0;
VM ← 0;
RF ← 0;

```

END;

### Flags Affected

The EFLAGS register is pushed onto the stack. The IF, TF, NT, AC, RF, and VM flags may be cleared, depending on the mode of operation of the processor when the INT instruction is executed (see the “Operation” section). If the interrupt uses a task gate, any flags may be set or cleared, controlled by the EFLAGS image in the new task’s TSS.

### Protected Mode Exceptions

#GP(error_code)	<p>If the instruction pointer in the IDT or in the interrupt-, trap-, or task gate is beyond the code segment limits.</p> <p>If the segment selector in the interrupt-, trap-, or task gate is NULL.</p> <p>If an interrupt-, trap-, or task gate, code segment, or TSS segment selector index is outside its descriptor table limits.</p> <p>If the vector selects a descriptor outside the IDT limits.</p> <p>If an IDT descriptor is not an interrupt-, trap-, or task-descriptor.</p> <p>If an interrupt is generated by the INT <i>n</i>, INT 3, or INTO instruction and the DPL of an interrupt-, trap-, or task-descriptor is less than the CPL.</p> <p>If the segment selector in an interrupt- or trap-gate does not point to a segment descriptor for a code segment.</p> <p>If the segment selector for a TSS has its local/global bit set for local.</p> <p>If a TSS segment descriptor specifies that the TSS is busy or not available.</p>
#SS(error_code)	<p>If pushing the return address, flags, or error code onto the stack exceeds the bounds of the stack segment and no stack switch occurs.</p> <p>If the SS register is being loaded and the segment pointed to is marked not present.</p> <p>If pushing the return address, flags, error code, or stack segment pointer exceeds the bounds of the new stack segment when a stack switch occurs.</p>
#NP(error_code)	If code segment, interrupt-, trap-, or task gate, or TSS is not present.
#TS(error_code)	<p>If the RPL of the stack segment selector in the TSS is not equal to the DPL of the code segment being accessed by the interrupt or trap gate.</p> <p>If DPL of the stack segment descriptor pointed to by the stack segment selector in the TSS is not equal to the DPL of the code segment descriptor for the interrupt or trap gate.</p> <p>If the stack segment selector in the TSS is NULL.</p> <p>If the stack segment for the TSS is not a writable data segment.</p> <p>If segment-selector index for stack segment is outside descriptor table limits.</p>
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
#AC(EXT)	If alignment checking is enabled, the gate DPL is 3, and a stack push is unaligned.

### Real-Address Mode Exceptions

#GP	<p>If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.</p> <p>If the interrupt vector number is outside the IDT limits.</p>
#SS	<p>If stack limit violation on push.</p> <p>If pushing the return address, flags, or error code onto the stack exceeds the bounds of the stack segment.</p>
#UD	If the LOCK prefix is used.

**Virtual-8086 Mode Exceptions**

#GP(error_code)	<p>(For INT <i>n</i>, INTO, or BOUND instruction) If the IOPL is less than 3 or the DPL of the interrupt-, trap-, or task-gate descriptor is not equal to 3.</p> <p>If the instruction pointer in the IDT or in the interrupt-, trap-, or task gate is beyond the code segment limits.</p> <p>If the segment selector in the interrupt-, trap-, or task gate is NULL.</p> <p>If a interrupt-, trap-, or task gate, code segment, or TSS segment selector index is outside its descriptor table limits.</p> <p>If the vector selects a descriptor outside the IDT limits.</p> <p>If an IDT descriptor is not an interrupt-, trap-, or task-descriptor.</p> <p>If an interrupt is generated by the INT <i>n</i> instruction and the DPL of an interrupt-, trap-, or task-descriptor is less than the CPL.</p> <p>If the segment selector in an interrupt- or trap-gate does not point to a segment descriptor for a code segment.</p> <p>If the segment selector for a TSS has its local/global bit set for local.</p>
#SS(error_code)	<p>If the SS register is being loaded and the segment pointed to is marked not present.</p> <p>If pushing the return address, flags, error code, stack segment pointer, or data segments exceeds the bounds of the stack segment.</p>
#NP(error_code)	If code segment, interrupt-, trap-, or task gate, or TSS is not present.
#TS(error_code)	<p>If the RPL of the stack segment selector in the TSS is not equal to the DPL of the code segment being accessed by the interrupt or trap gate.</p> <p>If DPL of the stack segment descriptor for the TSS's stack segment is not equal to the DPL of the code segment descriptor for the interrupt or trap gate.</p> <p>If the stack segment selector in the TSS is NULL.</p> <p>If the stack segment for the TSS is not a writable data segment.</p> <p>If segment-selector index for stack segment is outside descriptor table limits.</p>
#PF(fault-code)	If a page fault occurs.
#BP	If the INT 3 instruction is executed.
#OF	If the INTO instruction is executed and the OF flag is set.
#UD	If the LOCK prefix is used.
#AC(EXT)	If alignment checking is enabled, the gate DPL is 3, and a stack push is unaligned.

**Compatibility Mode Exceptions**

Same exceptions as in protected mode.

## 64-Bit Mode Exceptions

#GP(error_code)	<p>If the instruction pointer in the 64-bit interrupt gate or 64-bit trap gate is non-canonical.</p> <p>If the segment selector in the 64-bit interrupt or trap gate is NULL.</p> <p>If the vector selects a descriptor outside the IDT limits.</p> <p>If the vector points to a gate which is in non-canonical space.</p> <p>If the vector points to a descriptor which is not a 64-bit interrupt gate or 64-bit trap gate.</p> <p>If the descriptor pointed to by the gate selector is outside the descriptor table limit.</p> <p>If the descriptor pointed to by the gate selector is in non-canonical space.</p> <p>If the descriptor pointed to by the gate selector is not a code segment.</p> <p>If the descriptor pointed to by the gate selector doesn't have the L-bit set, or has both the L-bit and D-bit set.</p> <p>If the descriptor pointed to by the gate selector has <math>DPL &gt; CPL</math>.</p>
#SS(error_code)	<p>If a push of the old EFLAGS, CS selector, EIP, or error code is in non-canonical space with no stack switch.</p> <p>If a push of the old SS selector, ESP, EFLAGS, CS selector, EIP, or error code is in non-canonical space on a stack switch (either CPL change or no-CPL with IST).</p>
#NP(error_code)	If the 64-bit interrupt-gate, 64-bit trap-gate, or code segment is not present.
#TS(error_code)	<p>If an attempt to load RSP from the TSS causes an access to non-canonical space.</p> <p>If the RSP from the TSS is outside descriptor table limits.</p>
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
#AC(EXT)	If alignment checking is enabled, the gate DPL is 3, and a stack push is unaligned.

## IRET/IRETD—Interrupt Return

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
CF	IRET	Z0	Valid	Valid	Interrupt return (16-bit operand size).
CF	IRETD	Z0	Valid	Valid	Interrupt return (32-bit operand size).
REX.W + CF	IRETQ	Z0	Valid	N.E.	Interrupt return (64-bit operand size).

### Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	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 preforms a far return to the interrupted program or procedure. During this operation, the processor pops the return instruction pointer, return code segment selector, and EFLAGS image from the stack to the EIP, CS, and EFLAGS registers, respectively, and then resumes execution of the interrupted program or procedure.

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

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

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

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

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

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

If nonmaskable interrupts (NMIs) are blocked (see Section 6.7.1, “Handling Multiple NMIs” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*), execution of the IRET instruction unblocks NMIs.

This unblocking occurs even if the instruction causes a fault. In such a case, NMIs are unmasked before the exception handler is invoked.

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

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;

ELSIF (IA32\_EFER.LMA = 0)

THEN

IF (EFLAGS.VM = 1)

THEN GOTO RETURN-FROM-VIRTUAL-8086-MODE;

ELSE GOTO PROTECTED-MODE;

FI;

ELSE GOTO IA-32e-MODE;

FI;

REAL-ADDRESS-MODE;

IF OperandSize = 32

THEN

EIP ← Pop();

CS ← Pop(); (\* 32-bit pop, high-order 16 bits discarded \*)

tempEFLAGS ← Pop();

EFLAGS ← (tempEFLAGS AND 257FD5H) OR (EFLAGS AND 1A0000H);

ELSE (\* OperandSize = 16 \*)

EIP ← Pop(); (\* 16-bit pop; clear upper 16 bits \*)

CS ← Pop(); (\* 16-bit pop \*)

EFLAGS[15:0] ← Pop();

FI;

END;

RETURN-FROM-VIRTUAL-8086-MODE:

(\* Processor is in virtual-8086 mode when IRET is executed and stays in virtual-8086 mode \*)

IF IOPL = 3 (\* Virtual mode: PE = 1, VM = 1, IOPL = 3 \*)

THEN IF OperandSize = 32

THEN

EIP ← Pop();

CS ← Pop(); (\* 32-bit pop, high-order 16 bits discarded \*)

EFLAGS ← Pop();

(\* VM, IOPL, VIP and VIF EFLAG bits not modified by pop \*)

IF EIP not within CS limit

THEN #GP(0); FI;

ELSE (\* OperandSize = 16 \*)

EIP ← Pop(); (\* 16-bit pop; clear upper 16 bits \*)

CS ← Pop(); (\* 16-bit pop \*)

EFLAGS[15:0] ← Pop(); (\* IOPL in EFLAGS not modified by pop \*)

IF EIP not within CS limit

THEN #GP(0); FI;

FI;

ELSE

```

        #GP(0); (* Trap to virtual-8086 monitor: PE = 1, VM = 1, IOPL < 3 *)
FI;
END;

PROTECTED-MODE:
  IF NT = 1
    THEN GOTO TASK-RETURN; (* PE = 1, VM = 0, NT = 1 *)
FI;
  IF OperandSize = 32
    THEN
      EIP ← Pop();
      CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
      tempEFLAGS ← Pop();
    ELSE (* OperandSize = 16 *)
      EIP ← Pop(); (* 16-bit pop; clear upper bits *)
      CS ← Pop(); (* 16-bit pop *)
      tempEFLAGS ← Pop(); (* 16-bit pop; clear upper bits *)
FI;
  IF tempEFLAGS(VM) = 1 and CPL = 0
    THEN GOTO RETURN-TO-VIRTUAL-8086-MODE;
    ELSE GOTO PROTECTED-MODE-RETURN;
FI;

TASK-RETURN: (* PE = 1, VM = 0, NT = 1 *)
  SWITCH-TASKS (without nesting) to TSS specified in link field of current TSS;
  Mark the task just abandoned as NOT BUSY;
  IF EIP is not within CS limit
    THEN #GP(0); FI;
END;

RETURN-TO-VIRTUAL-8086-MODE:
  (* Interrupted procedure was in virtual-8086 mode: PE = 1, CPL=0, VM = 1 in flag image *)
  IF EIP not within CS limit
    THEN #GP(0); FI;
  EFLAGS ← tempEFLAGS;
  ESP ← Pop();
  SS ← Pop(); (* Pop 2 words; throw away high-order word *)
  ES ← Pop(); (* Pop 2 words; throw away high-order word *)
  DS ← Pop(); (* Pop 2 words; throw away high-order word *)
  FS ← Pop(); (* Pop 2 words; throw away high-order word *)
  GS ← Pop(); (* Pop 2 words; throw away high-order word *)
  CPL ← 3;
  (* Resume execution in Virtual-8086 mode *)
END;

PROTECTED-MODE-RETURN: (* PE = 1 *)
  IF CS(RPL) > CPL
    THEN GOTO RETURN-TO-OUTER-PRIVILEGE-LEVEL;
    ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL; FI;
END;

RETURN-TO-OUTER-PRIVILEGE-LEVEL:
  IF OperandSize = 32
    THEN

```



```

    ESP ← Pop();
    SS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
ELSE IF OperandSize = 16
    THEN
    ESP ← Pop(); (* 16-bit pop; clear upper bits *)
    SS ← Pop(); (* 16-bit pop *)
ELSE (* OperandSize = 64 *)
    RSP ← Pop();
    SS ← Pop(); (* 64-bit pop, high-order 48 bits discarded *)
FI;
IF new mode ≠ 64-Bit Mode
    THEN
    IF EIP is not within CS limit
        THEN #GP(0); FI;
    ELSE (* new mode = 64-bit mode *)
        IF RIP is non-canonical
            THEN #GP(0); FI;
FI;
EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
IF OperandSize = 32 or OperandSize = 64
    THEN EFLAGS(RF, AC, ID) ← tempEFLAGS; FI;
IF CPL ≤ IOPL
    THEN EFLAGS(IF) ← tempEFLAGS; FI;
IF CPL = 0
    THEN
    EFLAGS(IOPL) ← tempEFLAGS;
    IF OperandSize = 32 or OperandSize = 64
        THEN EFLAGS(VIF, VIP) ← tempEFLAGS; FI;
FI;
CPL ← CS(RPL);
FOR each SegReg in (ES, FS, GS, and DS)
    DO
    tempDesc ← descriptor cache for SegReg (* hidden part of segment register *)
    IF (SegmentSelector == NULL) OR (tempDesc(DPL) < CPL AND tempDesc(Type) is (data or non-conforming code)))
        THEN (* Segment register invalid *)
            SegmentSelector ← 0; (*Segment selector becomes null*)
    FI;
OD;
END;

RETURN-TO-SAME-PRIVILEGE-LEVEL: (* PE = 1, RPL = CPL *)
IF new mode ≠ 64-Bit Mode
    THEN
    IF EIP is not within CS limit
        THEN #GP(0); FI;
    ELSE (* new mode = 64-bit mode *)
        IF RIP is non-canonical
            THEN #GP(0); FI;
FI;
EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
IF OperandSize = 32 or OperandSize = 64
    THEN EFLAGS(RF, AC, ID) ← tempEFLAGS; FI;
IF CPL ≤ IOPL
    THEN EFLAGS(IF) ← tempEFLAGS; FI;

```

```

IF CPL = 0
  THEN
    EFLAGS(IOPL) ← tempEFLAGS;
    IF OperandSize = 32 or OperandSize = 64
      THEN EFLAGS(VIF, VIP) ← tempEFLAGS; FI;
  FI;
END;

IA-32e-MODE:
IF NT = 1
  THEN #GP(0);
ELSE IF OperandSize = 32
  THEN
    EIP ← Pop();
    CS ← Pop();
    tempEFLAGS ← Pop();
  ELSE IF OperandSize = 16
    THEN
      EIP ← Pop(); (* 16-bit pop; clear upper bits *)
      CS ← Pop(); (* 16-bit pop *)
      tempEFLAGS ← Pop(); (* 16-bit pop; clear upper bits *)
    FI;
  ELSE (* OperandSize = 64 *)
    THEN
      RIP ← Pop();
      CS ← Pop(); (* 64-bit pop, high-order 48 bits discarded *)
      tempRFLAGS ← Pop();
    FI;
  IF CS.RPL > CPL
    THEN GOTO RETURN-TO-OUTER-PRIVILEGE-LEVEL;
  ELSE
    IF instruction began in 64-Bit Mode
      THEN
        IF OperandSize = 32
          THEN
            ESP ← Pop();
            SS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
          ELSE IF OperandSize = 16
            THEN
              ESP ← Pop(); (* 16-bit pop; clear upper bits *)
              SS ← Pop(); (* 16-bit pop *)
            ELSE (* OperandSize = 64 *)
              RSP ← Pop();
              SS ← Pop(); (* 64-bit pop, high-order 48 bits discarded *)
            FI;
          FI;
        GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL; FI;
  END;

```

### Flags Affected

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

**Protected Mode Exceptions**

#GP(0)	If the return code or stack segment selector is NULL. If the return instruction pointer is not within the return code segment limit.
#GP(selector)	If a segment selector index is outside its descriptor table limits. If the return code segment selector RPL is less than the CPL. If the DPL of a conforming-code segment is greater than the return code segment selector RPL. If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector. If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector. If the stack segment is not a writable data segment. If the stack segment selector RPL is not equal to the RPL of the return code segment selector. If the segment descriptor for a code segment does not indicate it is a code segment. If the segment selector for a TSS has its local/global bit set for local. If a TSS segment descriptor specifies that the TSS is not busy. If a TSS segment descriptor specifies that the TSS is not available.
#SS(0)	If the top bytes of stack are not within stack limits. If the return stack segment is not present.
#NP (selector)	If the return code segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.
#UD	If the LOCK prefix is used.

**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)	<p>If EFLAGS.NT[bit 14] = 1.</p> <p>If the return code segment selector is NULL.</p> <p>If the stack segment selector is NULL going back to compatibility mode.</p> <p>If the stack segment selector is NULL going back to CPL3 64-bit mode.</p> <p>If a NULL stack segment selector RPL is not equal to CPL going back to non-CPL3 64-bit mode.</p> <p>If the return instruction pointer is not within the return code segment limit.</p> <p>If the return instruction pointer is non-canonical.</p>
#GP(Selector)	<p>If a segment selector index is outside its descriptor table limits.</p> <p>If a segment descriptor memory address is non-canonical.</p> <p>If the segment descriptor for a code segment does not indicate it is a code segment.</p> <p>If the proposed new code segment descriptor has both the D-bit and L-bit set.</p> <p>If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector.</p> <p>If CPL is greater than the RPL of the code segment selector.</p> <p>If the DPL of a conforming-code segment is greater than the return code segment selector RPL.</p> <p>If the stack segment is not a writable data segment.</p> <p>If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector.</p> <p>If the stack segment selector RPL is not equal to the RPL of the return code segment selector.</p>
#SS(0)	<p>If an attempt to pop a value off the stack violates the SS limit.</p> <p>If an attempt to pop a value off the stack causes a non-canonical address to be referenced.</p> <p>If the return stack segment is not present.</p>
#NP (selector)	If the return code segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.
#UD	If the LOCK prefix is used.

## KMOVW/KMOVB/KMOVQ/KMOVD—Move from and to Mask Registers

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.L0.0F.W0 90 /r KMOVW k1, k2/m16	RM	V/V	AVX512F	Move 16 bits mask from k2/m16 and store the result in k1.
VEX.L0.66.0F.W0 90 /r KMOVB k1, k2/m8	RM	V/V	AVX512DQ	Move 8 bits mask from k2/m8 and store the result in k1.
VEX.L0.0F.W1 90 /r KMOVQ k1, k2/m64	RM	V/V	AVX512BW	Move 64 bits mask from k2/m64 and store the result in k1.
VEX.L0.66.0F.W1 90 /r KMOVD k1, k2/m32	RM	V/V	AVX512BW	Move 32 bits mask from k2/m32 and store the result in k1.
VEX.L0.0F.W0 91 /r KMOVW m16, k1	MR	V/V	AVX512F	Move 16 bits mask from k1 and store the result in m16.
VEX.L0.66.0F.W0 91 /r KMOVB m8, k1	MR	V/V	AVX512DQ	Move 8 bits mask from k1 and store the result in m8.
VEX.L0.0F.W1 91 /r KMOVQ m64, k1	MR	V/V	AVX512BW	Move 64 bits mask from k1 and store the result in m64.
VEX.L0.66.0F.W1 91 /r KMOVD m32, k1	MR	V/V	AVX512BW	Move 32 bits mask from k1 and store the result in m32.
VEX.L0.0F.W0 92 /r KMOVW k1, r32	RR	V/V	AVX512F	Move 16 bits mask from r32 to k1.
VEX.L0.66.0F.W0 92 /r KMOVB k1, r32	RR	V/V	AVX512DQ	Move 8 bits mask from r32 to k1.
VEX.L0.F2.0F.W1 92 /r KMOVQ k1, r64	RR	V/I	AVX512BW	Move 64 bits mask from r64 to k1.
VEX.L0.F2.0F.W0 92 /r KMOVD k1, r32	RR	V/V	AVX512BW	Move 32 bits mask from r32 to k1.
VEX.L0.0F.W0 93 /r KMOVW r32, k1	RR	V/V	AVX512F	Move 16 bits mask from k1 to r32.
VEX.L0.66.0F.W0 93 /r KMOVB r32, k1	RR	V/V	AVX512DQ	Move 8 bits mask from k1 to r32.
VEX.L0.F2.0F.W1 93 /r KMOVQ r64, k1	RR	V/I	AVX512BW	Move 64 bits mask from k1 to r64.
VEX.L0.F2.0F.W0 93 /r KMOVD r32, k1	RR	V/V	AVX512BW	Move 32 bits mask from k1 to r32.

### Instruction Operand Encoding

Op/En	Operand 1	Operand 2
RM	ModRM:reg (w)	ModRM:r/m (r)
MR	ModRM:r/m (w, ModRM:[7:6] must not be 11b)	ModRM:reg (r)
RR	ModRM:reg (w)	ModRM:r/m (r, ModRM:[7:6] must be 11b)

### Description

Copies values from the source operand (second operand) to the destination operand (first operand). The source and destination operands can be mask registers, memory location or general purpose. The instruction cannot be used to transfer data between general purpose registers and or memory locations.

When moving to a mask register, the result is zero extended to MAX\_KL size (i.e., 64 bits currently). When moving to a general-purpose register (GPR), the result is zero-extended to the size of the destination. In 32-bit mode, the default GPR destination's size is 32 bits. In 64-bit mode, the default GPR destination's size is 64 bits. Note that VEX.W can only be used to modify the size of the GPR operand in 64b mode.

## Operation

### KMOVW

IF \*destination is a memory location\*

$DEST[15:0] \leftarrow SRC[15:0]$

IF \*destination is a mask register or a GPR \*

$DEST \leftarrow ZeroExtension(SRC[15:0])$

### KMOVB

IF \*destination is a memory location\*

$DEST[7:0] \leftarrow SRC[7:0]$

IF \*destination is a mask register or a GPR \*

$DEST \leftarrow ZeroExtension(SRC[7:0])$

### KMOVQ

IF \*destination is a memory location or a GPR\*

$DEST[63:0] \leftarrow SRC[63:0]$

IF \*destination is a mask register\*

$DEST \leftarrow ZeroExtension(SRC[63:0])$

### KMOVD

IF \*destination is a memory location\*

$DEST[31:0] \leftarrow SRC[31:0]$

IF \*destination is a mask register or a GPR \*

$DEST \leftarrow ZeroExtension(SRC[31:0])$

## Intel C/C++ Compiler Intrinsic Equivalent

KMOVW `__mmask16 _mm512_kmov(__mmask16 a);`

## Flags Affected

None

## SIMD Floating-Point Exceptions

None

## Other Exceptions

Instructions with RR operand encoding See Exceptions Type K20.

Instructions with RM or MR operand encoding See Exceptions Type K21.

## LSL—Load Segment Limit

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 03 /r	LSL r16, r16/m16	RM	Valid	Valid	Load: r16 ← segment limit, selector r16/m16.
OF 03 /r	LSL r32, r32/m16*	RM	Valid	Valid	Load: r32 ← segment limit, selector r32/m16.
REX.W + OF 03 /r	LSL r64, r32/m16*	RM	Valid	Valid	Load: r64 ← segment limit, selector r32/m16

### NOTES:

\* For all loads (regardless of destination sizing), only bits 16-0 are used. Other bits are ignored.

### Instruction Operand Encoding

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

### Description

Loads the unscrambled segment limit from the segment descriptor specified with the second operand (source operand) into the first operand (destination operand) and sets the ZF flag in the EFLAGS register. The source operand (which can be a register or a memory location) contains the segment selector for the segment descriptor being accessed. The destination operand is a general-purpose register.

The processor performs access checks as part of the loading process. Once loaded in the destination register, software can compare the segment limit with the offset of a pointer.

The segment limit is a 20-bit value contained in bytes 0 and 1 and in the first 4 bits of byte 6 of the segment descriptor. If the descriptor has a byte granular segment limit (the granularity flag is set to 0), the destination operand is loaded with a byte granular value (byte limit). If the descriptor has a page granular segment limit (the granularity flag is set to 1), the LSL instruction will translate the page granular limit (page limit) into a byte limit before loading it into the destination operand. The translation is performed by shifting the 20-bit “raw” limit left 12 bits and filling the low-order 12 bits with 1s.

When the operand size is 32 bits, the 32-bit byte limit is stored in the destination operand. When the operand size is 16 bits, a valid 32-bit limit is computed; however, the upper 16 bits are truncated and only the low-order 16 bits are loaded into the destination operand.

This instruction performs the following checks before it loads the segment limit into the destination register:

- Checks that the segment selector is not NULL.
- Checks that the segment selector points to a descriptor that is within the limits of the GDT or LDT being accessed
- Checks that the descriptor type is valid for this instruction. All code and data segment descriptors are valid for (can be accessed with) the LSL instruction. The valid special segment and gate descriptor types are given in the following table.
- If the segment is not a conforming code segment, the instruction checks that the specified segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector are less than or equal to the DPL of the segment selector).

If the segment descriptor cannot be accessed or is an invalid type for the instruction, the ZF flag is cleared and no value is loaded in the destination operand.

Table 3-55. Segment and Gate Descriptor Types

Type	Protected Mode		IA-32e Mode	
	Name	Valid	Name	Valid
0	Reserved	No	Reserved	No
1	Available 16-bit TSS	Yes	Reserved	No
2	LDT	Yes	LDT <sup>1</sup>	Yes
3	Busy 16-bit TSS	Yes	Reserved	No
4	16-bit call gate	No	Reserved	No
5	16-bit/32-bit task gate	No	Reserved	No
6	16-bit interrupt gate	No	Reserved	No
7	16-bit trap gate	No	Reserved	No
8	Reserved	No	Reserved	No
9	Available 32-bit TSS	Yes	64-bit TSS <sup>1</sup>	Yes
A	Reserved	No	Reserved	No
B	Busy 32-bit TSS	Yes	Busy 64-bit TSS <sup>1</sup>	Yes
C	32-bit call gate	No	64-bit call gate	No
D	Reserved	No	Reserved	No
E	32-bit interrupt gate	No	64-bit interrupt gate	No
F	32-bit trap gate	No	64-bit trap gate	No

**NOTES:**

1. In this case, the descriptor comprises 16 bytes; bits 12:8 of the upper 4 bytes must be 0.

**Operation**

IF SRC(Offset) > descriptor table limit  
THEN ZF ← 0; FI;

Read segment descriptor;

IF SegmentDescriptor(Type) ≠ conforming code segment  
and (CPL > DPL) OR (RPL > DPL)  
or Segment type is not valid for instruction

THEN

ZF ← 0;

ELSE

temp ← SegmentLimit([SRC]);

IF (G ← 1)

THEN temp ← ShiftLeft(12, temp) OR 00000FFFH;

ELSE IF OperandSize = 32

THEN DEST ← temp; FI;

ELSE IF OperandSize = 64 (\* REX.W used \*)

THEN DEST (\* Zero-extended \*) ← temp; FI;

ELSE (\* OperandSize = 16 \*)

DEST ← temp AND FFFFH;

FI;

FI;



## Flags Affected

The ZF flag is set to 1 if the segment limit is loaded successfully; otherwise, it is set to 0.

## Protected Mode Exceptions

#GP(0)	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 the memory operand effective address is unaligned while the current privilege level is 3.
#UD	If the LOCK prefix is used.

## Real-Address Mode Exceptions

#UD	The LSL instruction cannot be executed in real-address mode.
-----	--

## Virtual-8086 Mode Exceptions

#UD	The LSL instruction cannot be executed in virtual-8086 mode.
-----	--

## Compatibility Mode Exceptions

Same exceptions as in protected mode.

## 64-Bit Mode Exceptions

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

## 6. 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-U*.

-----  
Change to this chapter: Updates to tuple type naming throughout chapter; these updates are not covered in this document as they are considered minor in nature and extensive in page count. See changes listed in chapter 2 of Volume 2A for details on the minor adjustments to naming.

Updates to the following instructions are covered here with change bars: MOV, MOVSX/MOVSXD, PHMINPOSUW, SETcc, SFENCE, SWAPGS, and UD.

Moved PREFETCHWT1 to chapter 7.

## 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 <sup>***</sup> ,r8 <sup>***</sup>	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 <sup>***</sup> ,r/m8 <sup>***</sup>	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 <sup>**</sup>	MR	Valid	Valid	Move segment register to r/m16.
REX.W + 8C /r	MOV r16/r32/m16, Sreg <sup>**</sup>	MR	Valid	Valid	Move zero extended 16-bit segment register to r16/r32/r64/m16.
REX.W + 8C /r	MOV r64/m16, Sreg <sup>**</sup>	MR	Valid	Valid	Move zero extended 16-bit segment register to r64/m16.
8E /r	MOV Sreg,r/m16 <sup>**</sup>	RM	Valid	Valid	Move r/m16 to segment register.
REX.W + 8E /r	MOV Sreg,r/m64 <sup>**</sup>	RM	Valid	Valid	Move lower 16 bits of r/m64 to segment register.
A0	MOV AL,moffs8 <sup>*</sup>	FD	Valid	Valid	Move byte at (seg:offset) to AL.
REX.W + A0	MOV AL,moffs8 <sup>*</sup>	FD	Valid	N.E.	Move byte at (offset) to AL.
A1	MOV AX,moffs16 <sup>*</sup>	FD	Valid	Valid	Move word at (seg:offset) to AX.
A1	MOV EAX,moffs32 <sup>*</sup>	FD	Valid	Valid	Move doubleword at (seg:offset) to EAX.
REX.W + A1	MOV RAX,moffs64 <sup>*</sup>	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 <sup>***</sup> ,AL	TD	Valid	N.E.	Move AL to (offset).
A3	MOV moffs16 <sup>*</sup> ,AX	TD	Valid	Valid	Move AX to (seg:offset).
A3	MOV moffs32 <sup>*</sup> ,EAX	TD	Valid	Valid	Move EAX to (seg:offset).
REX.W + A3	MOV moffs64 <sup>*</sup> ,RAX	TD	Valid	N.E.	Move RAX to (offset).
B0+ rb ib	MOV r8,imm8	OI	Valid	Valid	Move imm8 to r8.
REX + B0+ rb ib	MOV r8 <sup>***</sup> ,imm8	OI	Valid	N.E.	Move imm8 to r8.
B8+ rw iw	MOV r16,imm16	OI	Valid	Valid	Move imm16 to r16.
B8+ rd id	MOV r32,imm32	OI	Valid	Valid	Move imm32 to r32.
REX.W + B8+ rd io	MOV r64,imm64	OI	Valid	N.E.	Move imm64 to r64.
C6 /O ib	MOV r/m8,imm8	MI	Valid	Valid	Move imm8 to r/m8.
REX + C6 /O ib	MOV r/m8 <sup>***</sup> ,imm8	MI	Valid	N.E.	Move imm8 to r/m8.
C7 /O iw	MOV r/m16,imm16	MI	Valid	Valid	Move imm16 to r/m16.
C7 /O id	MOV r/m32,imm32	MI	Valid	Valid	Move imm32 to r/m32.
REX.W + C7 /O id	MOV r/m64,imm32	MI	Valid	N.E.	Move imm32 sign extended to 64-bits to r/m64.

**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 information 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 occurs<sup>1</sup>. Be aware that the LSS instruction offers a more efficient method of loading the SS and ESP registers.

When executing MOV Reg, Sreg, the processor copies the content of Sreg to the 16 least significant bits of the general-purpose register. The upper bits of the destination register are zero for most IA-32 processors (Pentium

- 
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
```

Pro processors and later) and all Intel 64 processors, with the exception that bits 31:16 are undefined for Intel Quark X1000 processors, Pentium and earlier processors.

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
      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.

**64-Bit Mode Exceptions**

#GP(0)	<p>If the memory address is in a non-canonical form.</p> <p>If an attempt is made to load SS register with NULL segment selector when CPL = 3.</p> <p>If an attempt is made to load SS register with NULL segment selector when CPL &lt; 3 and CPL ≠ RPL.</p>
#GP(selector)	<p>If segment selector index is outside descriptor table limits.</p> <p>If the memory access to the descriptor table is non-canonical.</p> <p>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.</p> <p>If the SS register is being loaded and the segment pointed to is a nonwritable data segment.</p> <p>If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.</p> <p>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.</p>
#SS(0)	<p>If the stack address is in a non-canonical form.</p>
#SS(selector)	<p>If the SS register is being loaded and the segment pointed to is marked not present.</p>
#PF(fault-code)	<p>If a page fault occurs.</p>
#AC(0)	<p>If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.</p>
#UD	<p>If attempt is made to load the CS register.</p> <p>If the LOCK prefix is used.</p>

## MOVSX/MOVSXD—Move with Sign-Extension

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF BE /r	MOVSX <i>r16, r/m8</i>	RM	Valid	Valid	Move byte to word with sign-extension.
OF BE /r	MOVSX <i>r32, r/m8</i>	RM	Valid	Valid	Move byte to doubleword with sign-extension.
REX.W + OF BE /r	MOVSX <i>r64, r/m8</i>	RM	Valid	N.E.	Move byte to quadword with sign-extension.
OF BF /r	MOVSX <i>r32, r/m16</i>	RM	Valid	Valid	Move word to doubleword, with sign-extension.
REX.W + OF BF /r	MOVSX <i>r64, r/m16</i>	RM	Valid	N.E.	Move word to quadword with sign-extension.
63 /r*	MOVSXD <i>r16, r/m16</i>	RM	Valid	Valid	Move word to word with sign-extension.
63 /r*	MOVSXD <i>r32, r/m32</i>	RM	Valid	Valid	Move doubleword to doubleword with sign-extension.
REX.W + 63 /r	MOVSXD <i>r64, r/m32</i>	RM	Valid	N.E.	Move doubleword to quadword with sign-extension.

### NOTES:

\* The use of MOVSXD without REX.W in 64-bit mode is discouraged. Regular MOV should be used instead of using MOVSXD without REX.W.

### Instruction Operand Encoding

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

### Description

Copies the contents of the source operand (register or memory location) to the destination operand (register) and sign extends the value to 16 or 32 bits (see Figure 7-6 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*). The size of the converted value depends on the operand-size attribute.

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 ← SignExtend(SRC);

### Flags Affected

None.

### Protected Mode Exceptions

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



**Real-Address Mode Exceptions**

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

**Virtual-8086 Mode Exceptions**

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.

**Compatibility Mode Exceptions**

Same exceptions as in protected mode.

**64-Bit Mode Exceptions**

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

## PHMINPOSUW — Packed Horizontal Word Minimum

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 38 41 /r PHMINPOSUW <i>xmm1, xmm2/m128</i>	RM	V/V	SSE4_1	Find the minimum unsigned word in <i>xmm2/m128</i> and place its value in the low word of <i>xmm1</i> and its index in the second-lowest word of <i>xmm1</i> .
VEX.128.66.0F38.WIG 41 /r VPHMINPOSUW <i>xmm1, xmm2/m128</i>	RM	V/V	AVX	Find the minimum unsigned word in <i>xmm2/m128</i> and place its value in the low word of <i>xmm1</i> and its index in the second-lowest word of <i>xmm1</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

Determine the minimum unsigned word value in the source operand (second operand) and place the unsigned word in the low word (bits 0-15) of the destination operand (first operand). The word index of the minimum value is stored in bits 16-18 of the destination operand. The remaining upper bits of the destination are set to zero.

- 128-bit Legacy SSE version: Bits (MAXVL-1:128) of the corresponding XMM destination register remain unchanged.
- VEX.128 encoded version: Bits (MAXVL-1:128) of the destination XMM register are zeroed. VEX.vvvv is reserved and must be 1111b, VEX.L must be 0, otherwise the instruction will #UD.

### Operation

#### PHMINPOSUW (128-bit Legacy SSE version)

```

INDEX ← 0;
MIN ← SRC[15:0]
IF (SRC[31:16] < MIN)
    THEN INDEX ← 1; MIN ← SRC[31:16]; FI;
IF (SRC[47:32] < MIN)
    THEN INDEX ← 2; MIN ← SRC[47:32]; FI;
* Repeat operation for words 3 through 6
IF (SRC[127:112] < MIN)
    THEN INDEX ← 7; MIN ← SRC[127:112]; FI;
DEST[15:0] ← MIN;
DEST[18:16] ← INDEX;
DEST[127:19] ← 00000000000000000000000000000000H;

```

**VPHMINPOSUW (VEX.128 encoded version)**

```

INDEX ← 0
MIN ← SRC[15:0]
IF (SRC[31:16] < MIN) THEN INDEX ← 1; MIN ← SRC[31:16]
IF (SRC[47:32] < MIN) THEN INDEX ← 2; MIN ← SRC[47:32]
* Repeat operation for words 3 through 6
IF (SRC[127:112] < MIN) THEN INDEX ← 7; MIN ← SRC[127:112]
DEST[15:0] ← MIN
DEST[18:16] ← INDEX
DEST[127:19] ← 00000000000000000000000000000000H
DEST[MAXVL-1:128] ← 0

```

**Intel C/C++ Compiler Intrinsic Equivalent**

```
PHMINPOSUW:    __m128i _mm_minpos_epu16(__m128i packed_words);
```

**Flags Affected**

None.

**SIMD Floating-Point Exceptions**

None.

**Other Exceptions**

See Exceptions Type 4; additionally

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

## SETcc—Set Byte on Condition

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F 97	SETA <i>r/m8</i>	M	Valid	Valid	Set byte if above (CF=0 and ZF=0).
REX + 0F 97	SETA <i>r/m8</i> *	M	Valid	N.E.	Set byte if above (CF=0 and ZF=0).
0F 93	SETAE <i>r/m8</i>	M	Valid	Valid	Set byte if above or equal (CF=0).
REX + 0F 93	SETAE <i>r/m8</i> *	M	Valid	N.E.	Set byte if above or equal (CF=0).
0F 92	SETB <i>r/m8</i>	M	Valid	Valid	Set byte if below (CF=1).
REX + 0F 92	SETB <i>r/m8</i> *	M	Valid	N.E.	Set byte if below (CF=1).
0F 96	SETBE <i>r/m8</i>	M	Valid	Valid	Set byte if below or equal (CF=1 or ZF=1).
REX + 0F 96	SETBE <i>r/m8</i> *	M	Valid	N.E.	Set byte if below or equal (CF=1 or ZF=1).
0F 92	SETC <i>r/m8</i>	M	Valid	Valid	Set byte if carry (CF=1).
REX + 0F 92	SETC <i>r/m8</i> *	M	Valid	N.E.	Set byte if carry (CF=1).
0F 94	SETE <i>r/m8</i>	M	Valid	Valid	Set byte if equal (ZF=1).
REX + 0F 94	SETE <i>r/m8</i> *	M	Valid	N.E.	Set byte if equal (ZF=1).
0F 9F	SETG <i>r/m8</i>	M	Valid	Valid	Set byte if greater (ZF=0 and SF=0F).
REX + 0F 9F	SETG <i>r/m8</i> *	M	Valid	N.E.	Set byte if greater (ZF=0 and SF=0F).
0F 9D	SETGE <i>r/m8</i>	M	Valid	Valid	Set byte if greater or equal (SF=0F).
REX + 0F 9D	SETGE <i>r/m8</i> *	M	Valid	N.E.	Set byte if greater or equal (SF=0F).
0F 9C	SETL <i>r/m8</i>	M	Valid	Valid	Set byte if less (SF≠ 0F).
REX + 0F 9C	SETL <i>r/m8</i> *	M	Valid	N.E.	Set byte if less (SF≠ 0F).
0F 9E	SETLE <i>r/m8</i>	M	Valid	Valid	Set byte if less or equal (ZF=1 or SF≠ 0F).
REX + 0F 9E	SETLE <i>r/m8</i> *	M	Valid	N.E.	Set byte if less or equal (ZF=1 or SF≠ 0F).
0F 96	SETNA <i>r/m8</i>	M	Valid	Valid	Set byte if not above (CF=1 or ZF=1).
REX + 0F 96	SETNA <i>r/m8</i> *	M	Valid	N.E.	Set byte if not above (CF=1 or ZF=1).
0F 92	SETNAE <i>r/m8</i>	M	Valid	Valid	Set byte if not above or equal (CF=1).
REX + 0F 92	SETNAE <i>r/m8</i> *	M	Valid	N.E.	Set byte if not above or equal (CF=1).
0F 93	SETNB <i>r/m8</i>	M	Valid	Valid	Set byte if not below (CF=0).
REX + 0F 93	SETNB <i>r/m8</i> *	M	Valid	N.E.	Set byte if not below (CF=0).
0F 97	SETNBE <i>r/m8</i>	M	Valid	Valid	Set byte if not below or equal (CF=0 and ZF=0).
REX + 0F 97	SETNBE <i>r/m8</i> *	M	Valid	N.E.	Set byte if not below or equal (CF=0 and ZF=0).
0F 93	SETNC <i>r/m8</i>	M	Valid	Valid	Set byte if not carry (CF=0).
REX + 0F 93	SETNC <i>r/m8</i> *	M	Valid	N.E.	Set byte if not carry (CF=0).
0F 95	SETNE <i>r/m8</i>	M	Valid	Valid	Set byte if not equal (ZF=0).
REX + 0F 95	SETNE <i>r/m8</i> *	M	Valid	N.E.	Set byte if not equal (ZF=0).
0F 9E	SETNG <i>r/m8</i>	M	Valid	Valid	Set byte if not greater (ZF=1 or SF≠ 0F)
REX + 0F 9E	SETNG <i>r/m8</i> *	M	Valid	N.E.	Set byte if not greater (ZF=1 or SF≠ 0F).
0F 9C	SETNGE <i>r/m8</i>	M	Valid	Valid	Set byte if not greater or equal (SF≠ 0F).
REX + 0F 9C	SETNGE <i>r/m8</i> *	M	Valid	N.E.	Set byte if not greater or equal (SF≠ 0F).
0F 9D	SETNL <i>r/m8</i>	M	Valid	Valid	Set byte if not less (SF=0F).
REX + 0F 9D	SETNL <i>r/m8</i> *	M	Valid	N.E.	Set byte if not less (SF=0F).
0F 9F	SETNLE <i>r/m8</i>	M	Valid	Valid	Set byte if not less or equal (ZF=0 and SF=0F).

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
REX + 0F 9F	SETNLE <i>r/m8</i> *	M	Valid	N.E.	Set byte if not less or equal (ZF=0 and SF=OF).
0F 91	SETNO <i>r/m8</i>	M	Valid	Valid	Set byte if not overflow (OF=0).
REX + 0F 91	SETNO <i>r/m8</i> *	M	Valid	N.E.	Set byte if not overflow (OF=0).
0F 9B	SETNP <i>r/m8</i>	M	Valid	Valid	Set byte if not parity (PF=0).
REX + 0F 9B	SETNP <i>r/m8</i> *	M	Valid	N.E.	Set byte if not parity (PF=0).
0F 99	SETNS <i>r/m8</i>	M	Valid	Valid	Set byte if not sign (SF=0).
REX + 0F 99	SETNS <i>r/m8</i> *	M	Valid	N.E.	Set byte if not sign (SF=0).
0F 95	SETNZ <i>r/m8</i>	M	Valid	Valid	Set byte if not zero (ZF=0).
REX + 0F 95	SETNZ <i>r/m8</i> *	M	Valid	N.E.	Set byte if not zero (ZF=0).
0F 90	SETO <i>r/m8</i>	M	Valid	Valid	Set byte if overflow (OF=1)
REX + 0F 90	SETO <i>r/m8</i> *	M	Valid	N.E.	Set byte if overflow (OF=1).
0F 9A	SETP <i>r/m8</i>	M	Valid	Valid	Set byte if parity (PF=1).
REX + 0F 9A	SETP <i>r/m8</i> *	M	Valid	N.E.	Set byte if parity (PF=1).
0F 9A	SETPE <i>r/m8</i>	M	Valid	Valid	Set byte if parity even (PF=1).
REX + 0F 9A	SETPE <i>r/m8</i> *	M	Valid	N.E.	Set byte if parity even (PF=1).
0F 9B	SETPO <i>r/m8</i>	M	Valid	Valid	Set byte if parity odd (PF=0).
REX + 0F 9B	SETPO <i>r/m8</i> *	M	Valid	N.E.	Set byte if parity odd (PF=0).
0F 98	SETS <i>r/m8</i>	M	Valid	Valid	Set byte if sign (SF=1).
REX + 0F 98	SETS <i>r/m8</i> *	M	Valid	N.E.	Set byte if sign (SF=1).
0F 94	SETZ <i>r/m8</i>	M	Valid	Valid	Set byte if zero (ZF=1).
REX + 0F 94	SETZ <i>r/m8</i> *	M	Valid	N.E.	Set byte if zero (ZF=1).

**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
M	ModRM:r/m (r)	NA	NA	NA

### Description

Sets the destination operand to 0 or 1 depending on the settings of the status flags (CF, SF, OF, ZF, and PF) in the EFLAGS register. The destination operand points to a byte register or a byte in memory. The condition code suffix (*cc*) indicates the condition being tested for.

The terms “above” and “below” are associated with the CF flag and refer to the relationship between two unsigned integer values. The terms “greater” and “less” are associated with the SF and OF flags and refer to the relationship between two signed integer values.

Many of the SET<sub>cc</sub> instruction opcodes have alternate mnemonics. For example, SETG (set byte if greater) and SETNLE (set if not less or equal) have the same opcode and test for the same condition: ZF equals 0 and SF equals OF. These alternate mnemonics are provided to make code more intelligible. Appendix B, “EFLAGS Condition Codes,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*, shows the alternate mnemonics for various test conditions.

Some languages represent a logical one as an integer with all bits set. This representation can be obtained by choosing the logically opposite condition for the SET<sub>cc</sub> instruction, then decrementing the result. For example, to test for overflow, use the SETNO instruction, then decrement the result.

The reg field of the ModR/M byte is not used for the SETCC instruction and those opcode bits are ignored by the processor.

In IA-64 mode, the operand size is fixed at 8 bits. Use of REX prefix enable uniform addressing to additional byte registers. Otherwise, this instruction's operation is the same as in legacy mode and compatibility mode.

## Operation

IF condition

THEN DEST ← 1;

ELSE DEST ← 0;

FI;

## Flags Affected

None.

## 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.
#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.
#UD	If the LOCK prefix is used.

## Compatibility Mode Exceptions

Same exceptions as in protected mode.

## 64-Bit Mode Exceptions

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

## SFENCE—Store Fence

Opcode*	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
NP OF AE F8	SFENCE	Z0	Valid	Valid	Serializes store operations.

### Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	NA	NA	NA	NA

### Description

Orders processor execution relative to all memory stores prior the SFENCE instruction. The processor ensures that every store prior to SFENCE is globally visible before any store after SFENCE becomes globally visible. The SFENCE instruction is ordered with respect to memory stores, other SFENCE instructions, MFENCE instructions, and any serializing instructions (such as the CPUID instruction). It is not ordered with respect to memory loads or the LFENCE instruction.

Weakly ordered memory types can be used to achieve higher processor performance through such techniques as out-of-order issue, write-combining, and write-collapsing. The degree to which a consumer of data recognizes or knows that the data is weakly ordered varies among applications and may be unknown to the producer of this data. The SFENCE instruction provides a performance-efficient way of ensuring store ordering between routines that produce weakly-ordered results and routines that consume this data.

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

Specification of the instruction's opcode above indicates a ModR/M byte of F8. For this instruction, the processor ignores the r/m field of the ModR/M byte. Thus, SFENCE is encoded by any opcode of the form OF AE Fx, where x is in the range 8-F.

### Operation

Wait\_On\_Following\_Stores\_Until(preceding\_stores\_globally\_visible);

### Intel C/C++ Compiler Intrinsic Equivalent

```
void _mm_sfence(void)
```

### Exceptions (All Operating Modes)

#UD                    If CPUID.01H:EDX.SSE[bit 25] = 0.  
                           If the LOCK prefix is used.

## SWAPGS—Swap GS Base Register

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 01 F8	SWAPGS	Z0	Valid	Invalid	Exchanges the current GS base register value with the value contained in MSR address C0000102H.

### Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	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.

#UD If the LOCK prefix is used.

## UD—Undefined Instruction

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF FF /r	<sup>1</sup> <i>r32, r/m32</i>	RM	Valid	Valid	Raise invalid opcode exception.
OF B9 /r	UD1 <i>r32, r/m32</i>	RM	Valid	Valid	Raise invalid opcode exception.
OF 0B	UD2	ZO	Valid	Valid	Raise invalid opcode exception.

### NOTES:

- Some older processors decode the UD0 instruction without a ModR/M byte. As a result, those processors would deliver an invalid-opcode exception instead of a fault on instruction fetch when the instruction with a ModR/M byte (and any implied bytes) would cross a page or segment boundary.

### Instruction Operand Encoding

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

### Description

Generates an invalid opcode exception. This instruction is provided for software testing to explicitly generate an invalid opcode exception. The opcodes for this instruction are reserved for this purpose.

Other than raising the invalid opcode exception, this instruction has no effect on processor state or memory.

Even though it is the execution of the UD instruction that causes the invalid opcode exception, the instruction pointer saved by delivery of the exception references the UD instruction (and not the following instruction).

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

### Operation

#UD (\* Generates invalid opcode exception \*);

### Flags Affected

None.

### Exceptions (All Operating Modes)

#UD                      Raises an invalid opcode exception in all operating modes.

## 7. 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, V-Z*.

-----  
Change to this chapter: Updates to tuple type naming throughout chapter; these updates are not covered in this document as they are considered minor in nature and extensive in page count. See changes listed in chapter 2 of Volume 2A for details on the minor adjustments to naming.

Added the following instructions: VPERMI2B, VPERMT2B, VPMADD52HUQ, VPMADD52LUQ, and VPMULTISHIFTQB.

Moved some instructions to chapter 7.

## VPERMI2B—Full Permute of Bytes from Two Tables Overwriting the Index

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.DDS.128.66.0F38.W0 75 /r VPERMI2B xmm1 {k1}{z}, xmm2, xmm3/m128	A	V/V	AVX512VL AVX512_VBMI	Permute bytes in xmm3/m128 and xmm2 using byte indexes in xmm1 and store the byte results in xmm1 using writemask k1.
EVEX.DDS.256.66.0F38.W0 75 /r VPERMI2B ymm1 {k1}{z}, ymm2, ymm3/m256	A	V/V	AVX512VL AVX512_VBMI	Permute bytes in ymm3/m256 and ymm2 using byte indexes in ymm1 and store the byte results in ymm1 using writemask k1.
EVEX.DDS.512.66.0F38.W0 75 /r VPERMI2B zmm1 {k1}{z}, zmm2, zmm3/m512	A	V/V	AVX512_VBMI	Permute bytes in zmm3/m512 and zmm2 using byte indexes in zmm1 and store the byte results in zmm1 using writemask k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full Mem	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

### Description

Permutes byte values in the second operand (the first source operand) and the third operand (the second source operand) using the byte indices in the first operand (the destination operand) to select byte elements from the second or third source operands. The selected byte elements are written to the destination at byte granularity under the writemask k1.

The first and second operands are ZMM/YMM/XMM registers. The first operand contains input indices to select elements from the two input tables in the 2nd and 3rd operands. The first operand is also the destination of the result. The third operand can be a ZMM/YMM/XMM register, or a 512/256/128-bit memory location. In each index byte, the id bit for table selection is bit 6/5/4, and bits [5:0]/[4:0]/[3:0] selects element within each input table.

Note that these instructions permit a byte value in the source operands to be copied to more than one location in the destination operand. Also, the same tables can be reused in subsequent iterations, but the index elements are overwritten.

Bits (MAX\_VL-1:256/128) of the destination are zeroed for VL=256,128.

**Operation****VPERMI2B (EVEX encoded versions)**

(KL, VL) = (16, 128), (32, 256), (64, 512)

IF VL = 128:

id ← 3;

ELSE IF VL = 256:

id ← 4;

ELSE IF VL = 512:

id ← 5;

FI;

TMP\_DEST[VL-1:0] ← DEST[VL-1:0];

FOR j ← 0 TO KL-1

off ← 8\*SRC1[j\*8 + id:j\*8];

IF k1[j] OR \*no writemask\*:

DEST[j\*8 + 7:j\*8] ← TMP\_DEST[j\*8+id+1]: SRC2[off+7:off] : SRC1[off+7:off];

ELSE IF \*zeroing-masking\*

DEST[j\*8 + 7:j\*8] ← 0;

\*ELSE

DEST[j\*8 + 7:j\*8] remains unchanged\*

FI;

ENDFOR

DEST[MAX\_VL-1:VL] ← 0;

**Intel C/C++ Compiler Intrinsic Equivalent**

VPERMI2B \_\_m512i \_\_mm512\_permutex2var\_epi8(\_\_m512i a, \_\_m512i idx, \_\_m512i b);

VPERMI2B \_\_m512i \_\_mm512\_mask2\_permutex2var\_epi8(\_\_m512i a, \_\_m512i idx, \_\_mmask64 k, \_\_m512i b);

VPERMI2B \_\_m512i \_\_mm512\_maskz\_permutex2var\_epi8(\_\_mmask64 k, \_\_m512i a, \_\_m512i idx, \_\_m512i b);

VPERMI2B \_\_m256i \_\_mm256\_permutex2var\_epi8(\_\_m256i a, \_\_m256i idx, \_\_m256i b);

VPERMI2B \_\_m256i \_\_mm256\_mask2\_permutex2var\_epi8(\_\_m256i a, \_\_m256i idx, \_\_mmask32 k, \_\_m256i b);

VPERMI2B \_\_m256i \_\_mm256\_maskz\_permutex2var\_epi8(\_\_mmask32 k, \_\_m256i a, \_\_m256i idx, \_\_m256i b);

VPERMI2B \_\_m128i \_\_mm\_permutex2var\_epi8(\_\_m128i a, \_\_m128i idx, \_\_m128i b);

VPERMI2B \_\_m128i \_\_mm\_mask2\_permutex2var\_epi8(\_\_m128i a, \_\_m128i idx, \_\_mmask16 k, \_\_m128i b);

VPERMI2B \_\_m128i \_\_mm\_maskz\_permutex2var\_epi8(\_\_mmask16 k, \_\_m128i a, \_\_m128i idx, \_\_m128i b);

**SIMD Floating-Point Exceptions**

None.

**Other Exceptions**

See Exceptions Type E4NF.nb.

## VPERMT2B—Full Permute of Bytes from Two Tables Overwriting a Table

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.DDS.128.66.0F38.W0 7D /r VPERMT2B xmm1 {k1}{z}, xmm2, xmm3/m128	A	V/V	AVX512VL AVX512_VBMI	Permute bytes in xmm3/m128 and xmm1 using byte indexes in xmm2 and store the byte results in xmm1 using writemask k1.
EVEX.NDS.256.66.0F38.W0 7D /r VPERMT2B ymm1 {k1}{z}, ymm2, ymm3/m256	A	V/V	AVX512VL AVX512_VBMI	Permute bytes in ymm3/m256 and ymm1 using byte indexes in ymm2 and store the byte results in ymm1 using writemask k1.
EVEX.NDS.512.66.0F38.W0 7D /r VPERMT2B zmm1 {k1}{z}, zmm2, zmm3/m512	A	V/V	AVX512_VBMI	Permute bytes in zmm3/m512 and zmm1 using byte indexes in zmm2 and store the byte results in zmm1 using writemask k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full Mem	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

### Description

Permutes byte values from two tables, comprising of the first operand (also the destination operand) and the third operand (the second source operand). The second operand (the first source operand) provides byte indices to select byte results from the two tables. The selected byte elements are written to the destination at byte granularity under the writemask k1.

The first and second operands are ZMM/YMM/XMM registers. The second operand contains input indices to select elements from the two input tables in the 1st and 3rd operands. The first operand is also the destination of the result. The second source operand can be a ZMM/YMM/XMM register, or a 512/256/128-bit memory location. In each index byte, the id bit for table selection is bit 6/5/4, and bits [5:0]/[4:0]/[3:0] selects element within each input table.

Note that these instructions permit a byte value in the source operands to be copied to more than one location in the destination operand. Also, the second table and the indices can be reused in subsequent iterations, but the first table is overwritten.

Bits (MAX\_VL-1:256/128) of the destination are zeroed for VL=256,128.

**Operation****VPERMT2B (EVEX encoded versions)**

(KL, VL) = (16, 128), (32, 256), (64, 512)

IF VL = 128:

id ← 3;

ELSE IF VL = 256:

id ← 4;

ELSE IF VL = 512:

id ← 5;

FI;

TMP\_DEST[VL-1:0] ← DEST[VL-1:0];

FOR j ← 0 TO KL-1

off ← 8\*SRC1[j\*8 + id; j\*8];

IF k1[j] OR \*no writemask\*:

DEST[j\*8 + 7: j\*8] ← SRC1[j\*8+id+1]? SRC2[off+7:off] : TMP\_DEST[off+7:off];

ELSE IF \*zeroing-masking\*

DEST[j\*8 + 7: j\*8] ← 0;

\*ELSE

DEST[j\*8 + 7: j\*8] remains unchanged\*

FI;

ENDFOR

DEST[MAX\_VL-1:VL] ← 0;

**Intel C/C++ Compiler Intrinsic Equivalent**

VPERMT2B \_\_m512i \_\_mm512\_permutex2var\_epi8(\_\_m512i a, \_\_m512i idx, \_\_m512i b);

VPERMT2B \_\_m512i \_\_mm512\_mask\_permutex2var\_epi8(\_\_m512i a, \_\_mmask64 k, \_\_m512i idx, \_\_m512i b);

VPERMT2B \_\_m512i \_\_mm512\_maskz\_permutex2var\_epi8(\_\_mmask64 k, \_\_m512i a, \_\_m512i idx, \_\_m512i b);

VPERMT2B \_\_m256i \_\_mm256\_permutex2var\_epi8(\_\_m256i a, \_\_m256i idx, \_\_m256i b);

VPERMT2B \_\_m256i \_\_mm256\_mask\_permutex2var\_epi8(\_\_m256i a, \_\_mmask32 k, \_\_m256i idx, \_\_m256i b);

VPERMT2B \_\_m256i \_\_mm256\_maskz\_permutex2var\_epi8(\_\_mmask32 k, \_\_m256i a, \_\_m256i idx, \_\_m256i b);

VPERMT2B \_\_m128i \_\_mm\_permutex2var\_epi8(\_\_m128i a, \_\_m128i idx, \_\_m128i b);

VPERMT2B \_\_m128i \_\_mm\_mask\_permutex2var\_epi8(\_\_m128i a, \_\_mmask16 k, \_\_m128i idx, \_\_m128i b);

VPERMT2B \_\_m128i \_\_mm\_maskz\_permutex2var\_epi8(\_\_mmask16 k, \_\_m128i a, \_\_m128i idx, \_\_m128i b);

**SIMD Floating-Point Exceptions**

None.

**Other Exceptions**

See Exceptions Type E4NF.nb.

## VPMADD52HUQ—Packed Multiply of Unsigned 52-bit Unsigned Integers and Add High 52-bit Products to 64-bit Accumulators

Opcode/ Instruction	Op/ En	32/64 bit Mode Support	CPUID	Description
EVEX.DDS.128.66.0F38.W1 B5 /r VPMADD52HUQ xmm1 {k1}{z}, xmm2, xmm3/m128/m64bcst	A	V/V	AVX512_IFMA AVX512VL	Multiply unsigned 52-bit integers in xmm2 and xmm3/m128 and add the high 52 bits of the 104-bit product to the qword unsigned integers in xmm1 using writemask k1.
EVEX.DDS.256.66.0F38.W1 B5 /r VPMADD52HUQ ymm1 {k1}{z}, ymm2, ymm3/m256/m64bcst	A	V/V	AVX512_IFMA AVX512VL	Multiply unsigned 52-bit integers in ymm2 and ymm3/m128 and add the high 52 bits of the 104-bit product to the qword unsigned integers in ymm1 using writemask k1.
EVEX.DDS.512.66.0F38.W1 B5 /r VPMADD52HUQ zmm1 {k1}{z}, zmm2, zmm3/m512/m64bcst	A	V/V	AVX512_IFMA	Multiply unsigned 52-bit integers in zmm2 and zmm3/m128 and add the high 52 bits of the 104-bit product to the qword unsigned integers in zmm1 using writemask k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m(r)	NA

### Description

Multiplies packed unsigned 52-bit integers in each qword element of the first source operand (the second operand) with the packed unsigned 52-bit integers in the corresponding elements of the second source operand (the third operand) to form packed 104-bit intermediate results. The high 52-bit, unsigned integer of each 104-bit product is added to the corresponding qword unsigned integer of the destination operand (the first operand) under the writemask k1.

The first source operand is a ZMM/YMM/XMM register. The second source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcasted from a 64-bit memory location. The destination operand is a ZMM/YMM/XMM register conditionally updated with writemask k1 at 64-bit granularity.



**Operation****VPMADD52HUQ (EVEX encoded)**

(KL, VL) = (2, 128), (4, 256), (8, 512)

FOR j ← 0 TO KL-1

i ← j \* 64;

IF k1[j] OR \*no writemask\* THEN

IF src2 is Memory AND EVEX.b=1 THEN

tsrc2[63:0] ← ZeroExtend64(src2[51:0]);

ELSE

tsrc2[63:0] ← ZeroExtend64(src2[i+51:i]);

FI;

Temp128[127:0] ← ZeroExtend64(src1[i+51:i]) \* tsrc2[63:0];

Temp2[63:0] ← DEST[i+63:i] + ZeroExtend64(temp128[103:52]);

DEST[i+63:i] ← Temp2[63:0];

ELSE

IF \*zeroing-masking\* THEN

DEST[i+63:i] ← 0;

ELSE \*merge-masking\*

DEST[i+63:i] is unchanged;

FI;

FI;

ENDFOR

DEST[MAX\_VL-1:VL] ← 0

**Intel C/C++ Compiler Intrinsic Equivalent**

VPMADD52HUQ \_\_m512i \_\_mm512\_madd52hi\_epu64( \_\_m512i a, \_\_m512i b, \_\_m512i c);

VPMADD52HUQ \_\_m512i \_\_mm512\_mask\_madd52hi\_epu64(\_\_m512i s, \_\_mmask8 k, \_\_m512i a, \_\_m512i b, \_\_m512i c);

VPMADD52HUQ \_\_m512i \_\_mm512\_maskz\_madd52hi\_epu64( \_\_mmask8 k, \_\_m512i a, \_\_m512i b, \_\_m512i c);

VPMADD52HUQ \_\_m256i \_\_mm256\_madd52hi\_epu64( \_\_m256i a, \_\_m256i b, \_\_m256i c);

VPMADD52HUQ \_\_m256i \_\_mm256\_mask\_madd52hi\_epu64(\_\_m256i s, \_\_mmask8 k, \_\_m256i a, \_\_m256i b, \_\_m256i c);

VPMADD52HUQ \_\_m256i \_\_mm256\_maskz\_madd52hi\_epu64( \_\_mmask8 k, \_\_m256i a, \_\_m256i b, \_\_m256i c);

VPMADD52HUQ \_\_m128i \_\_mm\_madd52hi\_epu64( \_\_m128i a, \_\_m128i b, \_\_m128i c);

VPMADD52HUQ \_\_m128i \_\_mm\_mask\_madd52hi\_epu64(\_\_m128i s, \_\_mmask8 k, \_\_m128i a, \_\_m128i b, \_\_m128i c);

VPMADD52HUQ \_\_m128i \_\_mm\_maskz\_madd52hi\_epu64( \_\_mmask8 k, \_\_m128i a, \_\_m128i b, \_\_m128i c);

**Flags Affected**

None.

**SIMD Floating-Point Exceptions**

None

**Other Exceptions**

See Exceptions Type E4.

## VPMADD52LUQ—Packed Multiply of Unsigned 52-bit Integers and Add the Low 52-bit Products to Qword Accumulators

Opcode/ Instruction	Op/En	32/64 bit Mode Support	CPUID	Description
EVEX.DDS.128.66.0F38.W1 B4 /r VPMADD52LUQ xmm1 {k1}{z}, xmm2,xmm3/m128/m64bcst	A	V/V	AVX512_IFMA AVX512VL	Multiply unsigned 52-bit integers in xmm2 and xmm3/m128 and add the low 52 bits of the 104-bit product to the qword unsigned integers in xmm1 using writemask k1.
EVEX.DDS.256.66.0F38.W1 B4 /r VPMADD52LUQ ymm1 {k1}{z}, ymm2, ymm3/m256/m64bcst	A	V/V	AVX512_IFMA AVX512VL	Multiply unsigned 52-bit integers in ymm2 and ymm3/m128 and add the low 52 bits of the 104-bit product to the qword unsigned integers in ymm1 using writemask k1.
EVEX.DDS.512.66.0F38.W1 B4 /r VPMADD52LUQ zmm1 {k1}{z}, zmm2,zmm3/m512/m64bcst	A	V/V	AVX512_IFMA	Multiply unsigned 52-bit integers in zmm2 and zmm3/m128 and add the low 52 bits of the 104-bit product to the qword unsigned integers in zmm1 using writemask k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m(r)	NA

### Description

Multiplies packed unsigned 52-bit integers in each qword element of the first source operand (the second operand) with the packed unsigned 52-bit integers in the corresponding elements of the second source operand (the third operand) to form packed 104-bit intermediate results. The low 52-bit, unsigned integer of each 104-bit product is added to the corresponding qword unsigned integer of the destination operand (the first operand) under the writemask k1.

The first source operand is a ZMM/YMM/XMM register. The second source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcasted from a 64-bit memory location. The destination operand is a ZMM/YMM/XMM register conditionally updated with writemask k1 at 64-bit granularity.

**Operation****VPMADD52LUQ (EVEX encoded)**

(KL, VL) = (2, 128), (4, 256), (8, 512)

FOR j ← 0 TO KL-1

i ← j \* 64;

IF k1[j] OR \*no writemask\* THEN

IF src2 is Memory AND EVEX.b=1 THEN

tsrc2[63:0] ← ZeroExtend64(src2[51:0]);

ELSE

tsrc2[63:0] ← ZeroExtend64(src2[i+51:i]);

FI;

Temp128[127:0] ← ZeroExtend64(src1[i+51:i]) \* tsrc2[63:0];

Temp2[63:0] ← DEST[i+63:i] + ZeroExtend64(temp128[51:0]);

DEST[i+63:i] ← Temp2[63:0];

ELSE

IF \*zeroing-masking\* THEN

DEST[i+63:i] ← 0;

ELSE \*merge-masking\*

DEST[i+63:i] is unchanged;

FI;

FI;

ENDFOR

DEST[MAX\_VL-1:VL] ← 0;

**Intel C/C++ Compiler Intrinsic Equivalent**

VPMADD52LUQ \_\_m512i \_\_mm512\_madd52lo\_epu64(\_\_m512i a, \_\_m512i b, \_\_m512i c);

VPMADD52LUQ \_\_m512i \_\_mm512\_mask\_madd52lo\_epu64(\_\_m512i s, \_\_mmask8 k, \_\_m512i a, \_\_m512i b, \_\_m512i c);

VPMADD52LUQ \_\_m512i \_\_mm512\_maskz\_madd52lo\_epu64(\_\_mmask8 k, \_\_m512i a, \_\_m512i b, \_\_m512i c);

VPMADD52LUQ \_\_m256i \_\_mm256\_madd52lo\_epu64(\_\_m256i a, \_\_m256i b, \_\_m256i c);

VPMADD52LUQ \_\_m256i \_\_mm256\_mask\_madd52lo\_epu64(\_\_m256i s, \_\_mmask8 k, \_\_m256i a, \_\_m256i b, \_\_m256i c);

VPMADD52LUQ \_\_m256i \_\_mm256\_maskz\_madd52lo\_epu64(\_\_mmask8 k, \_\_m256i a, \_\_m256i b, \_\_m256i c);

VPMADD52LUQ \_\_m128i \_\_mm\_madd52lo\_epu64(\_\_m128i a, \_\_m128i b, \_\_m128i c);

VPMADD52LUQ \_\_m128i \_\_mm\_mask\_madd52lo\_epu64(\_\_m128i s, \_\_mmask8 k, \_\_m128i a, \_\_m128i b, \_\_m128i c);

VPMADD52LUQ \_\_m128i \_\_mm\_maskz\_madd52lo\_epu64(\_\_mmask8 k, \_\_m128i a, \_\_m128i b, \_\_m128i c);

**Flags Affected**

None.

**SIMD Floating-Point Exceptions**

None

**Other Exceptions**

See Exceptions Type E4.

## VPMULTISHIFTQB - Select Packed Unaligned Bytes from Quadword Sources

Opcode / Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.NDS.128.66.0F38.W1 83 /r VPMULTISHIFTQB xmm1 {k1}{z}, xmm2,xmm3/m128/m64bcst	A	V/V	AVX512_VBMI AVX512VL	Select unaligned bytes from qwords in xmm3/m128/m64bcst using control bytes in xmm2, write byte results to xmm1 under k1.
EVEX.NDS.256.66.0F38.W1 83 /r VPMULTISHIFTQB ymm1 {k1}{z}, ymm2,ymm3/m256/m64bcst	A	V/V	AVX512_VBMI AVX512VL	Select unaligned bytes from qwords in ymm3/m256/m64bcst using control bytes in ymm2, write byte results to ymm1 under k1.
EVEX.NDS.512.66.0F38.W1 83 /r VPMULTISHIFTQB zmm1 {k1}{z}, zmm2,zmm3/m512/m64bcst	A	V/V	AVX512_VBMI	Select unaligned bytes from qwords in zmm3/m512/m64bcst using control bytes in zmm2, write byte results to zmm1 under k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

### Description

This instruction selects eight unaligned bytes from each input qword element of the second source operand (the third operand) and writes eight assembled bytes for each qword element in the destination operand (the first operand). Each byte result is selected using a byte-granular shift control within the corresponding qword element of the first source operand (the second operand). Each byte result in the destination operand is updated under the writemask k1.

Only the low 6 bits of each control byte are used to select an 8-bit slot to extract the output byte from the qword data in the second source operand. The starting bit of the 8-bit slot can be unaligned relative to any byte boundary and is left-shifted from the beginning of the input qword source by the amount specified in the low 6-bit of the control byte. If the 8-bit slot would exceed the qword boundary, the out-of-bound portion of the 8-bit slot is wrapped back to start from bit 0 of the input qword element.

The first source operand is a ZMM/YMM/XMM register. The second source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcasted from a 64-bit memory location. The destination operand is a ZMM/YMM/XMM register.

**Operation****VPMULTISHIFTQB DEST, SRC1, SRC2 (EVEX encoded version)**

(KL, VL) = (2, 128), (4, 256), (8, 512)

FOR i ← 0 TO KL-1

IF EVEX.b=1 AND src2 is memory THEN

tcur ← src2.qword[0]; //broadcasting

ELSE

tcur ← src2.qword[i];

FI;

FOR j ← 0 to 7

ctrl ← src1.qword[i].byte[j] &amp; 63;

FOR k ← 0 to 7

res.bit[k] ← tcur.bit[(ctrl+k) mod 64];

ENDFOR

IF k1[i\*8+j] or no writemask THEN

dst.qword[i].byte[j] ← res;

ELSE IF zeroing-masking THEN

dst.qword[i].byte[j] ← 0;

ENDFOR

ENDFOR

DEST.qword[MAX\_VL-1:VL] ← 0;

**Intel C/C++ Compiler Intrinsic Equivalent**

VPMULTISHIFTQB \_\_m512i \_\_mm512\_multishift\_epi64\_epi8( \_\_m512i a, \_\_m512i b);

VPMULTISHIFTQB \_\_m512i \_\_mm512\_mask\_multishift\_epi64\_epi8(\_\_m512i s, \_\_mmask64 k, \_\_m512i a, \_\_m512i b);

VPMULTISHIFTQB \_\_m512i \_\_mm512\_maskz\_multishift\_epi64\_epi8( \_\_mmask64 k, \_\_m512i a, \_\_m512i b);

VPMULTISHIFTQB \_\_m256i \_\_mm256\_multishift\_epi64\_epi8( \_\_m256i a, \_\_m256i b);

VPMULTISHIFTQB \_\_m256i \_\_mm256\_mask\_multishift\_epi64\_epi8(\_\_m256i s, \_\_mmask32 k, \_\_m256i a, \_\_m256i b);

VPMULTISHIFTQB \_\_m256i \_\_mm256\_maskz\_multishift\_epi64\_epi8( \_\_mmask32 k, \_\_m256i a, \_\_m256i b);

VPMULTISHIFTQB \_\_m128i \_\_mm\_multishift\_epi64\_epi8( \_\_m128i a, \_\_m128i b);

VPMULTISHIFTQB \_\_m128i \_\_mm\_mask\_multishift\_epi64\_epi8(\_\_m128i s, \_\_mmask8 k, \_\_m128i a, \_\_m128i b);

VPMULTISHIFTQB \_\_m128i \_\_mm\_maskz\_multishift\_epi64\_epi8( \_\_mmask8 k, \_\_m128i a, \_\_m128i b);

**SIMD Floating-Point Exceptions**

None.

**Other Exceptions**

See Exceptions Type E4NF.

## 8. Addition of Chapter 7, Volume 2D

Chapter 7 was added to the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D: Instruction Set Reference*.

-----

New chapter added.



# CHAPTER 7 INSTRUCTION SET REFERENCE UNIQUE TO INTEL® XEON PHI™ PROCESSORS

---

This chapter describes the instruction set that is unique to Intel® Xeon Phi™ Processors based on the Knights Landing microarchitecture. The set is not supported in any other Intel processors. Included are Intel® AVX-512 instructions. For additional instructions supported on these processors, see Chapter 3, “Instruction Set Reference, A-L”, Chapter 4, “Instruction Set Reference, M-U”, and Chapter 5, “Instruction Set Reference, V-Z”.

## PREFETCHWT1—Prefetch Vector Data Into Caches with Intent to Write and T1 Hint

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
OF 0D /2 PREFETCHWT1 m8	M	V/V	PREFETCHWT1	Move data from m8 closer to the processor using T1 hint with intent to write.

### Instruction Operand Encoding

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

### Description

Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by an intent to write hint (so that data is brought into 'Exclusive' state via a request for ownership) and a locality hint:

- T1 (temporal data with respect to first level cache)—prefetch data into the second level cache.

The source operand is a byte memory location. (The locality hints are encoded into the machine level instruction using bits 3 through 5 of the ModR/M byte. Use of any ModR/M value other than the specified ones will lead to unpredictable behavior.)

If the line selected is already present in the cache hierarchy at a level closer to the processor, no data movement occurs. Prefetches from uncacheable or WC memory are ignored.

The PREFETCHH instruction is merely a hint and does not affect program behavior. If executed, this instruction moves data closer to the processor in anticipation of future use.

The implementation of prefetch locality hints is implementation-dependent, and can be overloaded or ignored by a processor implementation. The amount of data prefetched is also processor implementation-dependent. It will, however, be a minimum of 32 bytes.

It should be noted that processors are free to speculatively fetch and cache data from system memory regions that are assigned a memory-type that permits speculative reads (that is, the WB, WC, and WT memory types). A PREFETCHH instruction is considered a hint to this speculative behavior. Because this speculative fetching can occur at any time and is not tied to instruction execution, a PREFETCHH instruction is not ordered with respect to the fence instructions (MFENCE, SFENCE, and LFENCE) or locked memory references. A PREFETCHH instruction is also unordered with respect to CLFLUSH and CLFLUSHOPT instructions, other PREFETCHH instructions, or any other general instruction. It is ordered with respect to serializing instructions such as CPUID, WRMSR, OUT, and MOV CR. This instruction's operation is the same in non-64-bit modes and 64-bit mode.

### Operation

PREFETCH(mem, Level, State) Prefetches a byte memory location pointed by 'mem' into the cache level specified by 'Level'; a request for exclusive/ownership is done if 'State' is 1. Note that the memory location ignore cache line splits. This operation is considered a hint for the processor and may be skipped depending on implementation.

Prefetch (m8, Level = 1, EXCLUSIVE=1);

### Flags Affected

All flags are affected

### C/C++ Compiler Intrinsic Equivalent

```
void _mm_prefetch( char const *, int hint= _MM_HINT_ET1);
```

### Protected Mode Exceptions

#UD If the LOCK prefix is used.



**Real-Address Mode Exceptions**

#UD If the LOCK prefix is used.

**Virtual-8086 Mode Exceptions**

#UD If the LOCK prefix is used.

**Compatibility Mode Exceptions**

#UD If the LOCK prefix is used.

**64-Bit Mode Exceptions**

#UD If the LOCK prefix is used.

## VEXP2PD—Approximation to the Exponential $2^x$ of Packed Double-Precision Floating-Point Values with Less Than $2^{-23}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W1 C8 /r VEXP2PD zmm1 {k1}{z}, zmm2/m512/m64bcst {sae}	A	V/V	AVX512ER	Computes approximations to the exponential $2^x$ (with less than $2^{-23}$ of maximum relative error) of the packed double-precision floating-point values from zmm2/m512/m64bcst and stores the floating-point result in zmm1 with writemask k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

### Description

Computes the approximate base-2 exponential evaluation of the double-precision floating-point values in the source operand (the second operand) and stores the results to the destination operand (the first operand) using the writemask k1. The approximate base-2 exponential is evaluated with less than  $2^{-23}$  of relative error.

Denormal input values are treated as zeros and do not signal #DE, irrespective of MXCSR.DAZ. Denormal results are flushed to zeros and do not signal #UE, irrespective of MXCSR.FTZ.

The source operand is a ZMM register, a 512-bit memory location or a 512-bit vector broadcasted from a 64-bit memory location. The destination operand is a ZMM register, conditionally updated using writemask k1.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

A numerically exact implementation of VEXP2xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VEXP2PD

(KL, VL) = (8, 512)

FOR j ← 0 TO KL-1

  i ← j \* 64

  IF k1[j] OR \*no writemask\* THEN

    IF (EVEX.b = 1) AND (SRC \*is memory\*)

      THEN DEST[i+63:i] ← EXP2\_23\_DP(SRC[63:0])

      ELSE DEST[i+63:i] ← EXP2\_23\_DP(SRC[i+63:i])

    FI;

  ELSE

    IF \*merging-masking\* ; merging-masking

      THEN \*DEST[i+63:i] remains unchanged\*

    ELSE ; zeroing-masking

      DEST[i+63:i] ← 0

    FI;

  FI;

ENDFOR;

**Table 6-1. Special Values Behavior**

Source Input	Result	Comments
NaN	QNaN(src)	If (SRC = SNaN) then #I
$+\infty$	$+\infty$	
$+/-0$	1.0f	<i>Exact result</i>
$-\infty$	+0.0f	
Integral value N	$2^N$	<i>Exact result</i>

**Intel C/C++ Compiler Intrinsic Equivalent**

VEXP2PD \_\_m512d \_\_mm512\_exp2a23\_round\_pd (\_\_m512d a, int sae);

VEXP2PD \_\_m512d \_\_mm512\_mask\_exp2a23\_round\_pd (\_\_m512d a, \_\_mmask8 m, \_\_m512d b, int sae);

VEXP2PD \_\_m512d \_\_mm512\_maskz\_exp2a23\_round\_pd (\_\_mmask8 m, \_\_m512d b, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Overflow

**Other Exceptions**

See Exceptions Type E2.

## VEXP2PS—Approximation to the Exponential $2^x$ of Packed Single-Precision Floating-Point Values with Less Than $2^{-23}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W0 C8 /r VEXP2PS zmm1 {k1}{z}, zmm2/m512/m32bcst {sae}	A	V/V	AVX512ER	Computes approximations to the exponential $2^x$ (with less than $2^{-23}$ of maximum relative error) of the packed single-precision floating-point values from zmm2/m512/m32bcst and stores the floating-point result in zmm1 with writemask k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

### Description

Computes the approximate base-2 exponential evaluation of the single-precision floating-point values in the source operand (the second operand) and store the results in the destination operand (the first operand) using the writemask k1. The approximate base-2 exponential is evaluated with less than  $2^{-23}$  of relative error.

Denormal input values are treated as zeros and do not signal #DE, irrespective of MXCSR.DAZ. Denormal results are flushed to zeros and do not signal #UE, irrespective of MXCSR.FTZ.

The source operand is a ZMM register, a 512-bit memory location, or a 512-bit vector broadcasted from a 32-bit memory location. The destination operand is a ZMM register, conditionally updated using writemask k1.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

A numerically exact implementation of VEXP2xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VEXP2PS

(KL, VL) = (16, 512)

FOR j ← 0 TO KL-1

  i ← j \* 32

  IF k1[j] OR \*no writemask\* THEN

    IF (EVEX.b = 1) AND (SRC \*is memory\*)

      THEN DEST[i+31:i] ← EXP2\_23\_SP(SRC[31:0])

      ELSE DEST[i+31:i] ← EXP2\_23\_SP(SRC[i+31:i])

    FI;

  ELSE

    IF \*merging-masking\* ; merging-masking

      THEN \*DEST[i+31:i] remains unchanged\*

    ELSE ; zeroing-masking

      DEST[i+31:i] ← 0

    FI;

  FI;

ENDFOR;

**Table 6-2. Special Values Behavior**

Source Input	Result	Comments
NaN	QNaN(src)	If (SRC = SNaN) then #I
$+\infty$	$+\infty$	
$\pm 0$	1.0f	<i>Exact result</i>
$-\infty$	+0.0f	
Integral value N	$2^N$	<i>Exact result</i>

**Intel C/C++ Compiler Intrinsic Equivalent**

VEXP2PS \_\_m512 \_\_mm512\_exp2a23\_round\_ps (\_\_m512 a, int sae);

VEXP2PS \_\_m512 \_\_mm512\_mask\_exp2a23\_round\_ps (\_\_m512 a, \_\_mmask16 m, \_\_m512 b, int sae);

VEXP2PS \_\_m512 \_\_mm512\_maskz\_exp2a23\_round\_ps (\_\_mmask16 m, \_\_m512 b, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Overflow

**Other Exceptions**

See Exceptions Type E2.

## VGATHERPFODPS/VGATHERPFOQPS/VGATHERPFODPD/VGATHERPFOQPD—Sparse Prefetch Packed SP/DP Data Values with Signed Dword, Signed Qword Indices Using T0 Hint

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W0 C6 /1 /vsib VGATHERPFODPS vm32z {k1}	A	V/V	AVX512PF	Using signed dword indices, prefetch sparse byte memory locations containing single-precision data using opmask k1 and T0 hint.
EVEX.512.66.0F38.W0 C7 /1 /vsib VGATHERPFOQPS vm64z {k1}	A	V/V	AVX512PF	Using signed qword indices, prefetch sparse byte memory locations containing single-precision data using opmask k1 and T0 hint.
EVEX.512.66.0F38.W1 C6 /1 /vsib VGATHERPFODPD vm32y {k1}	A	V/V	AVX512PF	Using signed dword indices, prefetch sparse byte memory locations containing double-precision data using opmask k1 and T0 hint.
EVEX.512.66.0F38.W1 C7 /1 /vsib VGATHERPFOQPD vm64z {k1}	A	V/V	AVX512PF	Using signed qword indices, prefetch sparse byte memory locations containing double-precision data using opmask k1 and T0 hint.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	BaseReg (R): VSIB:base, VectorReg(R): VSIB:index	NA	NA	NA

### Description

The instruction conditionally prefetches up to sixteen 32-bit or eight 64-bit integer byte data elements. The elements are specified via the VSIB (i.e., the index register is an zmm, holding packed indices). Elements will only be prefetched if their corresponding mask bit is one.

Lines prefetched are loaded into to a location in the cache hierarchy specified by a locality hint (T0):

- T0 (temporal data)—prefetch data into the first level cache.

[PS data] For dword indices, the instruction will prefetch sixteen memory locations. For qword indices, the instruction will prefetch eight values.

[PD data] For dword and qword indices, the instruction will prefetch eight memory locations.

Note that:

- (1) The prefetches may happen in any order (or not at all). The instruction is a hint.
- (2) The mask is left unchanged.
- (3) Not valid with 16-bit effective addresses. Will deliver a #UD fault.
- (4) No FP nor memory faults may be produced by this instruction.
- (5) Prefetches do not handle cache line splits
- (6) A #UD is signaled if the memory operand is encoded without the SIB byte.

### Operation

BASE\_ADDR stands for the memory operand base address (a GPR); may not exist

VINDEX stands for the memory operand vector of indices (a vector register)

SCALE stands for the memory operand scalar (1, 2, 4 or 8)

DISP is the optional 1, 2 or 4 byte displacement

PREFETCH(mem, Level, State) Prefetches a byte memory location pointed by 'mem' into the cache level specified by 'Level'; a request for exclusive/ownership is done if 'State' is 1. Note that the memory location ignore cache line splits. This operation is considered a hint for the processor and may be skipped depending on implementation.

**VGATHERPFODPS (EVEX encoded version)**

```
(KL, VL) = (16, 512)
FOR j ← 0 TO KL-1
  i ← j * 32
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[j+31:i]) * SCALE + DISP], Level=0, RFO = 0)
  FI;
ENDFOR
```

**VGATHERPFODPD (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  k ← j * 32
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[k+31:k]) * SCALE + DISP], Level=0, RFO = 0)
  FI;
ENDFOR
```

**VGATHERPFOQPS (EVEX encoded version)**

```
(KL, VL) = (8, 256)
FOR j ← 0 TO KL-1
  i ← j * 64
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[j+63:i]) * SCALE + DISP], Level=0, RFO = 0)
  FI;
ENDFOR
```

**VGATHERPFOQPD (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  k ← j * 64
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[k+63:k]) * SCALE + DISP], Level=0, RFO = 0)
  FI;
ENDFOR
```

**Intel C/C++ Compiler Intrinsic Equivalent**

```
VGATHERPFODPD void __mm512_mask_prefetch_i32gather_pd(__m256i vdx, __mmask8 m, void * base, int scale, int hint);
VGATHERPFODPS void __mm512_mask_prefetch_i32gather_ps(__m512i vdx, __mmask16 m, void * base, int scale, int hint);
VGATHERPFOQPD void __mm512_mask_prefetch_i64gather_pd(__m512i vdx, __mmask8 m, void * base, int scale, int hint);
VGATHERPFOQPS void __mm512_mask_prefetch_i64gather_ps(__m512i vdx, __mmask8 m, void * base, int scale, int hint);
```

**SIMD Floating-Point Exceptions**

None

**Other Exceptions**

See Exceptions Type E12NP.

## VGATHERPF1DPS/VGATHERPF1QPS/VGATHERPF1DPD/VGATHERPF1QPD—Sparse Prefetch Packed SP/DP Data Values with Signed Dword, Signed Qword Indices Using T1 Hint

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W0 C6 /2 /vsib VGATHERPF1DPS vm32z {k1}	A	V/V	AVX512PF	Using signed dword indices, prefetch sparse byte memory locations containing single-precision data using opmask k1 and T1 hint.
EVEX.512.66.0F38.W0 C7 /2 /vsib VGATHERPF1QPS vm64z {k1}	A	V/V	AVX512PF	Using signed qword indices, prefetch sparse byte memory locations containing single-precision data using opmask k1 and T1 hint.
EVEX.512.66.0F38.W1 C6 /2 /vsib VGATHERPF1DPD vm32y {k1}	A	V/V	AVX512PF	Using signed dword indices, prefetch sparse byte memory locations containing double-precision data using opmask k1 and T1 hint.
EVEX.512.66.0F38.W1 C7 /2 /vsib VGATHERPF1QPD vm64z {k1}	A	V/V	AVX512PF	Using signed qword indices, prefetch sparse byte memory locations containing double-precision data using opmask k1 and T1 hint.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	BaseReg (R): VSIB:base, VectorReg(R): VSIB:index	NA	NA	NA

#### Description

The instruction conditionally prefetches up to sixteen 32-bit or eight 64-bit integer byte data elements. The elements are specified via the VSIB (i.e., the index register is an zmm, holding packed indices). Elements will only be prefetched if their corresponding mask bit is one.

Lines prefetched are loaded into to a location in the cache hierarchy specified by a locality hint (T1):

- T1 (temporal data)—prefetch data into the second level cache.

[PS data] For dword indices, the instruction will prefetch sixteen memory locations. For qword indices, the instruction will prefetch eight values.

[PD data] For dword and qword indices, the instruction will prefetch eight memory locations.

Note that:

- (1) The prefetches may happen in any order (or not at all). The instruction is a hint.
- (2) The mask is left unchanged.
- (3) Not valid with 16-bit effective addresses. Will deliver a #UD fault.
- (4) No FP nor memory faults may be produced by this instruction.
- (5) Prefetches do not handle cache line splits
- (6) A #UD is signaled if the memory operand is encoded without the SIB byte.

#### Operation

BASE\_ADDR stands for the memory operand base address (a GPR); may not exist

VINDEX stands for the memory operand vector of indices (a vector register)

SCALE stands for the memory operand scalar (1, 2, 4 or 8)

DISP is the optional 1, 2 or 4 byte displacement

PREFETCH(mem, Level, State) Prefetches a byte memory location pointed by 'mem' into the cache level specified by 'Level'; a request for exclusive/ownership is done if 'State' is 1. Note that the memory location ignore cache line splits. This operation is considered a hint for the processor and may be skipped depending on implementation.



**VGATHERPF1DPS (EVEX encoded version)**

```
(KL, VL) = (16, 512)
FOR j ← 0 TO KL-1
  i ← j * 32
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[j+31:i]) * SCALE + DISP], Level=1, RFO = 0)
  FI;
ENDFOR
```

**VGATHERPF1DPD (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  k ← j * 32
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[k+31:k]) * SCALE + DISP], Level=1, RFO = 0)
  FI;
ENDFOR
```

**VGATHERPF1QPS (EVEX encoded version)**

```
(KL, VL) = (8, 256)
FOR j ← 0 TO KL-1
  i ← j * 64
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[j+63:i]) * SCALE + DISP], Level=1, RFO = 0)
  FI;
ENDFOR
```

**VGATHERPF1QPD (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  k ← j * 64
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[k+63:k]) * SCALE + DISP], Level=1, RFO = 0)
  FI;
ENDFOR
```

**Intel C/C++ Compiler Intrinsic Equivalent**

```
VGATHERPF1DPD void __mm512_mask_prefetch_i32gather_pd(__m256i vdx, __mmask8 m, void * base, int scale, int hint);
VGATHERPF1DPS void __mm512_mask_prefetch_i32gather_ps(__m512i vdx, __mmask16 m, void * base, int scale, int hint);
VGATHERPF1QPD void __mm512_mask_prefetch_i64gather_pd(__m512i vdx, __mmask8 m, void * base, int scale, int hint);
VGATHERPF1QPS void __mm512_mask_prefetch_i64gather_ps(__m512i vdx, __mmask8 m, void * base, int scale, int hint);
```

**SIMD Floating-Point Exceptions**

None

**Other Exceptions**

See Exceptions Type E12NP.

## VRCP28PD—Approximation to the Reciprocal of Packed Double-Precision Floating-Point Values with Less Than $2^{-28}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W1 CA /r VRCP28PD zmm1 {k1}{z}, zmm2/m512/m64bcst {sae}	A	V/V	AVX512ER	Computes the approximate reciprocals ( $< 2^{-28}$ relative error) of the packed double-precision floating-point values in zmm2/m512/m64bcst and stores the results in zmm1. Under writemask.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

### Description

Computes the reciprocal approximation of the float64 values in the source operand (the second operand) and store the results to the destination operand (the first operand). The approximate reciprocal is evaluated with less than  $2^{-28}$  of maximum relative error.

Denormal input values are treated as zeros and do not signal #DE, irrespective of MXCSR.DAZ. Denormal results are flushed to zeros and do not signal #UE, irrespective of MXCSR.FTZ.

If any source element is NaN, the quietized NaN source value is returned for that element. If any source element is  $\pm\infty$ ,  $\pm 0.0$  is returned for that element. Also, if any source element is  $\pm 0.0$ ,  $\pm\infty$  is returned for that element.

The source operand is a ZMM register, a 512-bit memory location or a 512-bit vector broadcasted from a 64-bit memory location. The destination operand is a ZMM register, conditionally updated using writemask k1.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

A numerically exact implementation of VRCP28xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VRCP28PD (EVEX encoded versions)

(KL, VL) = (8, 512)

FOR j  $\leftarrow$  0 TO KL-1

  i  $\leftarrow$  j \* 64

  IF k1[j] OR \*no writemask\* THEN

    IF (EVEX.b = 1) AND (SRC \*is memory\*)

      THEN DEST[i+63:i]  $\leftarrow$  RCP\_28\_DP(1.0/SRC[63:0]);

      ELSE DEST[i+63:i]  $\leftarrow$  RCP\_28\_DP(1.0/SRC[i+63:i]);

    FI;

  ELSE

    IF \*merging-masking\* ; merging-masking

      THEN \*DEST[i+63:i] remains unchanged\*

    ELSE ; zeroing-masking

      DEST[i+63:i]  $\leftarrow$  0

    FI;

  FI;

ENDFOR;

Table 6-3. VRCP28PD Special Cases

Input value	Result value	Comments
NAN	QNAN(input)	If (SRC = SNaN) then #I
$0 \leq X < 2^{-1022}$	INF	Positive input denormal or zero; #Z
$-2^{-1022} < X \leq -0$	-INF	Negative input denormal or zero; #Z
$X > 2^{1022}$	+0.0f	
$X < -2^{1022}$	-0.0f	
$X = +\infty$	+0.0f	
$X = -\infty$	-0.0f	
$X = 2^{-n}$	$2^n$	Exact result (unless input/output is a denormal)
$X = -2^{-n}$	$-2^n$	Exact result (unless input/output is a denormal)

**Intel C/C++ Compiler Intrinsic Equivalent**

VRCP28PD \_\_m512d \_\_mm512\_rcp28\_round\_pd (\_\_m512d a, int sae);

VRCP28PD \_\_m512d \_\_mm512\_mask\_rcp28\_round\_pd(\_\_m512d a, \_\_mmask8 m, \_\_m512d b, int sae);

VRCP28PD \_\_m512d \_\_mm512\_maskz\_rcp28\_round\_pd(\_\_mmask8 m, \_\_m512d b, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Divide-by-zero

**Other Exceptions**

See Exceptions Type E2.

## VRCP28SD—Approximation to the Reciprocal of Scalar Double-Precision Floating-Point Value with Less Than $2^{-28}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.NDS.LIG.66.0F38.W1 CB /r VRCP28SD xmm1 {k1}{z}, xmm2, xmm3/m64 {sae}	A	V/V	AVX512ER	Computes the approximate reciprocal ( $< 2^{-28}$ relative error) of the scalar double-precision floating-point value in xmm3/m64 and stores the results in xmm1. Under writemask. Also, upper double-precision floating-point value (bits[127:64]) from xmm2 is copied to xmm1[127:64].

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	EVEX.vvvv	ModRM:r/m (r)	NA

### Description

Computes the reciprocal approximation of the low float64 value in the second source operand (the third operand) and store the result to the destination operand (the first operand). The approximate reciprocal is evaluated with less than  $2^{-28}$  of maximum relative error. The result is written into the low float64 element of the destination operand according to the writemask k1. Bits 127:64 of the destination is copied from the corresponding bits of the first source operand (the second operand).

A denormal input value is treated as zero and does not signal #DE, irrespective of MXCSR.DAZ. A denormal result is flushed to zero and does not signal #UE, irrespective of MXCSR.FTZ.

If any source element is NaN, the quietized NaN source value is returned for that element. If any source element is  $\pm\infty$ ,  $\pm 0.0$  is returned for that element. Also, if any source element is  $\pm 0.0$ ,  $\pm\infty$  is returned for that element.

The first source operand is an XMM register. The second source operand is an XMM register or a 64-bit memory location. The destination operand is a XMM register, conditionally updated using writemask k1.

A numerically exact implementation of VRCP28xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VRCP28SD ((EVEX encoded versions)

```

IF k1[0] OR *no writemask* THEN
    DEST[63: 0] ← RCP_28_DP(1.0/SRC2[63: 0]);
ELSE
    IF *merging-masking*                ; merging-masking
        THEN *DEST[63: 0] remains unchanged*
    ELSE                                  ; zeroing-masking
        DEST[63: 0] ← 0
    FI;
FI;
ENDFOR;
DEST[127:64] ← SRC1[127: 64]
DEST[MAXVL-1:128] ← 0

```

Table 6-4. VRCP28SD Special Cases

Input value	Result value	Comments
NAN	QNAN(input)	If (SRC = SNaN) then #I
$0 \leq X < 2^{-1022}$	INF	Positive input denormal or zero; #Z
$-2^{-1022} < X \leq -0$	-INF	Negative input denormal or zero; #Z
$X > 2^{1022}$	+0.0f	
$X < -2^{1022}$	-0.0f	
$X = +\infty$	+0.0f	
$X = -\infty$	-0.0f	
$X = 2^{-n}$	$2^n$	Exact result (unless input/output is a denormal)
$X = -2^{-n}$	$-2^n$	Exact result (unless input/output is a denormal)

**Intel C/C++ Compiler Intrinsic Equivalent**

VRCP28SD \_\_m128d \_\_mm\_rcp28\_round\_sd (\_\_m128d a, \_\_m128d b, int sae);

VRCP28SD \_\_m128d \_\_mm\_mask\_rcp28\_round\_sd(\_\_m128d s, \_\_mmask8 m, \_\_m128d a, \_\_m128d b, int sae);

VRCP28SD \_\_m128d \_\_mm\_maskz\_rcp28\_round\_sd(\_\_mmask8 m, \_\_m128d a, \_\_m128d b, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Divide-by-zero

**Other Exceptions**

See Exceptions Type E3.

## VRCP28PS—Approximation to the Reciprocal of Packed Single-Precision Floating-Point Values with Less Than $2^{-28}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W0 CA /r VRCP28PS zmm1 {k1}{z}, zmm2/m512/m32bcst {sae}	A	V/V	AVX512ER	Computes the approximate reciprocals ( $< 2^{-28}$ relative error) of the packed single-precision floating-point values in zmm2/m512/m32bcst and stores the results in zmm1. Under writemask.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

### Description

Computes the reciprocal approximation of the float32 values in the source operand (the second operand) and store the results to the destination operand (the first operand) using the writemask k1. The approximate reciprocal is evaluated with less than  $2^{-28}$  of maximum relative error prior to final rounding. The final results are rounded to  $< 2^{-23}$  relative error before written to the destination.

Denormal input values are treated as zeros and do not signal #DE, irrespective of MXCSR.DAZ. Denormal results are flushed to zeros and do not signal #UE, irrespective of MXCSR.FTZ.

If any source element is NaN, the quietized NaN source value is returned for that element. If any source element is  $\pm\infty$ ,  $\pm 0.0$  is returned for that element. Also, if any source element is  $\pm 0.0$ ,  $\pm\infty$  is returned for that element.

The source operand is a ZMM register, a 512-bit memory location, or a 512-bit vector broadcasted from a 32-bit memory location. The destination operand is a ZMM register, conditionally updated using writemask k1.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

A numerically exact implementation of VRCP28xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VRCP28PS (EVEX encoded versions)

(KL, VL) = (16, 512)

FOR j  $\leftarrow$  0 TO KL-1

  i  $\leftarrow$  j \* 32

  IF k1[j] OR \*no writemask\* THEN

    IF (EVEX.b = 1) AND (SRC \*is memory\*)

      THEN DEST[i+31:i]  $\leftarrow$  RCP\_28\_SP(1.0/SRC[31:0]);

      ELSE DEST[i+31:i]  $\leftarrow$  RCP\_28\_SP(1.0/SRC[i+31:i]);

    FI;

  ELSE

    IF \*merging-masking\* ; merging-masking

      THEN \*DEST[i+31:i] remains unchanged\*

    ELSE ; zeroing-masking

      DEST[i+31:i]  $\leftarrow$  0

    FI;

  FI;

ENDFOR;

Table 6-5. VRCP28PS Special Cases

Input value	Result value	Comments
NAN	QNAN(input)	If (SRC = SNaN) then #I
$0 \leq X < 2^{-126}$	INF	Positive input denormal or zero; #Z
$-2^{-126} < X \leq -0$	-INF	Negative input denormal or zero; #Z
$X > 2^{126}$	+0.0f	
$X < -2^{126}$	-0.0f	
$X = +\infty$	+0.0f	
$X = -\infty$	-0.0f	
$X = 2^{-n}$	$2^n$	Exact result (unless input/output is a denormal)
$X = -2^{-n}$	$-2^n$	Exact result (unless input/output is a denormal)

**Intel C/C++ Compiler Intrinsic Equivalent**

VRCP28PS \_\_mm512\_rcp28\_round\_ps (\_\_m512 a, int sae);

VRCP28PS \_\_m512 \_\_mm512\_mask\_rcp28\_round\_ps(\_\_m512 s, \_\_mmask16 m, \_\_m512 a, int sae);

VRCP28PS \_\_m512 \_\_mm512\_maskz\_rcp28\_round\_ps(\_\_mmask16 m, \_\_m512 a, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Divide-by-zero

**Other Exceptions**

See Exceptions Type E2.

## VRCP28SS—Approximation to the Reciprocal of Scalar Single-Precision Floating-Point Value with Less Than $2^{-28}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.NDS.LIG.66.0F38.W0 CB /r VRCP28SS xmm1 {k1}{z}, xmm2, xmm3/m32 {sae}	A	V/V	AVX512ER	Computes the approximate reciprocal ( $< 2^{-28}$ relative error) of the scalar single-precision floating-point value in xmm3/m32 and stores the results in xmm1. Under writemask. Also, upper 3 single-precision floating-point values (bits[127:32]) from xmm2 is copied to xmm1[127:32].

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	EVEX.vvvv	ModRM:r/m (r)	NA

### Description

Computes the reciprocal approximation of the low float32 value in the second source operand (the third operand) and store the result to the destination operand (the first operand). The approximate reciprocal is evaluated with less than  $2^{-28}$  of maximum relative error prior to final rounding. The final result is rounded to  $< 2^{-23}$  relative error before written into the low float32 element of the destination according to writemask k1. Bits 127:32 of the destination is copied from the corresponding bits of the first source operand (the second operand).

A denormal input value is treated as zero and does not signal #DE, irrespective of MXCSR.DAZ. A denormal result is flushed to zero and does not signal #UE, irrespective of MXCSR.FTZ.

If any source element is NaN, the quietized NaN source value is returned for that element. If any source element is  $\pm\infty$ ,  $\pm 0.0$  is returned for that element. Also, if any source element is  $\pm 0.0$ ,  $\pm\infty$  is returned for that element.

The first source operand is an XMM register. The second source operand is an XMM register or a 32-bit memory location. The destination operand is a XMM register, conditionally updated using writemask k1.

A numerically exact implementation of VRCP28xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VRCP28SS ((EVEX encoded versions)

```

IF k1[0] OR *no writemask* THEN
    DEST[31: 0] ← RCP_28_SP(1.0/SRC2[31: 0]);
ELSE
    IF *merging-masking*           ; merging-masking
        THEN *DEST[31: 0] remains unchanged*
    ELSE                             ; zeroing-masking
        DEST[31: 0] ← 0
    FI;
FI;
ENDFOR;
DEST[127:32] ← SRC1[127: 32]
DEST[MAXVL-1:128] ← 0

```



Table 6-6. VRCP28SS Special Cases

Input value	Result value	Comments
NAN	QNAN(input)	If (SRC = SNaN) then #I
$0 \leq X < 2^{-126}$	INF	Positive input denormal or zero; #Z
$-2^{-126} < X \leq -0$	-INF	Negative input denormal or zero; #Z
$X > 2^{126}$	+0.0f	
$X < -2^{126}$	-0.0f	
$X = +\infty$	+0.0f	
$X = -\infty$	-0.0f	
$X = 2^{-n}$	$2^n$	Exact result (unless input/output is a denormal)
$X = -2^{-n}$	$-2^n$	Exact result (unless input/output is a denormal)

#### Intel C/C++ Compiler Intrinsic Equivalent

VRCP28SS \_\_m128\_mm\_rcp28\_round\_ss (\_\_m128 a, \_\_m128 b, int sae);

VRCP28SS \_\_m128\_mm\_mask\_rcp28\_round\_ss(\_\_m128 s, \_\_mmask8 m, \_\_m128 a, \_\_m128 b, int sae);

VRCP28SS \_\_m128\_mm\_maskz\_rcp28\_round\_ss(\_\_mmask8 m, \_\_m128 a, \_\_m128 b, int sae);

#### SIMD Floating-Point Exceptions

Invalid (if SNaN input), Divide-by-zero

#### Other Exceptions

See Exceptions Type E3.

## VRSQRT28PD—Approximation to the Reciprocal Square Root of Packed Double-Precision Floating-Point Values with Less Than $2^{-28}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W1 CC /r VRSQRT28PD zmm1 {k1}{z}, zmm2/m512/m64bcst {sae}	A	V/V	AVX512ER	Computes approximations to the Reciprocal square root ( $<2^{-28}$ relative error) of the packed double-precision floating-point values from zmm2/m512/m64bcst and stores result in zmm1 with writemask k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

### Description

Computes the reciprocal square root of the float64 values in the source operand (the second operand) and store the results to the destination operand (the first operand). The approximate reciprocal is evaluated with less than  $2^{-28}$  of maximum relative error.

If any source element is NaN, the quietized NaN source value is returned for that element. Negative (non-zero) source numbers, as well as  $-\infty$ , return the canonical NaN and set the Invalid Flag (#I).

A value of  $-0$  must return  $-\infty$  and set the DivByZero flags (#Z). Negative numbers should return NaN and set the Invalid flag (#I). Note however that the instruction flush input denormals to zero of the same sign, so negative denormals return  $-\infty$  and set the DivByZero flag.

The source operand is a ZMM register, a 512-bit memory location or a 512-bit vector broadcasted from a 64-bit memory location. The destination operand is a ZMM register, conditionally updated using writemask k1.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

A numerically exact implementation of VRSQRT28xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VRSQRT28PD (EVEX encoded versions)

(KL, VL) = (8, 512)

FOR j  $\leftarrow$  0 TO KL-1

  i  $\leftarrow$  j \* 64

  IF k1[j] OR \*no writemask\* THEN

    IF (EVEX.b = 1) AND (SRC \*is memory\*)

      THEN DEST[i+63:i]  $\leftarrow$  (1.0/ SQRT(SRC[63:0]));

      ELSE DEST[i+63:i]  $\leftarrow$  (1.0/ SQRT(SRC[i+63:i]));

    FI;

  ELSE

    IF \*merging-masking\* ; merging-masking

      THEN \*DEST[i+63:i] remains unchanged\*

    ELSE ; zeroing-masking

      DEST[i+63:i]  $\leftarrow$  0

    FI;

  FI;

ENDFOR;

**Table 6-7. VRSQRT28PD Special Cases**

Input value	Result value	Comments
NAN	QNAN(input)	If (SRC = SNaN) then #I
$X = 2^{-2n}$	$2^n$	
$X < 0$	QNAN_Indefinite	Including -INF
$X = -0$ or negative denormal	-INF	#Z
$X = +0$ or positive denormal	+INF	#Z
$X = +INF$	+0	

**Intel C/C++ Compiler Intrinsic Equivalent**

VRSQRT28PD \_\_m512d \_\_mm512\_rsqrt28\_round\_pd(\_\_m512d a, int sae);

VRSQRT28PD \_\_m512d \_\_mm512\_mask\_rsqrt28\_round\_pd(\_\_m512d s, \_\_mmask8 m, \_\_m512d a, int sae);

VRSQRT28PD \_\_m512d \_\_mm512\_maskz\_rsqrt28\_round\_pd(\_\_mmask8 m, \_\_m512d a, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Divide-by-zero

**Other Exceptions**

See Exceptions Type E2.

## VRSQRT28SD—Approximation to the Reciprocal Square Root of Scalar Double-Precision Floating-Point Value with Less Than $2^{-28}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EEX.NDS.LIG.66.0F38.W1 CD /r VRSQRT28SD xmm1 {k1}{z}, xmm2, xmm3/m64 {sae}	A	V/V	AVX512ER	Computes approximate reciprocal square root ( $<2^{-28}$ relative error) of the scalar double-precision floating-point value from xmm3/m64 and stores result in xmm1 with writemask k1. Also, upper double-precision floating-point value (bits[127:64]) from xmm2 is copied to xmm1[127:64].

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	EEX.vvvv (r)	ModRM:r/m (r)	NA

### Description

Computes the reciprocal square root of the low float64 value in the second source operand (the third operand) and store the result to the destination operand (the first operand). The approximate reciprocal square root is evaluated with less than  $2^{-28}$  of maximum relative error. The result is written into the low float64 element of xmm1 according to the writemask k1. Bits 127:64 of the destination is copied from the corresponding bits of the first source operand (the second operand).

If any source element is NaN, the quietized NaN source value is returned for that element. Negative (non-zero) source numbers, as well as  $-\infty$ , return the canonical NaN and set the Invalid Flag (#I).

A value of  $-0$  must return  $-\infty$  and set the DivByZero flags (#Z). Negative numbers should return NaN and set the Invalid flag (#I). Note however that the instruction flush input denormals to zero of the same sign, so negative denormals return  $-\infty$  and set the DivByZero flag.

The first source operand is an XMM register. The second source operand is an XMM register or a 64-bit memory location. The destination operand is a XMM register.

A numerically exact implementation of VRSQRT28xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VRSQRT28SD (EVEX encoded versions)

```

IF k1[0] OR *no writemask* THEN
    DEST[63: 0] ← (1.0/ SQRT(SRC[63: 0]));
ELSE
    IF *merging-masking*           ; merging-masking
    THEN *DEST[63: 0] remains unchanged*
    ELSE                             ; zeroing-masking
        DEST[63: 0] ← 0
    FI;
FI;
ENDFOR;
DEST[127:64] ← SRC1[127: 64]
DEST[MAXVL-1:128] ← 0

```

**Table 6-8. VRSQRT28SD Special Cases**

Input value	Result value	Comments
NAN	QNAN(input)	If (SRC = SNaN) then #I
$X = 2^{-2n}$	$2^n$	
$X < 0$	QNAN_Indefinite	Including -INF
$X = -0$ or negative denormal	-INF	#Z
$X = +0$ or positive denormal	+INF	#Z
$X = +INF$	+0	

**Intel C/C++ Compiler Intrinsic Equivalent**

VRSQRT28SD \_\_m128d \_\_mm\_rsqrt28\_round\_sd(\_\_m128d a, \_\_m128d b, int sae);

VRSQRT28SD \_\_m128d \_\_mm\_mask\_rsqrt28\_round\_pd(\_\_m128d s, \_\_mmask8 m, \_\_m128d a, \_\_m128d b, int sae);

VRSQRT28SD \_\_m128d \_\_mm\_maskz\_rsqrt28\_round\_pd(\_\_mmask8 m, \_\_m128d a, \_\_m128d b, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Divide-by-zero

**Other Exceptions**

See Exceptions Type E3.

## VRSQRT28PS—Approximation to the Reciprocal Square Root of Packed Single-Precision Floating-Point Values with Less Than $2^{-28}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W0 CC /r VRSQRT28PS zmm1 {k1}{z}, zmm2/m512/m32bcst {sae}	A	V/V	AVX512ER	Computes approximations to the Reciprocal square root ( $<2^{-28}$ relative error) of the packed single-precision floating-point values from zmm2/m512/m32bcst and stores result in zmm1 with writemask k1.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

### Description

Computes the reciprocal square root of the float32 values in the source operand (the second operand) and store the results to the destination operand (the first operand). The approximate reciprocal is evaluated with less than  $2^{-28}$  of maximum relative error prior to final rounding. The final results is rounded to  $< 2^{-23}$  relative error before written to the destination.

If any source element is NaN, the quietized NaN source value is returned for that element. Negative (non-zero) source numbers, as well as  $-\infty$ , return the canonical NaN and set the Invalid Flag (#I).

A value of  $-0$  must return  $-\infty$  and set the DivByZero flags (#Z). Negative numbers should return NaN and set the Invalid flag (#I). Note however that the instruction flush input denormals to zero of the same sign, so negative denormals return  $-\infty$  and set the DivByZero flag.

The source operand is a ZMM register, a 512-bit memory location, or a 512-bit vector broadcasted from a 32-bit memory location. The destination operand is a ZMM register, conditionally updated using writemask k1.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

A numerically exact implementation of VRSQRT28xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2>.

### Operation

#### VRSQRT28PS (EVEX encoded versions)

(KL, VL) = (16, 512)

FOR j  $\leftarrow$  0 TO KL-1

  i  $\leftarrow$  j \* 32

  IF k1[j] OR \*no writemask\* THEN

    IF (EVEX.b = 1) AND (SRC \*is memory\*)

      THEN DEST[i+31:i]  $\leftarrow$  (1.0/ SQRT(SRC[31:0]));

      ELSE DEST[i+31:i]  $\leftarrow$  (1.0/ SQRT(SRC[i+31:i]));

    FI;

  ELSE

    IF \*merging-masking\* ; merging-masking

      THEN \*DEST[i+31:i] remains unchanged\*

    ELSE ; zeroing-masking

      DEST[i+31:i]  $\leftarrow$  0

    FI;

  FI;

ENDFOR;

Table 6-9. VRSQRT28PS Special Cases

Input value	Result value	Comments
NAN	QNAN(input)	If (SRC = SNaN) then #I
$X = 2^{-2n}$	$2^n$	
$X < 0$	QNAN_Indefinite	Including -INF
$X = -0$ or negative denormal	-INF	#Z
$X = +0$ or positive denormal	+INF	#Z
$X = +INF$	+0	

**Intel C/C++ Compiler Intrinsic Equivalent**

VRSQRT28PS \_\_m512 \_\_mm512\_rsqrt28\_round\_ps(\_\_m512 a, int sae);

VRSQRT28PS \_\_m512 \_\_mm512\_mask\_rsqrt28\_round\_ps(\_\_m512 s, \_\_mmask16 m, \_\_m512 a, int sae);

VRSQRT28PS \_\_m512 \_\_mm512\_maskz\_rsqrt28\_round\_ps(\_\_mmask16 m, \_\_m512 a, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Divide-by-zero

**Other Exceptions**

See Exceptions Type E2.

## VRSQRT28SS—Approximation to the Reciprocal Square Root of Scalar Single-Precision Floating-Point Value with Less Than $2^{-28}$ Relative Error

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.NDS.LIG.66.0F38.W0 CD /r VRSQRT28SS xmm1 {k1}{z}, xmm2, xmm3/m32 {sae}	A	V/V	AVX512ER	Computes approximate reciprocal square root ( $<2^{-28}$ relative error) of the scalar single-precision floating-point value from xmm3/m32 and stores result in xmm1 with writemask k1. Also, upper 3 single-precision floating-point value (bits[127:32]) from xmm2 is copied to xmm1[127:32].

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

### Description

Computes the reciprocal square root of the low float32 value in the second source operand (the third operand) and store the result to the destination operand (the first operand). The approximate reciprocal square root is evaluated with less than  $2^{-28}$  of maximum relative error prior to final rounding. The final result is rounded to  $< 2^{-23}$  relative error before written to the low float32 element of the destination according to the writemask k1. Bits 127:32 of the destination is copied from the corresponding bits of the first source operand (the second operand).

If any source element is NaN, the quietized NaN source value is returned for that element. Negative (non-zero) source numbers, as well as  $-\infty$ , return the canonical NaN and set the Invalid Flag (#I).

A value of  $-0$  must return  $-\infty$  and set the DivByZero flags (#Z). Negative numbers should return NaN and set the Invalid flag (#I). Note however that the instruction flush input denormals to zero of the same sign, so negative denormals return  $-\infty$  and set the DivByZero flag.

The first source operand is an XMM register. The second source operand is an XMM register or a 32-bit memory location. The destination operand is a XMM register.

A numerically exact implementation of VRSQRT28xx can be found at <https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vcrcp14-vrsqrt14-vcrcp28-vrsqrt28-vexp2>.

### Operation

#### VRSQRT28SS (EVEX encoded versions)

```

IF k1[0] OR *no writemask* THEN
    DEST[31: 0] ← (1.0/ SQRT(SRC[31: 0]));
ELSE
    IF *merging-masking*           ; merging-masking
    THEN *DEST[31: 0] remains unchanged*
    ELSE                             ; zeroing-masking
        DEST[31: 0] ← 0
    FI;
FI;
ENDFOR;
DEST[127:32] ← SRC1[127: 32]
DEST[MAXVL-1:128] ← 0

```



**Table 6-10. VRSQRT28SS Special Cases**

Input value	Result value	Comments
NAN	QNAN(input)	If (SRC = SNaN) then #I
$X = 2^{-2n}$	$2^n$	
$X < 0$	QNAN_Indefinite	Including -INF
$X = -0$ or negative denormal	-INF	#Z
$X = +0$ or positive denormal	+INF	#Z
$X = +INF$	+0	

**Intel C/C++ Compiler Intrinsic Equivalent**

VRSQRT28SS \_\_m128 \_\_mm\_rsqrt28\_round\_ss(\_\_m128 a, \_\_m128 b, int sae);

VRSQRT28SS \_\_m128 \_\_mm512\_mask\_rsqrt28\_round\_ss(\_\_m128 s, \_\_mmask8 m, \_\_m128 a, \_\_m128 b, int sae);

VRSQRT28SS \_\_m128 \_\_mm512\_maskz\_rsqrt28\_round\_ss(\_\_mmask8 m, \_\_m128 a, \_\_m128 b, int sae);

**SIMD Floating-Point Exceptions**

Invalid (if SNaN input), Divide-by-zero

**Other Exceptions**

See Exceptions Type E3.

## VSCATTERPFODPS/VSCATTERPFOQPS/VSCATTERPFODPD/VSCATTERPFOQPD—Sparse Prefetch Packed SP/DP Data Values with Signed Dword, Signed Qword Indices Using T0 Hint with Intent to Write

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W0 C6 /5 /vsib VSCATTERPFODPS vm32z {k1}	A	V/V	AVX512PF	Using signed dword indices, prefetch sparse byte memory locations containing single-precision data using writemask k1 and T0 hint with intent to write.
EVEX.512.66.0F38.W0 C7 /5 /vsib VSCATTERPFOQPS vm64z {k1}	A	V/V	AVX512PF	Using signed qword indices, prefetch sparse byte memory locations containing single-precision data using writemask k1 and T0 hint with intent to write.
EVEX.512.66.0F38.W1 C6 /5 /vsib VSCATTERPFODPD vm32y {k1}	A	V/V	AVX512PF	Using signed dword indices, prefetch sparse byte memory locations containing double-precision data using writemask k1 and T0 hint with intent to write.
EVEX.512.66.0F38.W1 C7 /5 /vsib VSCATTERPFOQPD vm64z {k1}	A	V/V	AVX512PF	Using signed qword indices, prefetch sparse byte memory locations containing double-precision data using writemask k1 and T0 hint with intent to write.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	BaseReg (R): VSIB:base, VectorReg(R): VSIB:index	NA	NA	NA

### Description

The instruction conditionally prefetches up to sixteen 32-bit or eight 64-bit integer byte data elements. The elements are specified via the VSIB (i.e., the index register is an zmm, holding packed indices). Elements will only be prefetched if their corresponding mask bit is one.

cache lines will be brought into exclusive state (RFO) specified by a locality hint (T0):

- T0 (temporal data)—prefetch data into the first level cache.

[PS data] For dword indices, the instruction will prefetch sixteen memory locations. For qword indices, the instruction will prefetch eight values.

[PD data] For dword and qword indices, the instruction will prefetch eight memory locations.

Note that:

- (1) The prefetches may happen in any order (or not at all). The instruction is a hint.
- (2) The mask is left unchanged.
- (3) Not valid with 16-bit effective addresses. Will deliver a #UD fault.
- (4) No FP nor memory faults may be produced by this instruction.
- (5) Prefetches do not handle cache line splits
- (6) A #UD is signaled if the memory operand is encoded without the SIB byte.

### Operation

BASE\_ADDR stands for the memory operand base address (a GPR); may not exist

VINDEX stands for the memory operand vector of indices (a vector register)

SCALE stands for the memory operand scalar (1, 2, 4 or 8)

DISP is the optional 1, 2 or 4 byte displacement

PREFETCH(mem, Level, State) Prefetches a byte memory location pointed by 'mem' into the cache level specified by 'Level'; a request for exclusive/ownership is done if 'State' is 1. Note that the memory location ignore cache line splits. This operation is considered a hint for the processor and may be skipped depending on implementation.

**VSCATTERPFODPS (EVEX encoded version)**

```
(KL, VL) = (16, 512)
FOR j ← 0 TO KL-1
  i ← j * 32
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[j+31:i]) * SCALE + DISP], Level=0, RFO = 1)
  FI;
ENDFOR
```

**VSCATTERPFODPD (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  k ← j * 32
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[k+31:k]) * SCALE + DISP], Level=0, RFO = 1)
  FI;
ENDFOR
```

**VSCATTERPFOQPS (EVEX encoded version)**

```
(KL, VL) = (8, 256)
FOR j ← 0 TO KL-1
  i ← j * 64
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[j+63:i]) * SCALE + DISP], Level=0, RFO = 1)
  FI;
ENDFOR
```

**VSCATTERPFOQPD (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  k ← j * 64
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[k+63:k]) * SCALE + DISP], Level=0, RFO = 1)
  FI;
ENDFOR
```

**Intel C/C++ Compiler Intrinsic Equivalent**

```
VSCATTERPFODPD void _mm512_prefetch_i32scatter_pd(void *base, __m256i vdx, int scale, int hint);
VSCATTERPFODPD void _mm512_mask_prefetch_i32scatter_pd(void *base, __mmask8 m, __m256i vdx, int scale, int hint);
VSCATTERPFODPS void _mm512_prefetch_i32scatter_ps(void *base, __m512i vdx, int scale, int hint);
VSCATTERPFODPS void _mm512_mask_prefetch_i32scatter_ps(void *base, __mmask16 m, __m512i vdx, int scale, int hint);
VSCATTERPFOQPD void _mm512_prefetch_i64scatter_pd(void *base, __m512i vdx, int scale, int hint);
VSCATTERPFOQPD void _mm512_mask_prefetch_i64scatter_pd(void *base, __mmask8 m, __m512i vdx, int scale, int hint);
VSCATTERPFOQPS void _mm512_prefetch_i64scatter_ps(void *base, __m512i vdx, int scale, int hint);
VSCATTERPFOQPS void _mm512_mask_prefetch_i64scatter_ps(void *base, __mmask8 m, __m512i vdx, int scale, int hint);
```

**SIMD Floating-Point Exceptions**

None

**Other Exceptions**

See Exceptions Type E12NP.

## VSCATTERPF1DPS/VSCATTERPF1QPS/VSCATTERPF1DPD/VSCATTERPF1QPD—Sparse Prefetch Packed SP/DP Data Values with Signed Dword, Signed Qword Indices Using T1 Hint with Intent to Write

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F38.W0 C6 /6 /vsib VSCATTERPF1DPS vm32z {k1}	A	V/V	AVX512PF	Using signed dword indices, prefetch sparse byte memory locations containing single-precision data using writemask k1 and T1 hint with intent to write.
EVEX.512.66.0F38.W0 C7 /6 /vsib VSCATTERPF1QPS vm64z {k1}	A	V/V	AVX512PF	Using signed qword indices, prefetch sparse byte memory locations containing single-precision data using writemask k1 and T1 hint with intent to write.
EVEX.512.66.0F38.W1 C6 /6 /vsib VSCATTERPF1DPD vm32y {k1}	A	V/V	AVX512PF	Using signed dword indices, prefetch sparse byte memory locations containing double-precision data using writemask k1 and T1 hint with intent to write.
EVEX.512.66.0F38.W1 C7 /6 /vsib VSCATTERPF1QPD vm64z {k1}	A	V/V	AVX512PF	Using signed qword indices, prefetch sparse byte memory locations containing double-precision data using writemask k1 and T1 hint with intent to write.

### Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	BaseReg (R): VSIB:base, VectorReg(R): VSIB:index	NA	NA	NA

### Description

The instruction conditionally prefetches up to sixteen 32-bit or eight 64-bit integer byte data elements. The elements are specified via the VSIB (i.e., the index register is an zmm, holding packed indices). Elements will only be prefetched if their corresponding mask bit is one.

cache lines will be brought into exclusive state (RFO) specified by a locality hint (T1):

- T1 (temporal data)—prefetch data into the second level cache.

[PS data] For dword indices, the instruction will prefetch sixteen memory locations. For qword indices, the instruction will prefetch eight values.

[PD data] For dword and qword indices, the instruction will prefetch eight memory locations.

Note that:

- (1) The prefetches may happen in any order (or not at all). The instruction is a hint.
- (2) The mask is left unchanged.
- (3) Not valid with 16-bit effective addresses. Will deliver a #UD fault.
- (4) No FP nor memory faults may be produced by this instruction.
- (5) Prefetches do not handle cache line splits
- (6) A #UD is signaled if the memory operand is encoded without the SIB byte.

### Operation

BASE\_ADDR stands for the memory operand base address (a GPR); may not exist

VINDEX stands for the memory operand vector of indices (a vector register)

SCALE stands for the memory operand scalar (1, 2, 4 or 8)

DISP is the optional 1, 2 or 4 byte displacement

PREFETCH(mem, Level, State) Prefetches a byte memory location pointed by 'mem' into the cache level specified by 'Level'; a request for exclusive/ownership is done if 'State' is 1. Note that the memory location ignore cache line splits. This operation is considered a hint for the processor and may be skipped depending on implementation.

**VSCATTERPF1DPS (EVEX encoded version)**

```
(KL, VL) = (16, 512)
FOR j ← 0 TO KL-1
  i ← j * 32
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[j+31:i]) * SCALE + DISP], Level=1, RFO = 1)
  FI;
ENDFOR
```

**VSCATTERPF1DPD (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  k ← j * 32
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[k+31:k]) * SCALE + DISP], Level=1, RFO = 1)
  FI;
ENDFOR
```

**VSCATTERPF1QPS (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[j+63:i]) * SCALE + DISP], Level=1, RFO = 1)
  FI;
ENDFOR
```

**VSCATTERPF1QPD (EVEX encoded version)**

```
(KL, VL) = (8, 512)
FOR j ← 0 TO KL-1
  i ← j * 64
  k ← j * 64
  IF k1[j]
    Prefetch( [BASE_ADDR + SignExtend(VINDEX[k+63:k]) * SCALE + DISP], Level=1, RFO = 1)
  FI;
ENDFOR
```

**Intel C/C++ Compiler Intrinsic Equivalent**

```
VSCATTERPF1DPD void __mm512_prefetch_i32scatter_pd(void *base, __m256i vdx, int scale, int hint);
VSCATTERPF1DPD void __mm512_mask_prefetch_i32scatter_pd(void *base, __mmask8 m, __m256i vdx, int scale, int hint);
VSCATTERPF1DPS void __mm512_prefetch_i32scatter_ps(void *base, __m512i vdx, int scale, int hint);
VSCATTERPF1DPS void __mm512_mask_prefetch_i32scatter_ps(void *base, __mmask16 m, __m512i vdx, int scale, int hint);
VSCATTERPF1QPD void __mm512_prefetch_i64scatter_pd(void *base, __m512i vdx, int scale, int hint);
VSCATTERPF1QPD void __mm512_mask_prefetch_i64scatter_pd(void *base, __mmask8 m, __m512i vdx, int scale, int hint);
VSCATTERPF1QPS void __mm512_prefetch_i64scatter_ps(void *base, __m512i vdx, int scale, int hint);
VSCATTERPF1QPS void __mm512_mask_prefetch_i64scatter_ps(void *base, __mmask8 m, __m512i vdx, int scale, int hint);
```

**SIMD Floating-Point Exceptions**

None

**Other Exceptions**

See Exceptions Type E12NP.



## 9. 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*.

-----  
Changes to this chapter: Update in naming flags in control registers.

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

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

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

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

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

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

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

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

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

All Intel 64 and IA-32 processors enter real-address mode following a power-up or reset (see Chapter 9, "Processor Management and Initialization"). Software then initiates the switch from real-address mode to protected mode. If IA-32e mode operation is desired, software also initiates a switch from protected mode to IA-32e mode.

## 2.1 OVERVIEW OF THE SYSTEM-LEVEL ARCHITECTURE

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

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



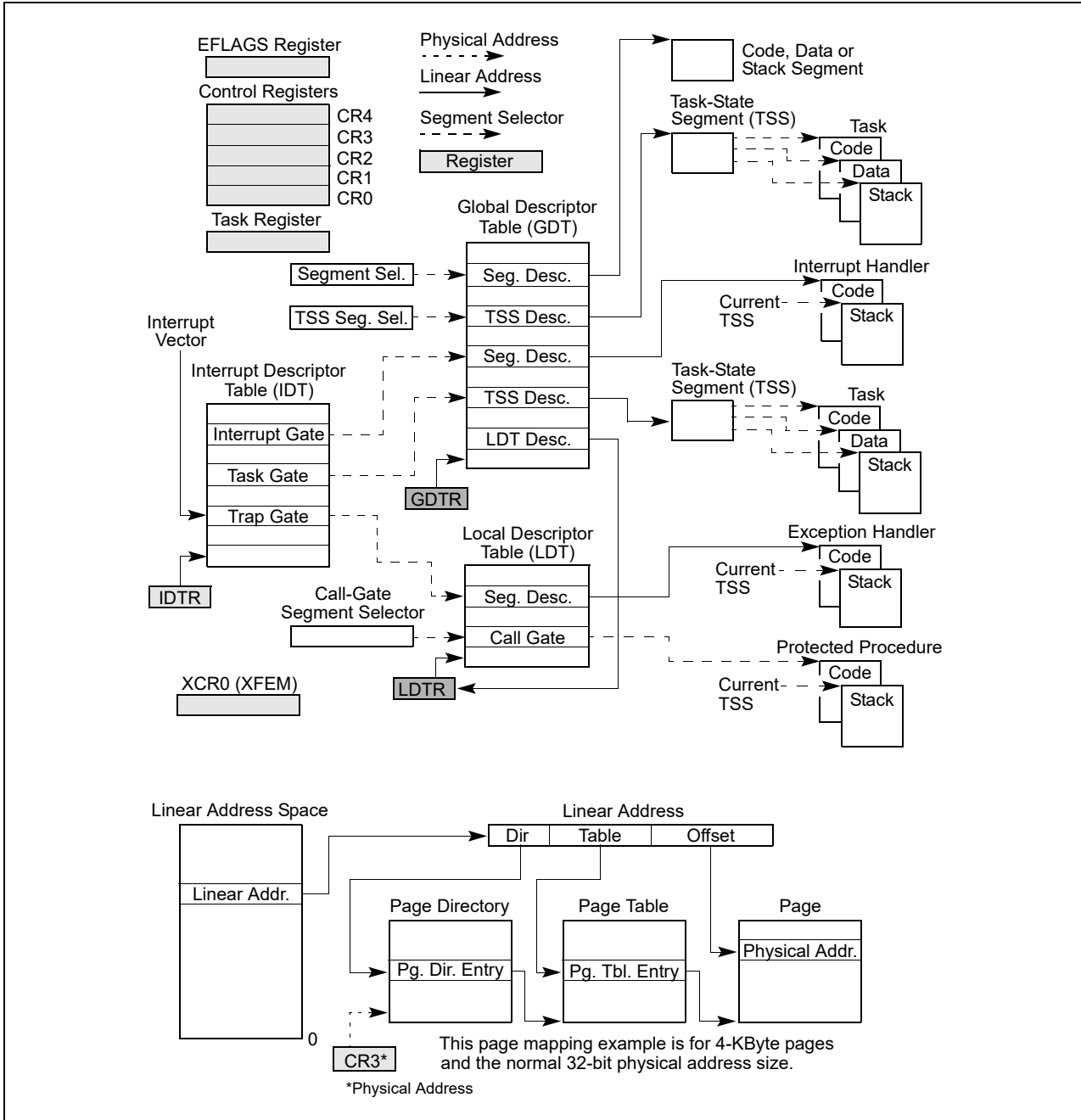


Figure 2-1. IA-32 System-Level Registers and Data Structures

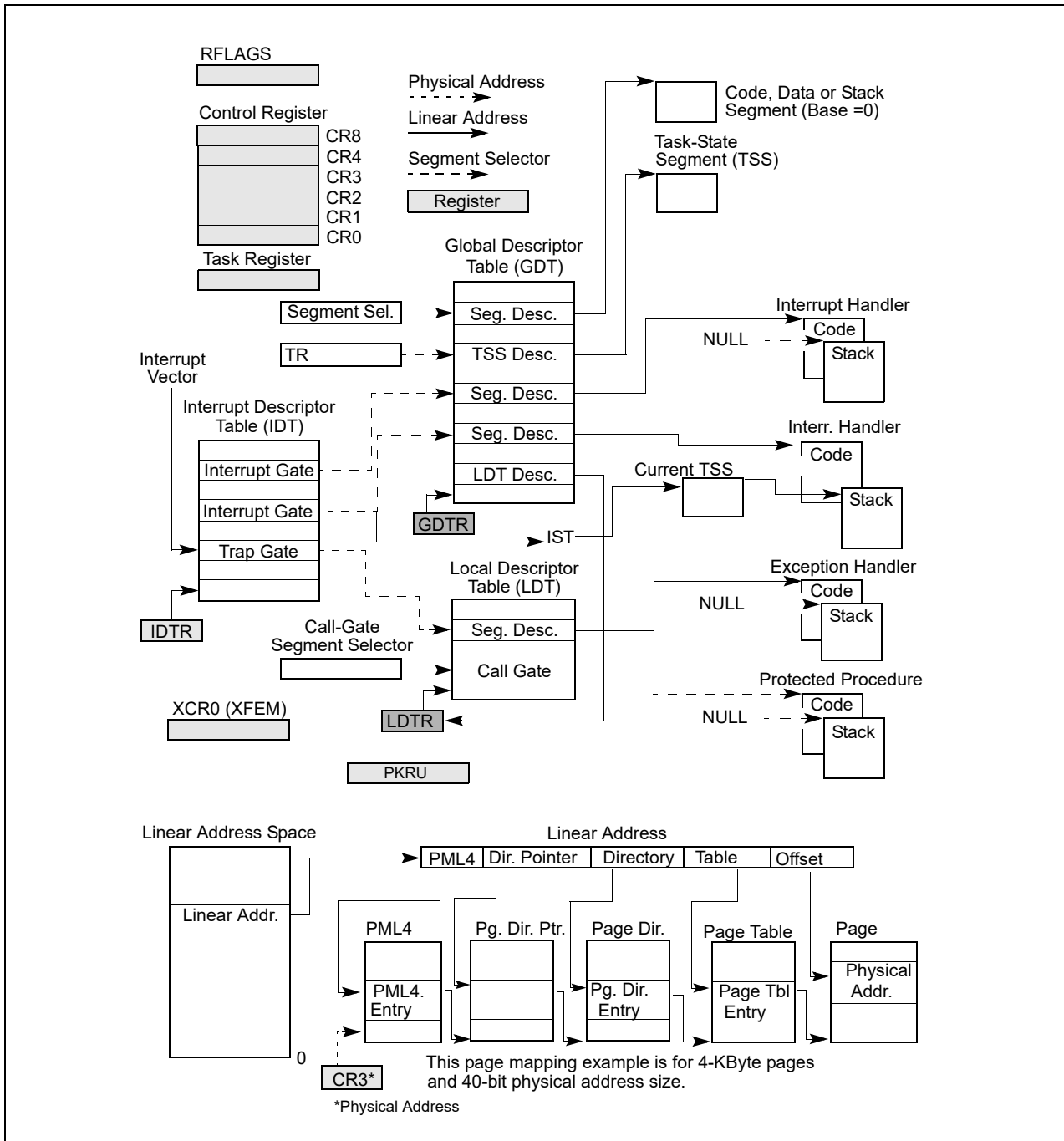


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode

### 2.1.1 Global and Local Descriptor Tables

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

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

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

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

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

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

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

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

## 2.1.2 System Segments, Segment Descriptors, and Gates

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

The architecture also defines a set of special descriptors called gates (call gates, interrupt gates, trap gates, and task gates). These provide protected gateways to system procedures and handlers that may operate at a different privilege level than application programs and most procedures. For example, a CALL to a call gate can provide access to a procedure in a code segment that is at the same or a numerically lower privilege level (more privileged) than the current code segment. To access a procedure through a call gate, the calling procedure<sup>1</sup> supplies the selector for the call gate. The processor then performs an access rights check on the call gate, comparing the CPL with the privilege level of the call gate and the destination code segment pointed to by the call gate.

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

### 2.1.2.1 Gates in IA-32e Mode

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

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

---

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

### 2.1.3 Task-State Segments and Task Gates

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

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

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

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

#### 2.1.3.1 Task-State Segments in IA-32e Mode

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

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

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

The task register is expanded to hold 64-bit base addresses in IA-32e mode. See also: Section 7.7, “Task Management in 64-bit Mode.”

### 2.1.4 Interrupt and Exception Handling

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

Gate descriptors in the IDT can be interrupt, trap, or task gate descriptors. To access an interrupt or exception handler, the processor first receives an interrupt vector from internal hardware, an external interrupt controller, or from software by means of an INT, INTO, INT 3, or BOUND instruction. The interrupt vector provides an index into the IDT. If the selected gate descriptor is an interrupt gate or a trap gate, the associated handler procedure is accessed in a manner similar to calling a procedure through a call gate. If the descriptor is a task gate, the handler is accessed through a task switch.

#### 2.1.4.1 Interrupt and Exception Handling IA-32e Mode

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

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

## 2.1.5 Memory Management

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

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

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

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

### 2.1.5.1 Memory Management in IA-32e Mode

In IA-32e mode, physical memory pages are managed by a set of system data structures. In compatibility mode and 64-bit mode, four levels of system data structures are used. These include:

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

## 2.1.6 System Registers

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

- The system flags and IOPL field in the EFLAGS register control task and mode switching, interrupt handling, instruction tracing, and access rights. See also: Section 2.3, "System Flags and Fields in the EFLAGS Register."
- The control registers (CR0, CR2, CR3, and CR4) contain a variety of flags and data fields for controlling system-level operations. Other flags in these registers are used to indicate support for specific processor capabilities within the operating system or executive. See also: Section 2.5, "Control Registers" and Section 2.6, "Extended Control Registers (Including XCR0)."
- The debug registers (not shown in Figure 2-1) allow the setting of breakpoints for use in debugging programs and systems software. See also: Chapter 17, "Debug, Branch Profile, TSC, and Resource Monitoring Features."
- The GDTR, LDTR, and IDTR registers contain the linear addresses and sizes (limits) of their respective tables. See also: Section 2.4, "Memory-Management Registers."
- The task register contains the linear address and size of the TSS for the current task. See also: Section 2.4, "Memory-Management Registers."
- Model-specific registers (not shown in Figure 2-1).

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

The number and function of these registers varies among different members of the Intel 64 and IA-32 processor families. See also: Section 9.4, "Model-Specific Registers (MSRs)," and Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*.

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

### 2.1.6.1 System Registers in IA-32e Mode

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

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

On systems that support IA-32e mode, the extended feature enable register (IA32\_EFER) is available. This model-specific register controls activation of IA-32e mode and other IA-32e mode operations. In addition, there are several model-specific registers that govern IA-32e mode instructions:

- IA32\_KERNEL\_GS\_BASE — Used by SWAPGS instruction.
- IA32\_LSTAR — Used by SYSCALL instruction.
- IA32\_FMASK — Used by SYSCALL instruction.
- IA32\_STAR — Used by SYSCALL and SYSRET instruction.

### 2.1.7 Other System Resources

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

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

Performance-monitoring counters are event counters that can be programmed to count processor events such as the number of instructions decoded, the number of interrupts received, or the number of cache loads. See also: Chapter 19, “Performance Monitoring Events.”

The processor provides several internal caches and buffers. The caches are used to store both data and instructions. The buffers are used to store things like decoded addresses to system and application segments and write operations waiting to be performed. See also: Chapter 11, “Memory Cache Control.”

## 2.2 MODES OF OPERATION

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

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

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

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

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

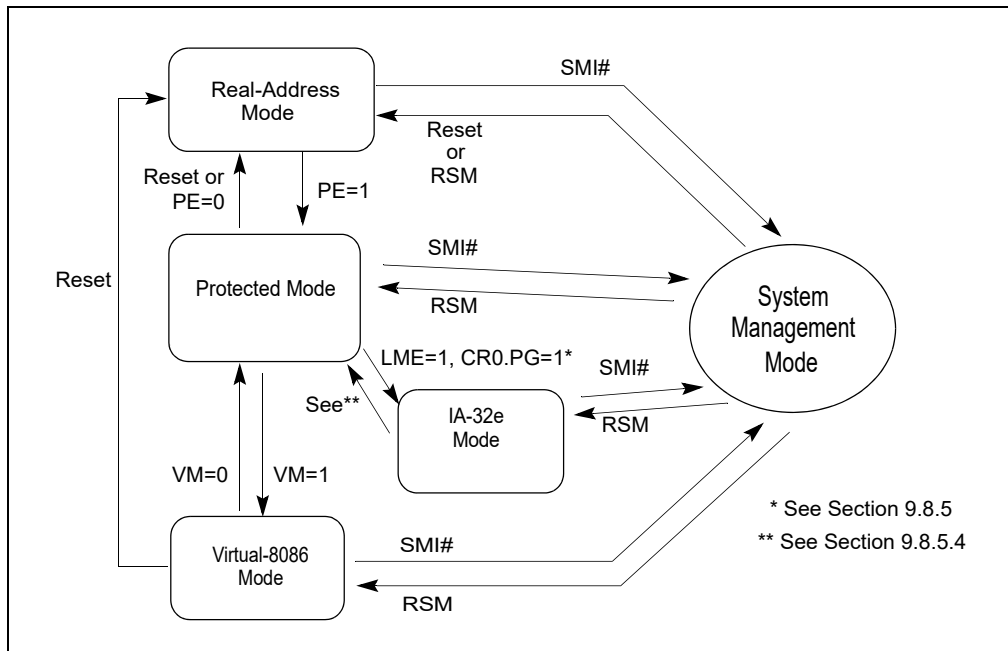


Figure 2-3. Transitions Among the Processor’s Operating Modes

The processor is placed in real-address mode following power-up or a reset. The PE flag in control register CR0 then controls whether the processor is operating in real-address or protected mode. See also: Section 9.9, “Mode Switching.” and Section 4.1.2, “Paging-Mode Enabling.”

The VM flag in the EFLAGS register determines whether the processor is operating in protected mode or virtual-8086 mode. Transitions between protected mode and virtual-8086 mode are generally carried out as part of a task switch or a return from an interrupt or exception handler. See also: Section 20.2.5, “Entering Virtual-8086 Mode.”

The LMA bit (IA32\_EFER.LMA[bit 10]) determines whether the processor is operating in IA-32e mode. When running in IA-32e mode, 64-bit or compatibility sub-mode operation is determined by CS.L bit of the code segment. The processor enters into IA-32e mode from protected mode by enabling paging and setting the LME bit (IA32\_EFER.LME[bit 8]). See also: Chapter 9, “Processor Management and Initialization.”

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

## 2.2.1 Extended Feature Enable Register

The IA32\_EFER MSR provides several fields related to IA-32e mode enabling and operation. It also provides one field that relates to page-access right modification (see Section 4.6, “Access Rights”). The layout of the IA32\_EFER MSR is shown in Figure 2-4.

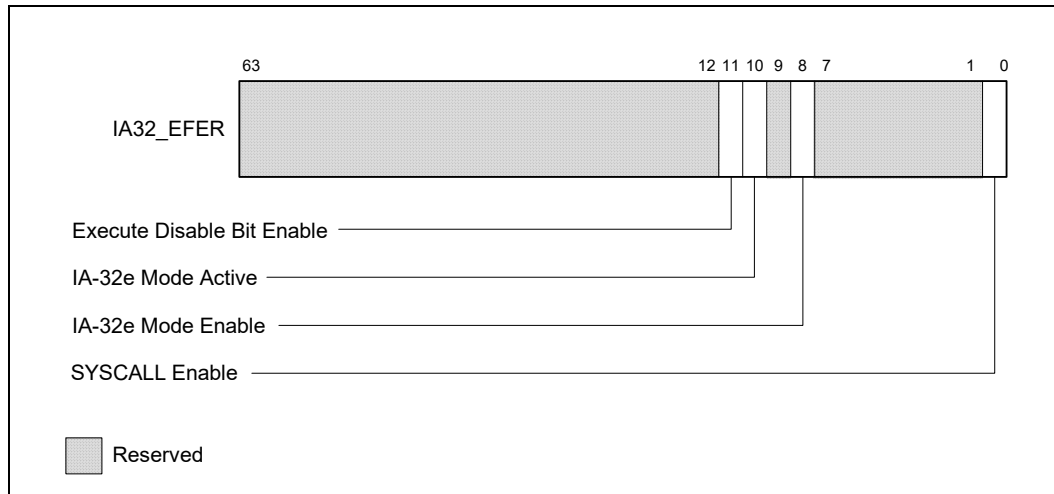


Figure 2-4. IA32\_EFER MSR Layout

Table 2-1. IA32\_EFER MSR Information

Bit	Description
0	<b>SYSCALL Enable: IA32_EFER.SCE (R/W)</b> Enables SYSCALL/SYSRET instructions in 64-bit mode.
7:1	Reserved.
8	<b>IA-32e Mode Enable: IA32_EFER.LME (R/W)</b> Enables IA-32e mode operation.
9	Reserved.
10	<b>IA-32e Mode Active: IA32_EFER.LMA (R)</b> Indicates IA-32e mode is active when set.
11	<b>Execute Disable Bit Enable: IA32_EFER.NXE (R/W)</b> Enables page access restriction by preventing instruction fetches from PAE pages with the XD bit set (See Section 4.6).
63:12	Reserved.

## 2.3 SYSTEM FLAGS AND FIELDS IN THE EFLAGS REGISTER

The system flags and IOPL field of the EFLAGS register control I/O, maskable hardware interrupts, debugging, task switching, and the virtual-8086 mode (see Figure 2-5). Only privileged code (typically operating system or executive code) should be allowed to modify these bits.

The system flags and IOPL are:

**TF**     **Trap (bit 8)** — Set to enable single-step mode for debugging; clear to disable single-step mode. In single-step mode, the processor generates a debug exception after each instruction. This allows the execution state of a program to be inspected after each instruction. If an application program sets the TF flag using a



POPF, POPFD, or IRET instruction, a debug exception is generated after the instruction that follows the POPF, POPFD, or IRET.

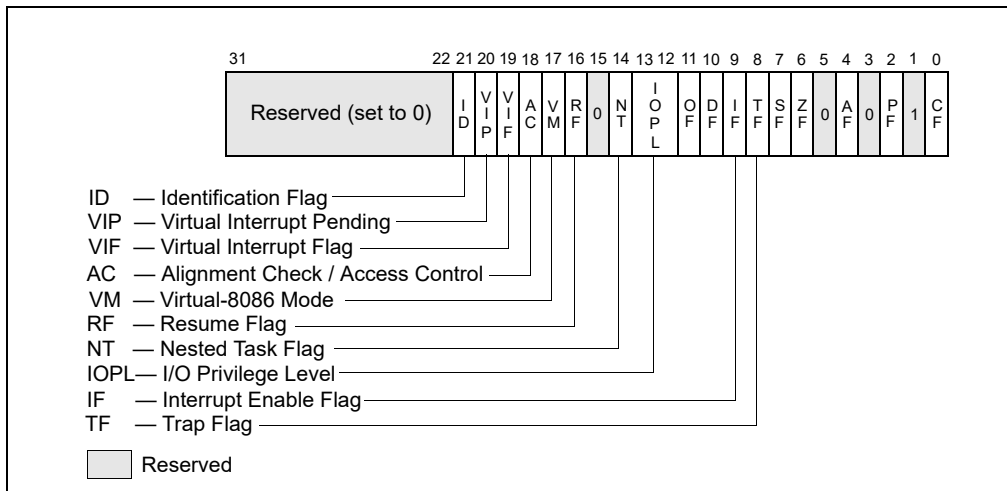


Figure 2-5. System Flags in the EFLAGS Register

**IF** **Interrupt enable (bit 9)** — Controls the response of the processor to maskable hardware interrupt requests (see also: Section 6.3.2, “Maskable Hardware Interrupts”). The flag is set to respond to maskable hardware interrupts; cleared to inhibit maskable hardware interrupts. The IF flag does not affect the generation of exceptions or nonmaskable interrupts (NMI interrupts). The CPL, IOPL, and the state of the VME flag in control register CR4 determine whether the IF flag can be modified by the CLI, STI, POPF, POPFD, and IRET.

**IOPL** **I/O privilege level field (bits 12 and 13)** — Indicates the I/O privilege level (IOPL) of the currently running program or task. The CPL of the currently running program or task must be less than or equal to the IOPL to access the I/O address space. The POPF and IRET instructions can modify this field only when operating at a CPL of 0.

The IOPL is also one of the mechanisms that controls the modification of the IF flag and the handling of interrupts in virtual-8086 mode when virtual mode extensions are in effect (when CR4.VME = 1). See also: Chapter 18, “Input/Output,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.

**NT** **Nested task (bit 14)** — Controls the chaining of interrupted and called tasks. The processor sets this flag on calls to a task initiated with a CALL instruction, an interrupt, or an exception. It examines and modifies this flag on returns from a task initiated with the IRET instruction. The flag can be explicitly set or cleared with the POPF/POPFD instructions; however, changing to the state of this flag can generate unexpected exceptions in application programs.

See also: Section 7.4, “Task Linking.”

**RF** **Resume (bit 16)** — Controls the processor’s response to instruction-breakpoint conditions. When set, this flag temporarily disables debug exceptions (#DB) from being generated for instruction breakpoints (although other exception conditions can cause an exception to be generated). When clear, instruction breakpoints will generate debug exceptions.

The primary function of the RF flag is to allow the restarting of an instruction following a debug exception that was caused by an instruction breakpoint condition. Here, debug software must set this flag in the EFLAGS image on the stack just prior to returning to the interrupted program with IRETD (to prevent the instruction breakpoint from causing another debug exception). The processor then automatically clears this flag after the instruction returned to has been successfully executed, enabling instruction breakpoint faults again.

See also: Section 17.3.1.1, “Instruction-Breakpoint Exception Condition.”

**VM** **Virtual-8086 mode (bit 17)** — Set to enable virtual-8086 mode; clear to return to protected mode.

See also: Section 20.2.1, “Enabling Virtual-8086 Mode.”

- AC Alignment check or access control (bit 18)** — If the AM bit is set in the CR0 register, alignment checking of user-mode data accesses is enabled if and only if this flag is 1. An alignment-check exception is generated when reference is made to an unaligned operand, such as a word at an odd byte address or a doubleword at an address which is not an integral multiple of four. Alignment-check exceptions are generated only in user mode (privilege level 3). Memory references that default to privilege level 0, such as segment descriptor loads, do not generate this exception even when caused by instructions executed in user-mode.
- The alignment-check exception can be used to check alignment of data. This is useful when exchanging data with processors which require all data to be aligned. The alignment-check exception can also be used by interpreters to flag some pointers as special by misaligning the pointer. This eliminates overhead of checking each pointer and only handles the special pointer when used.
- If the SMAP bit is set in the CR4 register, explicit supervisor-mode data accesses to user-mode pages are allowed if and only if this bit is 1. See Section 4.6, “Access Rights.”
- VIF Virtual Interrupt (bit 19)** — Contains a virtual image of the IF flag. This flag is used in conjunction with the VIP flag. The processor only recognizes the VIF flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. (The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.)
- See also: Section 20.3.3.5, “Method 6: Software Interrupt Handling,” and Section 20.4, “Protected-Mode Virtual Interrupts.”
- VIP Virtual interrupt pending (bit 20)** — Set by software to indicate that an interrupt is pending; cleared to indicate that no interrupt is pending. This flag is used in conjunction with the VIF flag. The processor reads this flag but never modifies it. The processor only recognizes the VIP flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.
- See Section 20.3.3.5, “Method 6: Software Interrupt Handling,” and Section 20.4, “Protected-Mode Virtual Interrupts.”
- ID Identification (bit 21)** — The ability of a program or procedure to set or clear this flag indicates support for the CPUID instruction.

### 2.3.1 System Flags and Fields in IA-32e Mode

In 64-bit mode, the RFLAGS register expands to 64 bits with the upper 32 bits reserved. System flags in RFLAGS (64-bit mode) or EFLAGS (compatibility mode) are shown in Figure 2-5.

In IA-32e mode, the processor does not allow the VM bit to be set because virtual-8086 mode is not supported (attempts to set the bit are ignored). Also, the processor will not set the NT bit. The processor does, however, allow software to set the NT bit (note that an IRET causes a general protection fault in IA-32e mode if the NT bit is set).

In IA-32e mode, the SYSCALL/SYSCALL instructions have a programmable method of specifying which bits are cleared in RFLAGS/EFLAGS. These instructions save/restore EFLAGS/RFLAGS.

## 2.4 MEMORY-MANAGEMENT REGISTERS

The processor provides four memory-management registers (GDTR, LDTR, IDTR, and TR) that specify the locations of the data structures which control segmented memory management (see Figure 2-6). Special instructions are provided for loading and storing these registers.

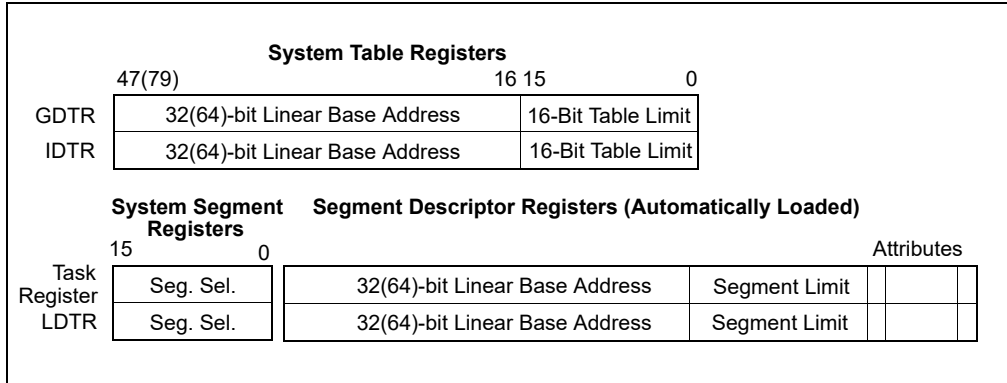


Figure 2-6. Memory Management Registers

### 2.4.1 Global Descriptor Table Register (GDTR)

The GDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and the 16-bit table limit for the GDT. The base address specifies the linear address of byte 0 of the GDT; the table limit specifies the number of bytes in the table.

The LGDT and SGDT instructions load and store the GDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. A new base address must be loaded into the GDTR as part of the processor initialization process for protected-mode operation.

See also: Section 3.5.1, "Segment Descriptor Tables."

### 2.4.2 Local Descriptor Table Register (LDTR)

The LDTR register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the LDT. The base address specifies the linear address of byte 0 of the LDT segment; the segment limit specifies the number of bytes in the segment. See also: Section 3.5.1, "Segment Descriptor Tables."

The LLDT and SLDT instructions load and store the segment selector part of the LDTR register, respectively. The segment that contains the LDT must have a segment descriptor in the GDT. When the LLDT instruction loads a segment selector in the LDTR: the base address, limit, and descriptor attributes from the LDT descriptor are automatically loaded in the LDTR.

When a task switch occurs, the LDTR is automatically loaded with the segment selector and descriptor for the LDT for the new task. The contents of the LDTR are not automatically saved prior to writing the new LDT information into the register.

On power up or reset of the processor, the segment selector and base address are set to the default value of 0 and the limit is set to 0FFFFH.

### 2.4.3 IDTR Interrupt Descriptor Table Register

The IDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and 16-bit table limit for the IDT. The base address specifies the linear address of byte 0 of the IDT; the table limit specifies the number of bytes in the table. The LIDT and SIDT instructions load and store the IDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. The base address and limit in the register can then be changed as part of the processor initialization process.

See also: Section 6.10, "Interrupt Descriptor Table (IDT)."

## 2.4.4 Task Register (TR)

The task register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the TSS of the current task. The selector references the TSS descriptor in the GDT. The base address specifies the linear address of byte 0 of the TSS; the segment limit specifies the number of bytes in the TSS. See also: Section 7.2.4, “Task Register.”

The LTR and STR instructions load and store the segment selector part of the task register, respectively. When the LTR instruction loads a segment selector in the task register, the base address, limit, and descriptor attributes from the TSS descriptor are automatically loaded into the task register. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH.

When a task switch occurs, the task register is automatically loaded with the segment selector and descriptor for the TSS for the new task. The contents of the task register are not automatically saved prior to writing the new TSS information into the register.

## 2.5 CONTROL REGISTERS

Control registers (CR0, CR1, CR2, CR3, and CR4; see Figure 2-7) determine operating mode of the processor and the characteristics of the currently executing task. These registers are 32 bits in all 32-bit modes and compatibility mode.

In 64-bit mode, control registers are expanded to 64 bits. The MOV CRn instructions are used to manipulate the register bits. Operand-size prefixes for these instructions are ignored. The following is also true:

- The control registers can be read and loaded (or modified) using the move-to-or-from-control-registers forms of the MOV instruction. In protected mode, the MOV instructions allow the control registers to be read or loaded (at privilege level 0 only). This restriction means that application programs or operating-system procedures (running at privilege levels 1, 2, or 3) are prevented from reading or loading the control registers.
- Bits 63:32 of CR0 and CR4 are reserved and must be written with zeros. Writing a nonzero value to any of the upper 32 bits results in a general-protection exception, #GP(0).
- All 64 bits of CR2 are writable by software.
- Bits 51:40 of CR3 are reserved and must be 0.
- The MOV CRn instructions do not check that addresses written to CR2 and CR3 are within the linear-address or physical-address limitations of the implementation.
- Register CR8 is available in 64-bit mode only.

The control registers are summarized below, and each architecturally defined control field in these control registers is described individually. In Figure 2-7, the width of the register in 64-bit mode is indicated in parenthesis (except for CR0).

- **CR0** — Contains system control flags that control operating mode and states of the processor.
- **CR1** — Reserved.
- **CR2** — Contains the page-fault linear address (the linear address that caused a page fault).
- **CR3** — Contains the physical address of the base of the paging-structure hierarchy and two flags (PCD and PWT). Only the most-significant bits (less the lower 12 bits) of the base address are specified; the lower 12 bits of the address are assumed to be 0. The first paging structure must thus be aligned to a page (4-KByte) boundary. The PCD and PWT flags control caching of that paging structure in the processor’s internal data caches (they do not control TLB caching of page-directory information).

When using the physical address extension, the CR3 register contains the base address of the page-directory-pointer table. In IA-32e mode, the CR3 register contains the base address of the PML4 table.

See also: Chapter 4, “Paging.”

- **CR4** — Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities.

- **CR8** — Provides read and write access to the Task Priority Register (TPR). It specifies the priority threshold value that operating systems use to control the priority class of external interrupts allowed to interrupt the processor. This register is available only in 64-bit mode. However, interrupt filtering continues to apply in compatibility mode.

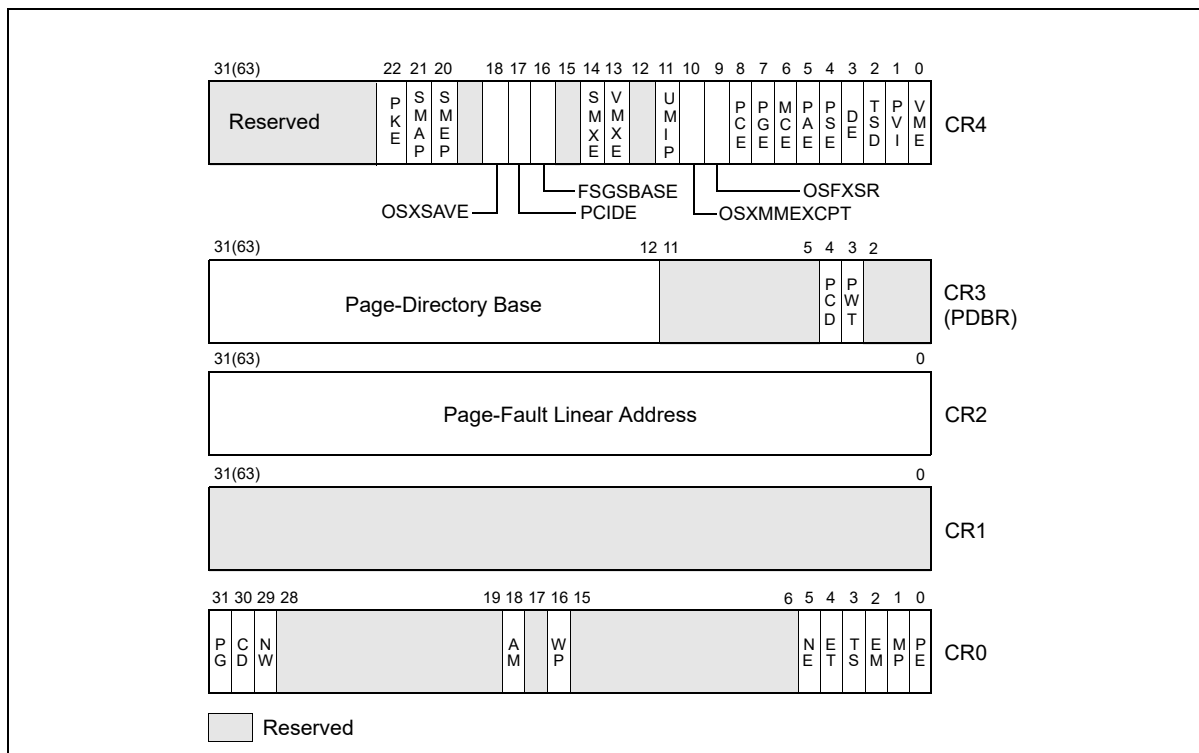


Figure 2-7. Control Registers

When loading a control register, reserved bits should always be set to the values previously read. The flags in control registers are:

**CR0.PG**

**Paging (bit 31 of CR0)** — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Chapter 4, "Paging."

On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.

**CR0.CD**

**Cache Disable (bit 30 of CR0)** — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor’s internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 11-5. To prevent the processor from accessing and updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.

See also: Section 11.5.3, "Preventing Caching," and Section 11.5, "Cache Control."

**CR0.NW**

**Not Write-through (bit 29 of CR0)** — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 11-5 for detailed information about the effect of the NW flag on caching for other settings of the CD and NW flags.

**CR0.AM**

**Alignment Mask (bit 18 of CR0)** — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

**CR0.WP**

**Write Protect (bit 16 of CR0)** — When set, inhibits supervisor-level procedures from writing into read-only pages; when clear, allows supervisor-level procedures to write into read-only pages (regardless of the U/S bit setting; see Section 4.1.3 and Section 4.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX.

**CR0.NE**

**Numeric Error (bit 5 of CR0)** — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.

The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting. Using FERR# and IGNNE# to handle floating-point exceptions is deprecated by modern operating systems; this non-native approach also limits newer processors to operate with one logical processor active.

See also: Section 8.7, "Handling x87 FPU Exceptions in Software" in Chapter 8, "Programming with the x87 FPU," and Appendix A, "EFLAGS Cross-Reference," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

**CR0.ET**

**Extension Type (bit 4 of CR0)** — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.

**CR0.TS**

**Task Switched (bit 3 of CR0)** — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context on a task switch to be delayed until an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction is actually executed by the new task. The processor sets this flag on every task switch and tests it when executing x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

- If the TS flag is set and the EM flag (bit 2 of CR0) is clear, a device-not-available exception (#NM) is raised prior to the execution of any x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction; with the exception of PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. See the paragraph below for the special case of the WAIT/FWAIT instructions.
- If the TS flag is set and the MP flag (bit 1 of CR0) and EM flag are clear, an #NM exception is not raised prior to the execution of an x87 FPU WAIT/FWAIT instruction.
- If the EM flag is set, the setting of the TS flag has no effect on the execution of x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

Table 2-2 shows the actions taken when the processor encounters an x87 FPU instruction based on the settings of the TS, EM, and MP flags. Table 12-1 and 13-1 show the actions taken when the processor encounters an MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction.

The processor does not automatically save the context of the x87 FPU, XMM, and MXCSR registers on a task switch. Instead, it sets the TS flag, which causes the processor to raise an #NM exception whenever it encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction in the instruction stream for the new task (with the exception of the instructions listed above).

The fault handler for the #NM exception can then be used to clear the TS flag (with the CLTS instruction) and save the context of the x87 FPU, XMM, and MXCSR registers. If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction, the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context is never saved.

**Table 2-2. Action Taken By x87 FPU Instructions for Different Combinations of EM, MP, and TS**

CRO Flags			x87 FPU Instruction Type	
EM	MP	TS	Floating-Point	WAIT/FWAIT
0	0	0	Execute	Execute.
0	0	1	#NM Exception	Execute.
0	1	0	Execute	Execute.
0	1	1	#NM Exception	#NM exception.
1	0	0	#NM Exception	Execute.
1	0	1	#NM Exception	Execute.
1	1	0	#NM Exception	Execute.
1	1	1	#NM Exception	#NM exception.

**CR0.EM**

**Emulation (bit 2 of CRO)** — Indicates that the processor does not have an internal or external x87 FPU when set; indicates an x87 FPU is present when clear. This flag also affects the execution of MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

When the EM flag is set, execution of an x87 FPU instruction generates a device-not-available exception (#NM). This flag must be set when the processor does not have an internal x87 FPU or is not connected to an external math coprocessor. Setting this flag forces all floating-point instructions to be handled by software emulation. Table 9-3 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the EM, MP, and TS flags.

Also, when the EM flag is set, execution of an MMX instruction causes an invalid-opcode exception (#UD) to be generated (see Table 12-1). Thus, if an IA-32 or Intel 64 processor incorporates MMX technology, the EM flag must be set to 0 to enable execution of MMX instructions.

Similarly for SSE/SSE2/SSE3/SSSE3/SSE4 extensions, when the EM flag is set, execution of most SSE/SSE2/SSE3/SSSE3/SSE4 instructions causes an invalid opcode exception (#UD) to be generated (see Table 13-1). If an IA-32 or Intel 64 processor incorporates the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the EM flag must be set to 0 to enable execution of these extensions. SSE/SSE2/SSE3/SSSE3/SSE4 instructions not affected by the EM flag include: PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

**CR0.MP**

**Monitor Coprocessor (bit 1 of CRO)** — Controls the interaction of the WAIT (or FWAIT) instruction with the TS flag (bit 3 of CRO). If the MP flag is set, a WAIT instruction generates a device-not-available exception (#NM) if the TS flag is also set. If the MP flag is clear, the WAIT instruction ignores the setting of the TS flag. Table 9-3 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the MP, EM, and TS flags.

**CR0.PE**

**Protection Enable (bit 0 of CRO)** — Enables protected mode when set; enables real-address mode when clear. This flag does not enable paging directly. It only enables segment-level protection. To enable paging, both the PE and PG flags must be set.

See also: Section 9.9, “Mode Switching.”

**CR3.PCD**

**Page-level Cache Disable (bit 4 of CR3)** — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 4.9, “Paging and Memory Typing”. This bit is not used if paging is disabled, with PAE paging, or with 4-level paging<sup>2</sup> if CR4.PCIDE=1.

2. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.



**CR3.PWT**

**Page-level Write-Through (bit 3 of CR3)** — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 4.9, “Paging and Memory Typing”. This bit is not used if paging is disabled, with PAE paging, or with 4-level paging if CR4.PCIDE=1.

**CR4.VME**

**Virtual-8086 Mode Extensions (bit 0 of CR4)** — Enables interrupt- and exception-handling extensions in virtual-8086 mode when set; disables the extensions when clear. Use of the virtual mode extensions can improve the performance of virtual-8086 applications by eliminating the overhead of calling the virtual-8086 monitor to handle interrupts and exceptions that occur while executing an 8086 program and, instead, redirecting the interrupts and exceptions back to the 8086 program’s handlers. It also provides hardware support for a virtual interrupt flag (VIF) to improve reliability of running 8086 programs in multi-tasking and multiple-processor environments.

See also: Section 20.3, “Interrupt and Exception Handling in Virtual-8086 Mode.”

**CR4.PVI**

**Protected-Mode Virtual Interrupts (bit 1 of CR4)** — Enables hardware support for a virtual interrupt flag (VIF) in protected mode when set; disables the VIF flag in protected mode when clear.

See also: Section 20.4, “Protected-Mode Virtual Interrupts.”

**CR4.TSD**

**Time Stamp Disable (bit 2 of CR4)** — Restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear. This bit also applies to the RDTSCP instruction if supported (if CPUID.80000001H:EDX[27] = 1).

**CR4.DE**

**Debugging Extensions (bit 3 of CR4)** — References to debug registers DR4 and DR5 cause an undefined opcode (#UD) exception to be generated when set; when clear, processor aliases references to registers DR4 and DR5 for compatibility with software written to run on earlier IA-32 processors.

See also: Section 17.2.2, “Debug Registers DR4 and DR5.”

**CR4.PSE**

**Page Size Extensions (bit 4 of CR4)** — Enables 4-MByte pages with 32-bit paging when set; restricts 32-bit paging to pages of 4 KBytes when clear.

See also: Section 4.3, “32-Bit Paging.”

**CR4.PAE**

**Physical Address Extension (bit 5 of CR4)** — When set, enables paging to produce physical addresses with more than 32 bits. When clear, restricts physical addresses to 32 bits. PAE must be set before entering IA-32e mode.

See also: Chapter 4, “Paging.”

**CR4.MCE**

**Machine-Check Enable (bit 6 of CR4)** — Enables the machine-check exception when set; disables the machine-check exception when clear.

See also: Chapter 15, “Machine-Check Architecture.”

**CR4.PGE**

**Page Global Enable (bit 7 of CR4)** — (Introduced in the P6 family processors.) Enables the global page feature when set; disables the global page feature when clear. The global page feature allows frequently used or shared pages to be marked as global to all users (done with the global flag, bit 8, in a page-directory or page-table entry). Global pages are not flushed from the translation-lookaside buffer (TLB) on a task switch or a write to register CR3.

When enabling the global page feature, paging must be enabled (by setting the PG flag in control register CR0) before the PGE flag is set. Reversing this sequence may affect program correctness, and processor performance will be impacted.

See also: Section 4.10, “Caching Translation Information.”



**CR4.PCE**

**Performance-Monitoring Counter Enable (bit 8 of CR4)** — Enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear.

**CR4.OSFXSR**

**Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4)** — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCH $h$ , SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX registers, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3 instruction, with the exception of PAUSE, PREFETCH $h$ , SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

**NOTE**

CPUID feature flag FXSR indicates availability of the FXSAVE/FXRSTOR instructions. The OSFXSR bit provides operating system software with a means of enabling FXSAVE/FXRSTOR to save/restore the contents of the X87 FPU, XMM and MXCSR registers. Consequently OSFXSR bit indicates that the operating system provides context switch support for SSE/SSE2/SSE3/SSSE3/SSE4.

**CR4.OSXMMEXCPT**

**Operating System Support for Unmasked SIMD Floating-Point Exceptions (bit 10 of CR4)** — When set, indicates that the operating system supports the handling of unmasked SIMD floating-point exceptions through an exception handler that is invoked when a SIMD floating-point exception (#XM) is generated. SIMD floating-point exceptions are only generated by SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instructions.

The operating system or executive must explicitly set this flag. If this flag is not set, the processor will generate an invalid opcode exception (#UD) whenever it detects an unmasked SIMD floating-point exception.

**CR4.UMIP**

**User-Mode Instruction Prevention (bit 11 of CR4)** — When set, the following instructions cannot be executed if CPL > 0: SGDT, SIDT, SLDT, SMSW, and STR. An attempt at such execution causes a general-protection exception (#GP).

**CR4.VMXE**

**VMX-Enable Bit (bit 13 of CR4)** — Enables VMX operation when set. See Chapter 23, "Introduction to Virtual Machine Extensions."

**CR4.SMXE**

**SMX-Enable Bit (bit 14 of CR4)** — Enables SMX operation when set. See Chapter 6, "Safer Mode Extensions Reference" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D*.

**CR4.FSGSBASE**

**FSGSBASE-Enable Bit (bit 16 of CR4)** — Enables the instructions RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE.

**CR4.PCIDE**

**PCID-Enable Bit (bit 17 of CR4)** — Enables process-context identifiers (PCIDs) when set. See Section 4.10.1, "Process-Context Identifiers (PCIDs)". Can be set only in IA-32e mode (if IA32\_EFER.LMA = 1).

**CR4.OSXSAVE**

**XSAVE and Processor Extended States-Enable Bit (bit 18 of CR4)** — When set, this flag: (1) indicates (via CPUID.01H:ECX.OSXSAVE[bit 27]) that the operating system supports the use of the XGETBV, XSAVE and XRSTOR instructions by general software; (2) enables the XSAVE and XRSTOR instructions to save and restore the x87 FPU state (including MMX registers), the SSE state (XMM registers and MXCSR), along with other processor extended states enabled in XCR0; (3) enables the processor to execute XGETBV and XSETBV instructions in order to read and write XCR0. See Section 2.6 and Chapter 13, “System Programming for Instruction Set Extensions and Processor Extended States”.

**CR4.SMEP**

**SMEP-Enable Bit (bit 20 of CR4)** — Enables supervisor-mode execution prevention (SMEP) when set. See Section 4.6, “Access Rights”.

**CR4.SMAP**

**SMAP-Enable Bit (bit 21 of CR4)** — Enables supervisor-mode access prevention (SMAP) when set. See Section 4.6, “Access Rights”.

**CR4.PKE**

**Protection-Key-Enable Bit (bit 22 of CR4)** — Enables 4-level paging to associate each linear address with a protection key. The PKRU register specifies, for each protection key, whether user-mode linear addresses with that protection key can be read or written. This bit also enables access to the PKRU register using the RDPKRU and WRPKRU instructions.

**CR8.TPL**

**Task Priority Level (bit 3:0 of CR8)** — This sets the threshold value corresponding to the highest-priority interrupt to be blocked. A value of 0 means all interrupts are enabled. This field is available in 64-bit mode. A value of 15 means all interrupts will be disabled.

## 2.5.1 CPUID Qualification of Control Register Flags

Not all flags in control register CR4 are implemented on all processors. With the exception of the PCE flag, they can be qualified with the CPUID instruction to determine if they are implemented on the processor before they are used.

The CR8 register is available on processors that support Intel 64 architecture.

## 2.6 EXTENDED CONTROL REGISTERS (INCLUDING XCR0)

If CPUID.01H:ECX.XSAVE[bit 26] is 1, the processor supports one or more **extended control registers (XCRs)**. Currently, the only such register defined is XCR0. This register specifies the set of processor state components for which the operating system provides context management, e.g. x87 FPU state, SSE state, AVX state. The OS programs XCR0 to reflect the features for which it provides context management.

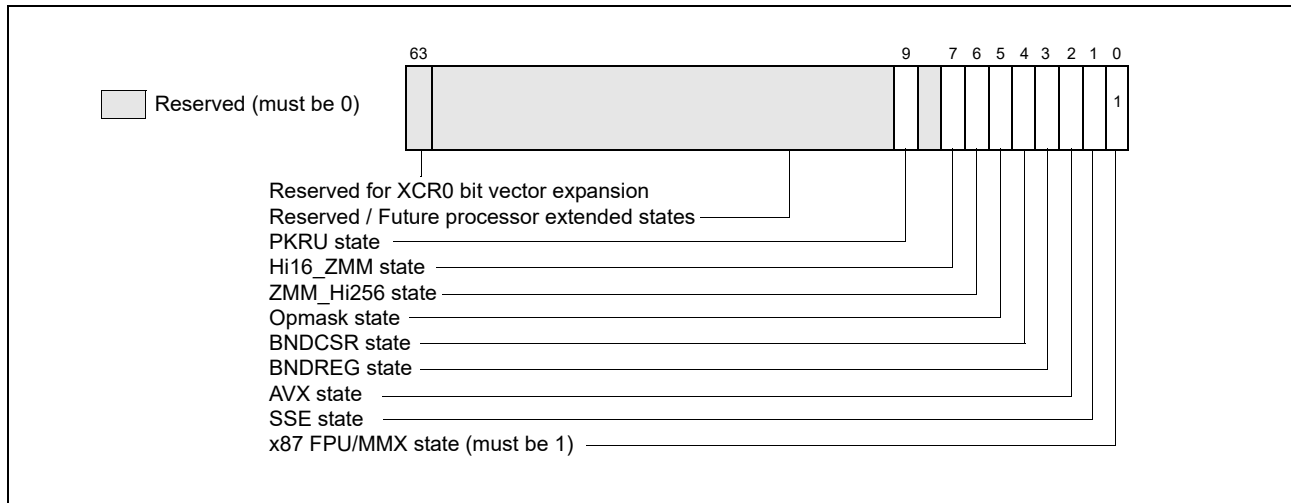


Figure 2-8. XCR0

Software can access XCR0 only if CR4.OSXSAVE[bit 18] = 1. (This bit is also readable as CPUID.01H:ECX.OSXSAVE[bit 27].) Software can use CPUID leaf function 0DH to enumerate the bits in XCR0 that the processor supports (see CPUID instruction in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Each supported state component is represented by a bit in XCR0. System software enables state components by loading an appropriate bit mask value into XCR0 using the XSETBV instruction.

As each bit in XCR0 (except bit 63) corresponds to a processor state component, XCR0 thus provides support for up to 63 sets of processor state components. Bit 63 of XCR0 is reserved for future expansion and will not represent a processor state component.

Currently, XCR0 defines support for the following state components:

- XCR0.X87 (bit 0): This bit 0 must be 1. An attempt to write 0 to this bit causes a #GP exception.
- XCR0.SSE (bit 1): If 1, the XSAVE feature set can be used to manage MXCSR and the XMM registers (XMM0-XMM15 in 64-bit mode; otherwise XMM0-XMM7).
- XCR0.AVX (bit 2): If 1, AVX instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the YMM registers (YMM0-YMM15 in 64-bit mode; otherwise YMM0-YMM7).
- XCR0.BNDREG (bit 3): If 1, MPX instructions can be executed and the XSAVE feature set can be used to manage the bounds registers BND0-BND3.
- XCR0.BNDCSR (bit 4): If 1, MPX instructions can be executed and the XSAVE feature set can be used to manage the BNDCFGU and BNDSTATUS registers.
- XCR0.opmask (bit 5): If 1, AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the opmask registers k0-k7.
- XCR0.ZMM\_Hi256 (bit 6): If 1, AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the lower ZMM registers (ZMM0-ZMM15 in 64-bit mode; otherwise ZMM0-ZMM7).
- XCR0.Hi16\_ZMM (bit 7): If 1, AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper ZMM registers (ZMM16-ZMM31, only in 64-bit mode).
- XCR0.PKRU (bit 9): If 1, the XSAVE feature set can be used to manage the PKRU register (see Section 2.7).

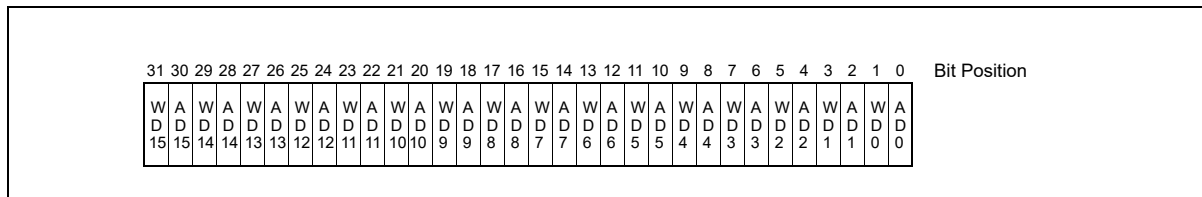
An attempt to use XSETBV to write to XCR0 results in general-protection exceptions (#GP) if it would do any of the following:

- Set a bit reserved in XCR0 for a given processor (as determined by the contents of EAX and EDX after executing CPUID with EAX=0DH, ECX= 0H).
- Clear XCR0.x87.
- Clear XCR0.SSE and set XCR0.AVX.
- Clear XCR0.AVX and set any of XCR0.opmask, XCR0.ZMM\_Hi256, and XCR0.Hi16\_ZMM.
- Set either XCR0.BNDREG and XCR0.BNDCSR while not setting the other.
- Set any of XCR0.opmask, XCR0.ZMM\_Hi256, and XCR0.Hi16\_ZMM while not setting all of them.

After reset, all bits (except bit 0) in XCR0 are cleared to zero; XCR0[0] is set to 1.

## 2.7 PROTECTION KEY RIGHTS REGISTER (PKRU)

If CPUID.(EAX=07H,ECX=0H):ECX.PKU [bit 3] = 1, the processor supports the protection-key feature for 4-level paging. The feature allows selective protection of user-mode pages depending on the 4-bit protection key assigned to each page. The **protection key rights register for user pages (PKRU)** allows software to specify the access rights for each protection key.



**Figure 2-9. Protection Key Rights Register for User Pages (PKRU)**

The layout of the PKRU register is shown in Figure 2-9. It contains 16 pairs of disable controls to prevent data accesses to user-mode linear addresses based on their protection keys. Each protection key  $i$  is associated with two bits in the PKRU register:

- Bit  $2i$ , shown as “AD” (access disable): if set, the processor prevents any data accesses to user-mode linear addresses with protection key  $i$ .
- Bit  $2i+1$ , shown as “WD” (write disable): if set, the processor prevents write accesses to user-mode linear addresses with protection key  $i$ .

See Section 4.6.2, “Protection Keys,” for details of how the processor uses the PKRU register to control accesses to user-mode linear addresses.

## 2.8 SYSTEM INSTRUCTION SUMMARY

System instructions handle system-level functions such as loading system registers, managing the cache, managing interrupts, or setting up the debug registers. Many of these instructions can be executed only by operating-system or executive procedures (that is, procedures running at privilege level 0). Others can be executed at any privilege level and are thus available to application programs.

Table 2-3 lists the system instructions and indicates whether they are available and useful for application programs. These instructions are described in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C & 2D*.

**Table 2-3. Summary of System Instructions**

Instruction	Description	Useful to Application?	Protected from Application?
LLDT	Load LDT Register	No	Yes
SLDT	Store LDT Register	No	If CR4.UMIP = 1
LGDT	Load GDT Register	No	Yes
SGDT	Store GDT Register	No	If CR4.UMIP = 1
LTR	Load Task Register	No	Yes
STR	Store Task Register	No	If CR4.UMIP = 1
LIDT	Load IDT Register	No	Yes
SIDT	Store IDT Register	No	If CR4.UMIP = 1
MOV CR <sub>n</sub>	Load and store control registers	No	Yes
SMSW	Store MSW	Yes	If CR4.UMIP = 1
LMSW	Load MSW	No	Yes
CLTS	Clear TS flag in CRO	No	Yes
ARPL	Adjust RPL	Yes <sup>1,5</sup>	No
LAR	Load Access Rights	Yes	No
LSL	Load Segment Limit	Yes	No
VERR	Verify for Reading	Yes	No
VERW	Verify for Writing	Yes	No
MOV DR <sub>n</sub>	Load and store debug registers	No	Yes
INVD	Invalidate cache, no writeback	No	Yes
WBINVD	Invalidate cache, with writeback	No	Yes
INVLPG	Invalidate TLB entry	No	Yes
HLT	Halt Processor	No	Yes
LOCK (Prefix)	Bus Lock	Yes	No
RSM	Return from system management mode	No	Yes
RDMSR <sup>3</sup>	Read Model-Specific Registers	No	Yes
WRMSR <sup>3</sup>	Write Model-Specific Registers	No	Yes
RDPMS <sup>4</sup>	Read Performance-Monitoring Counter	Yes	Yes <sup>2</sup>
RDTSC <sup>3</sup>	Read Time-Stamp Counter	Yes	Yes <sup>2</sup>
RDTSCP <sup>7</sup>	Read Serialized Time-Stamp Counter	Yes	Yes <sup>2</sup>

Table 2-3. Summary of System Instructions (Contd.)

Instruction	Description	Useful to Application?	Protected from Application?
XGETBV	Return the state of XCRO	Yes	No
XSETBV	Enable one or more processor extended states	No <sup>6</sup>	Yes

**NOTES:**

1. Useful to application programs running at a CPL of 1 or 2.
2. The TSD and PCE flags in control register CR4 control access to these instructions by application programs running at a CPL of 3.
3. These instructions were introduced into the IA-32 Architecture with the Pentium processor.
4. This instruction was introduced into the IA-32 Architecture with the Pentium Pro processor and the Pentium processor with MMX technology.
5. This instruction is not supported in 64-bit mode.
6. Application uses XGETBV to query which set of processor extended states are enabled.
7. RDTSCP is introduced in Intel Core i7 processor.

## 2.8.1 Loading and Storing System Registers

The GDTR, LDTR, IDTR, and TR registers each have a load and store instruction for loading data into and storing data from the register:

- **LGDT (Load GDTR Register)** — Loads the GDT base address and limit from memory into the GDTR register.
- **SGDT (Store GDTR Register)** — Stores the GDT base address and limit from the GDTR register into memory.
- **LIDT (Load IDTR Register)** — Loads the IDT base address and limit from memory into the IDTR register.
- **SIDT (Store IDTR Register)** — Stores the IDT base address and limit from the IDTR register into memory.
- **LLDT (Load LDTR Register)** — Loads the LDT segment selector and segment descriptor from memory into the LDTR. (The segment selector operand can also be located in a general-purpose register.)
- **SLDT (Store LDTR Register)** — Stores the LDT segment selector from the LDTR register into memory or a general-purpose register.
- **LTR (Load Task Register)** — Loads segment selector and segment descriptor for a TSS from memory into the task register. (The segment selector operand can also be located in a general-purpose register.)
- **STR (Store Task Register)** — Stores the segment selector for the current task TSS from the task register into memory or a general-purpose register.

The LMSW (load machine status word) and SMSW (store machine status word) instructions operate on bits 0 through 15 of control register CR0. These instructions are provided for compatibility with the 16-bit Intel 286 processor. Programs written to run on 32-bit IA-32 processors should not use these instructions. Instead, they should access the control register CR0 using the MOV CR instruction.

The CLTS (clear TS flag in CR0) instruction is provided for use in handling a device-not-available exception (#NM) that occurs when the processor attempts to execute a floating-point instruction when the TS flag is set. This instruction allows the TS flag to be cleared after the x87 FPU context has been saved, preventing further #NM exceptions. See Section 2.5, “Control Registers,” for more information on the TS flag.

The control registers (CR0, CR1, CR2, CR3, CR4, and CR8) are loaded using the MOV instruction. The instruction loads a control register from a general-purpose register or stores the content of a control register in a general-purpose register.

## 2.8.2 Verifying of Access Privileges

The processor provides several instructions for examining segment selectors and segment descriptors to determine if access to their associated segments is allowed. These instructions duplicate some of the automatic access rights and type checking done by the processor, thus allowing operating-system or executive software to prevent exceptions from being generated.

The ARPL (adjust RPL) instruction adjusts the RPL (requestor privilege level) of a segment selector to match that of the program or procedure that supplied the segment selector. See Section 5.10.4, “Checking Caller Access Privileges (ARPL Instruction)” for a detailed explanation of the function and use of this instruction. Note that ARPL is not supported in 64-bit mode.

The LAR (load access rights) instruction verifies the accessibility of a specified segment and loads access rights information from the segment’s segment descriptor into a general-purpose register. Software can then examine the access rights to determine if the segment type is compatible with its intended use. See Section 5.10.1, “Checking Access Rights (LAR Instruction)” for a detailed explanation of the function and use of this instruction.

The LSL (load segment limit) instruction verifies the accessibility of a specified segment and loads the segment limit from the segment’s segment descriptor into a general-purpose register. Software can then compare the segment limit with an offset into the segment to determine whether the offset lies within the segment. See Section 5.10.3, “Checking That the Pointer Offset Is Within Limits (LSL Instruction)” for a detailed explanation of the function and use of this instruction.

The VERR (verify for reading) and VERW (verify for writing) instructions verify if a selected segment is readable or writable, respectively, at a given CPL. See Section 5.10.2, “Checking Read/Write Rights (VERR and VERW Instructions)” for a detailed explanation of the function and use of these instructions.

### 2.8.3 Loading and Storing Debug Registers

Internal debugging facilities in the processor are controlled by a set of 8 debug registers (DR0-DR7). The MOV instruction allows setup data to be loaded to and stored from these registers.

On processors that support Intel 64 architecture, debug registers DR0-DR7 are 64 bits. In 32-bit modes and compatibility mode, writes to a debug register fill the upper 32 bits with zeros. Reads return the lower 32 bits. In 64-bit mode, the upper 32 bits of DR6-DR7 are reserved and must be written with zeros. Writing one to any of the upper 32 bits causes an exception, #GP(0).

In 64-bit mode, MOV DRn instructions read or write all 64 bits of a debug register (operand-size prefixes are ignored). All 64 bits of DR0-DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0-DR3 are in the limits of the implementation. Address matching is supported only on valid addresses generated by the processor implementation.

### 2.8.4 Invalidating Caches and TLBs

The processor provides several instructions for use in explicitly invalidating its caches and TLB entries. The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and sends a signal to the external caches indicating that they should also be invalidated.

The WBINVD (invalidate cache with writeback) instruction performs the same function as the INVD instruction, except that it writes back modified lines in its internal caches to memory before it invalidates the caches. After invalidating the caches local to the executing logical processor or processor core, WBINVD signals caches higher in the cache hierarchy (caches shared with the invalidating logical processor or core) to write back any data they have in modified state at the time of instruction execution and to invalidate their contents.

Note, non-shared caches may not be written back nor invalidated. In Figure 2-10 below, if code executing on either LP0 or LP1 were to execute a WBINVD, the shared L1 and L2 for LP0/LP1 will be written back and invalidated as will the shared L3. However, the L1 and L2 caches not shared with LP0 and LP1 will not be written back nor invalidated.

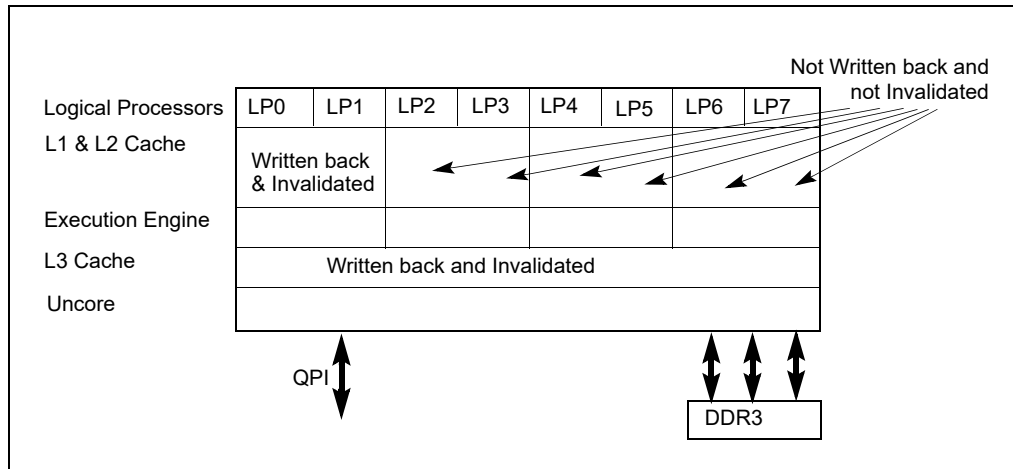


Figure 2-10. WBINVD Invalidation of Shared and Non-Shared Cache Hierarchy

The INVLPG (invalidate TLB entry) instruction invalidates (flushes) the TLB entry for a specified page.

## 2.8.5 Controlling the Processor

The HLT (halt processor) instruction stops the processor until an enabled interrupt (such as NMI or SMI, which are normally enabled), a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered.

Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked (note that the BINIT# pin was introduced with the Pentium Pro processor). If any non-wake events are pending during shutdown, they will be handled after the wake event from shutdown is processed (for example, A20M# interrupts).

The LOCK prefix invokes a locked (atomic) read-modify-write operation when modifying a memory operand. This mechanism is used to allow reliable communications between processors in multiprocessor systems, as described below:

- In the Pentium processor and earlier IA-32 processors, the LOCK prefix causes the processor to assert the LOCK# signal during the instruction. This always causes an explicit bus lock to occur.
- In the Pentium 4, Intel Xeon, and P6 family processors, the locking operation is handled with either a cache lock or bus lock. If a memory access is cacheable and affects only a single cache line, a cache lock is invoked and the system bus and the actual memory location in system memory are not locked during the operation. Here, other Pentium 4, Intel Xeon, or P6 family processors on the bus write-back any modified data and invalidate their caches as necessary to maintain system memory coherency. If the memory access is not cacheable and/or it crosses a cache line boundary, the processor's LOCK# signal is asserted and the processor does not respond to requests for bus control during the locked operation.

The RSM (return from SMM) instruction restores the processor (from a context dump) to the state it was in prior to a system management mode (SMM) interrupt.

## 2.8.6 Reading Performance-Monitoring and Time-Stamp Counters

The RDPMC (read performance-monitoring counter) and RDTSC (read time-stamp counter) instructions allow application programs to read the processor's performance-monitoring and time-stamp counters, respectively. Processors based on Intel NetBurst® microarchitecture have eighteen 40-bit performance-monitoring counters; P6 family processors have two 40-bit counters. Intel® Atom™ processors and most of the processors based on the Intel Core microarchitecture support two types of performance monitoring counters: programmable performance counters similar to those available in the P6 family, and three fixed-function performance monitoring counters.



Details of programmable and fixed-function performance monitoring counters for each processor generation are described in Chapter 18, “Performance Monitoring”.

The programmable performance counters can support counting either the occurrence or duration of events. Events that can be monitored on programmable counters generally are model specific (except for architectural performance events enumerated by CPUID leaf 0AH); they may include the number of instructions decoded, interrupts received, or the number of cache loads. Individual counters can be set up to monitor different events. Use the system instruction WRMSR to set up values in one of the IA32\_PERFEVTSELx MSR, in one of the 45 ESCRs and one of the 18 CCCR MSRs (for Pentium 4 and Intel Xeon processors); or in the PerfEvtSel0 or the PerfEvtSel1 MSR (for the P6 family processors). The RDPMC instruction loads the current count from the selected counter into the EDX:EAX registers.

Fixed-function performance counters record only specific events that are defined in Chapter 19, “Performance Monitoring Events”, and the width/number of fixed-function counters are enumerated by CPUID leaf 0AH.

The time-stamp counter is a model-specific 64-bit counter that is reset to zero each time the processor is reset. If not reset, the counter will increment  $\sim 9.5 \times 10^{16}$  times per year when the processor is operating at a clock rate of 3GHz. At this clock frequency, it would take over 190 years for the counter to wrap around. The RDTSC instruction loads the current count of the time-stamp counter into the EDX:EAX registers.

See Section 18.1, “Performance Monitoring Overview,” and Section 17.17, “Time-Stamp Counter,” for more information about the performance monitoring and time-stamp counters.

The RDTSC instruction was introduced into the IA-32 architecture with the Pentium processor. The RDPMC instruction was introduced into the IA-32 architecture with the Pentium Pro processor and the Pentium processor with MMX technology. Earlier Pentium processors have two performance-monitoring counters, but they can be read only with the RDMSR instruction, and only at privilege level 0.

### 2.8.6.1 Reading Counters in 64-Bit Mode

In 64-bit mode, RDTSC operates the same as in protected mode. The count in the time-stamp counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

RDPMC requires an index to specify the offset of the performance-monitoring counter. In 64-bit mode for Pentium 4 or Intel Xeon processor families, the index is specified in ECX[30:0]. The current count of the performance-monitoring counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

## 2.8.7 Reading and Writing Model-Specific Registers

The RDMSR (read model-specific register) and WRMSR (write model-specific register) instructions allow a processor’s 64-bit model-specific registers (MSRs) to be read and written, respectively. The MSR to be read or written is specified by the value in the ECX register.

RDMSR reads the value from the specified MSR to the EDX:EAX registers; WRMSR writes the value in the EDX:EAX registers to the specified MSR. RDMSR and WRMSR were introduced into the IA-32 architecture with the Pentium processor.

See Section 9.4, “Model-Specific Registers (MSRs),” for more information.

### 2.8.7.1 Reading and Writing Model-Specific Registers in 64-Bit Mode

RDMSR and WRMSR require an index to specify the address of an MSR. In 64-bit mode, the index is 32 bits; it is specified using ECX.

## 2.8.8 Enabling Processor Extended States

The XSETBV instruction is required to enable OS support of individual processor extended states in XCR0 (see Section 2.6).

## 10. Updates to Chapter 3, Volume 3A

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

-----  
Change to this chapter: Update to Table 3-2 "System-Segment and Gate-Descriptor Types".

## CHAPTER 3

# PROTECTED-MODE MEMORY MANAGEMENT

---

This chapter describes the Intel 64 and IA-32 architecture's protected-mode memory management facilities, including the physical memory requirements, segmentation mechanism, and paging mechanism.

See also: Chapter 5, "Protection" (for a description of the processor's protection mechanism) and Chapter 20, "8086 Emulation" (for a description of memory addressing protection in real-address and virtual-8086 modes).

### 3.1 MEMORY MANAGEMENT OVERVIEW

The memory management facilities of the IA-32 architecture are divided into two parts: segmentation and paging. Segmentation provides a mechanism of isolating individual code, data, and stack modules so that multiple programs (or tasks) can run on the same processor without interfering with one another. Paging provides a mechanism for implementing a conventional demand-paged, virtual-memory system where sections of a program's execution environment are mapped into physical memory as needed. Paging can also be used to provide isolation between multiple tasks. When operating in protected mode, some form of segmentation must be used. **There is no mode bit to disable segmentation.** The use of paging, however, is optional.

These two mechanisms (segmentation and paging) can be configured to support simple single-program (or single-task) systems, multitasking systems, or multiple-processor systems that used shared memory.

As shown in Figure 3-1, segmentation provides a mechanism for dividing the processor's addressable memory space (called the **linear address space**) into smaller protected address spaces called **segments**. Segments can be used to hold the code, data, and stack for a program or to hold system data structures (such as a TSS or LDT). If more than one program (or task) is running on a processor, each program can be assigned its own set of segments. The processor then enforces the boundaries between these segments and insures that one program does not interfere with the execution of another program by writing into the other program's segments. The segmentation mechanism also allows typing of segments so that the operations that may be performed on a particular type of segment can be restricted.

All the segments in a system are contained in the processor's linear address space. To locate a byte in a particular segment, a **logical address** (also called a far pointer) must be provided. A logical address consists of a segment selector and an offset. The segment selector is a unique identifier for a segment. Among other things it provides an offset into a descriptor table (such as the global descriptor table, GDT) to a data structure called a segment descriptor. Each segment has a segment descriptor, which specifies the size of the segment, the access rights and privilege level for the segment, the segment type, and the location of the first byte of the segment in the linear address space (called the base address of the segment). The offset part of the logical address is added to the base address for the segment to locate a byte within the segment. The base address plus the offset thus forms a **linear address** in the processor's linear address space.

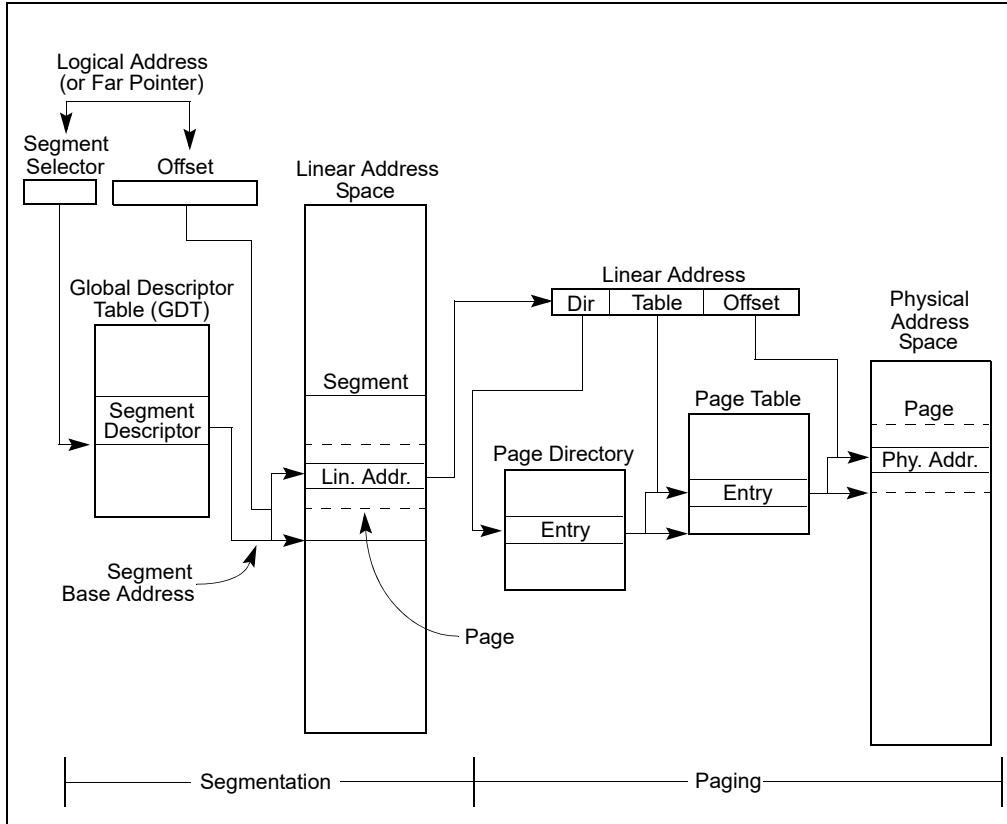


Figure 3-1. Segmentation and Paging

If paging is not used, the linear address space of the processor is mapped directly into the physical address space of processor. The physical address space is defined as the range of addresses that the processor can generate on its address bus.

Because multitasking computing systems commonly define a linear address space much larger than it is economically feasible to contain all at once in physical memory, some method of “virtualizing” the linear address space is needed. This virtualization of the linear address space is handled through the processor’s paging mechanism.

Paging supports a “virtual memory” environment where a large linear address space is simulated with a small amount of physical memory (RAM and ROM) and some disk storage. When using paging, each segment is divided into pages (typically 4 KBytes each in size), which are stored either in physical memory or on the disk. The operating system or executive maintains a page directory and a set of page tables to keep track of the pages. When a program (or task) attempts to access an address location in the linear address space, the processor uses the page directory and page tables to translate the linear address into a physical address and then performs the requested operation (read or write) on the memory location.

If the page being accessed is not currently in physical memory, the processor interrupts execution of the program (by generating a page-fault exception). The operating system or executive then reads the page into physical memory from the disk and continues executing the program.

When paging is implemented properly in the operating-system or executive, the swapping of pages between physical memory and the disk is transparent to the correct execution of a program. Even programs written for 16-bit IA-32 processors can be paged (transparently) when they are run in virtual-8086 mode.

### 3.2 USING SEGMENTS

The segmentation mechanism supported by the IA-32 architecture can be used to implement a wide variety of system designs. These designs range from flat models that make only minimal use of segmentation to protect

programs to multi-segmented models that employ segmentation to create a robust operating environment in which multiple programs and tasks can be executed reliably.

The following sections give several examples of how segmentation can be employed in a system to improve memory management performance and reliability.

### 3.2.1 Basic Flat Model

The simplest memory model for a system is the basic “flat model,” in which the operating system and application programs have access to a continuous, unsegmented address space. To the greatest extent possible, this basic flat model hides the segmentation mechanism of the architecture from both the system designer and the application programmer.

To implement a basic flat memory model with the IA-32 architecture, at least two segment descriptors must be created, one for referencing a code segment and one for referencing a data segment (see Figure 3-2). Both of these segments, however, are mapped to the entire linear address space: that is, both segment descriptors have the same base address value of 0 and the same segment limit of 4 GBytes. By setting the segment limit to 4 GBytes, the segmentation mechanism is kept from generating exceptions for out of limit memory references, even if no physical memory resides at a particular address. ROM (EPROM) is generally located at the top of the physical address space, because the processor begins execution at FFFF\_FFF0H. RAM (DRAM) is placed at the bottom of the address space because the initial base address for the DS data segment after reset initialization is 0.

### 3.2.2 Protected Flat Model

The protected flat model is similar to the basic flat model, except the segment limits are set to include only the range of addresses for which physical memory actually exists (see Figure 3-3). A general-protection exception (#GP) is then generated on any attempt to access nonexistent memory. This model provides a minimum level of hardware protection against some kinds of program bugs.

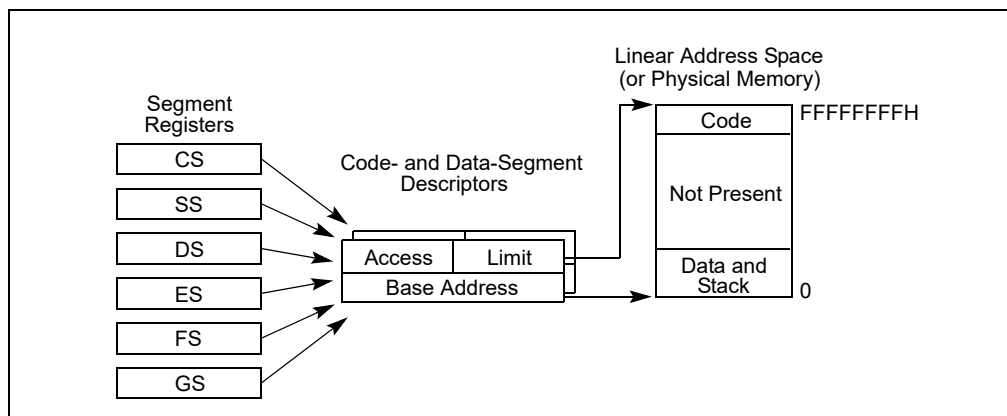


Figure 3-2. Flat Model

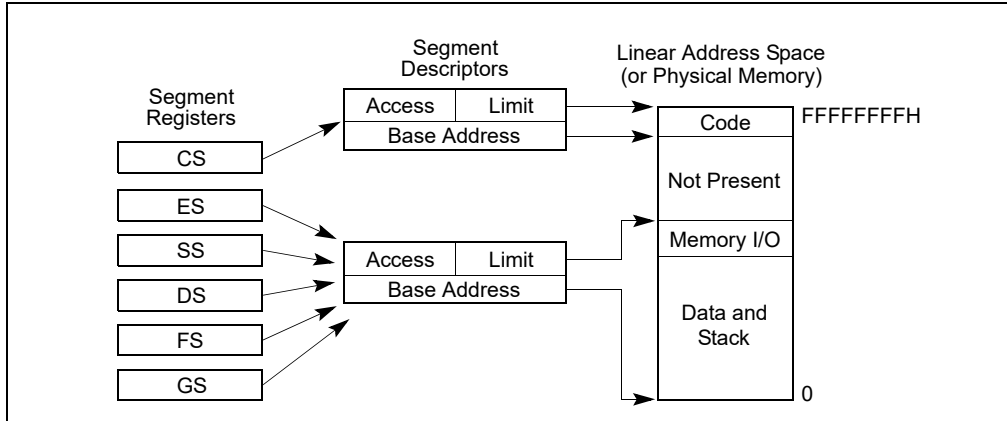


Figure 3-3. Protected Flat Model

More complexity can be added to this protected flat model to provide more protection. For example, for the paging mechanism to provide isolation between user and supervisor code and data, four segments need to be defined: code and data segments at privilege level 3 for the user, and code and data segments at privilege level 0 for the supervisor. Usually these segments all overlay each other and start at address 0 in the linear address space. This flat segmentation model along with a simple paging structure can protect the operating system from applications, and by adding a separate paging structure for each task or process, it can also protect applications from each other. Similar designs are used by several popular multitasking operating systems.

### 3.2.3 Multi-Segment Model

A multi-segment model (such as the one shown in Figure 3-4) uses the full capabilities of the segmentation mechanism to provide hardware enforced protection of code, data structures, and programs and tasks. Here, each program (or task) is given its own table of segment descriptors and its own segments. The segments can be completely private to their assigned programs or shared among programs. Access to all segments and to the execution environments of individual programs running on the system is controlled by hardware.

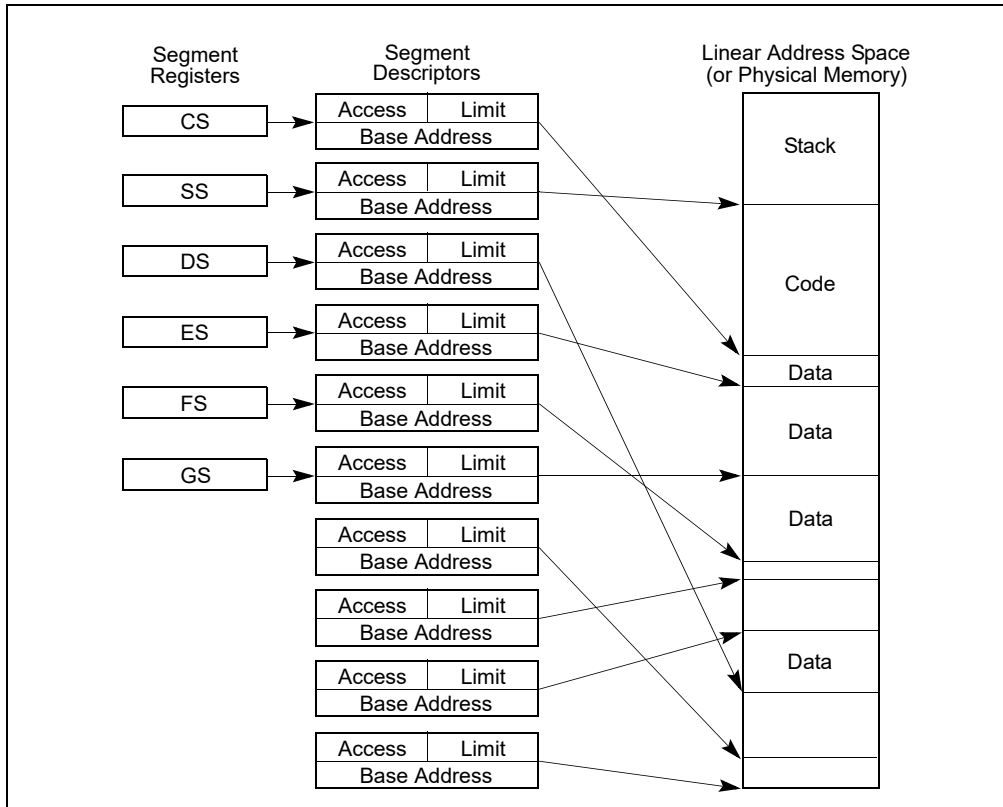


Figure 3-4. Multi-Segment Model

Access checks can be used to protect not only against referencing an address outside the limit of a segment, but also against performing disallowed operations in certain segments. For example, since code segments are designated as read-only segments, hardware can be used to prevent writes into code segments. The access rights information created for segments can also be used to set up protection rings or levels. Protection levels can be used to protect operating-system procedures from unauthorized access by application programs.

### 3.2.4 Segmentation in IA-32e Mode

In IA-32e mode of Intel 64 architecture, the effects of segmentation depend on whether the processor is running in compatibility mode or 64-bit mode. In compatibility mode, segmentation functions just as it does using legacy 16-bit or 32-bit protected mode semantics.

In 64-bit mode, segmentation is generally (but not completely) disabled, creating a flat 64-bit linear-address space. The processor treats the segment base of CS, DS, ES, SS as zero, creating a linear address that is equal to the effective address. The FS and GS segments are exceptions. These segment registers (which hold the segment base) can be used as additional base registers in linear address calculations. They facilitate addressing local data and certain operating system data structures.

Note that the processor does not perform segment limit checks at runtime in 64-bit mode.

### 3.2.5 Paging and Segmentation

Paging can be used with any of the segmentation models described in Figures 3-2, 3-3, and 3-4. The processor's paging mechanism divides the linear address space (into which segments are mapped) into pages (as shown in Figure 3-1). These linear-address-space pages are then mapped to pages in the physical address space. The paging mechanism offers several page-level protection facilities that can be used with or instead of the segment-

protection facilities. For example, it lets read-write protection be enforced on a page-by-page basis. The paging mechanism also provides two-level user-supervisor protection that can also be specified on a page-by-page basis.

### 3.3 PHYSICAL ADDRESS SPACE

In protected mode, the IA-32 architecture provides a normal physical address space of 4 GBytes ( $2^{32}$  bytes). This is the address space that the processor can address on its address bus. This address space is flat (unsegmented), with addresses ranging continuously from 0 to FFFFFFFFH. This physical address space can be mapped to read-write memory, read-only memory, and memory mapped I/O. The memory mapping facilities described in this chapter can be used to divide this physical memory up into segments and/or pages.

Starting with the Pentium Pro processor, the IA-32 architecture also supports an extension of the physical address space to  $2^{36}$  bytes (64 GBytes); with a maximum physical address of FFFFFFFFH. This extension is invoked in either of two ways:

- Using the physical address extension (PAE) flag, located in bit 5 of control register CR4.
- Using the 36-bit page size extension (PSE-36) feature (introduced in the Pentium III processors).

Physical address support has since been extended beyond 36 bits. See Chapter 4, "Paging" for more information about 36-bit physical addressing.

#### 3.3.1 Intel® 64 Processors and Physical Address Space

On processors that support Intel 64 architecture (CPUID.8000001H:EDX[29] = 1), the size of the physical address range is implementation-specific and indicated by CPUID.8000008H:EAX[bits 7-0].

For the format of information returned in EAX, see "CPUID—CPU Identification" in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*. See also: Chapter 4, "Paging."

### 3.4 LOGICAL AND LINEAR ADDRESSES

At the system-architecture level in protected mode, the processor uses two stages of address translation to arrive at a physical address: logical-address translation and linear address space paging.

Even with the minimum use of segments, every byte in the processor's address space is accessed with a logical address. A logical address consists of a 16-bit segment selector and a 32-bit offset (see Figure 3-5). The segment selector identifies the segment the byte is located in and the offset specifies the location of the byte in the segment relative to the base address of the segment.

The processor translates every logical address into a linear address. A linear address is a 32-bit address in the processor's linear address space. Like the physical address space, the linear address space is a flat (unsegmented),  $2^{32}$ -byte address space, with addresses ranging from 0 to FFFFFFFFH. The linear address space contains all the segments and system tables defined for a system.

To translate a logical address into a linear address, the processor does the following:

1. Uses the offset in the segment selector to locate the segment descriptor for the segment in the GDT or LDT and reads it into the processor. (This step is needed only when a new segment selector is loaded into a segment register.)
2. Examines the segment descriptor to check the access rights and range of the segment to insure that the segment is accessible and that the offset is within the limits of the segment.
3. Adds the base address of the segment from the segment descriptor to the offset to form a linear address.



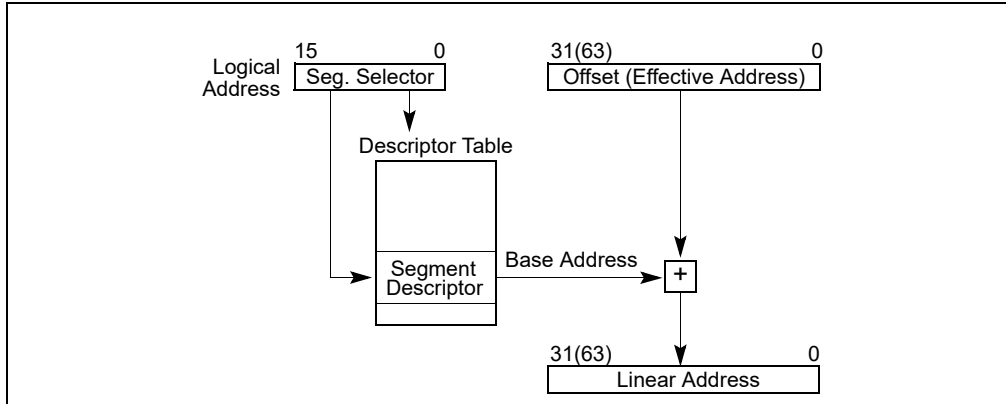


Figure 3-5. Logical Address to Linear Address Translation

If paging is not used, the processor maps the linear address directly to a physical address (that is, the linear address goes out on the processor’s address bus). If the linear address space is paged, a second level of address translation is used to translate the linear address into a physical address.

See also: Chapter 4, “Paging.”

### 3.4.1 Logical Address Translation in IA-32e Mode

In IA-32e mode, an Intel 64 processor uses the steps described above to translate a logical address to a linear address. In 64-bit mode, the offset and base address of the segment are 64-bits instead of 32 bits. The linear address format is also 64 bits wide and is subject to the canonical form requirement.

Each code segment descriptor provides an L bit. This bit allows a code segment to execute 64-bit code or legacy 32-bit code by code segment.

### 3.4.2 Segment Selectors

A segment selector is a 16-bit identifier for a segment (see Figure 3-6). It does not point directly to the segment, but instead points to the segment descriptor that defines the segment. A segment selector contains the following items:

**Index** (Bits 3 through 15) — Selects one of 8192 descriptors in the GDT or LDT. The processor multiplies the index value by 8 (the number of bytes in a segment descriptor) and adds the result to the base address of the GDT or LDT (from the GDTR or LDTR register, respectively).

**TI (table indicator) flag** (Bit 2) — Specifies the descriptor table to use: clearing this flag selects the GDT; setting this flag selects the current LDT.

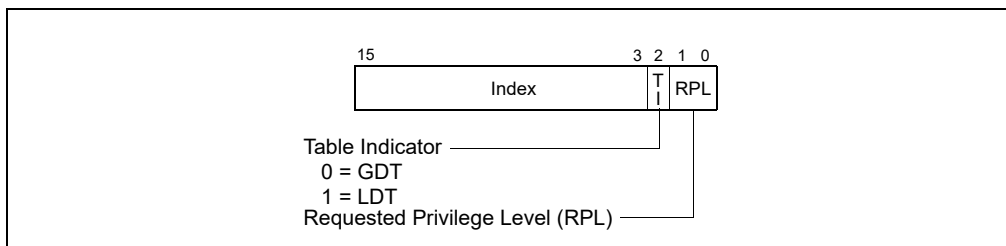


Figure 3-6. Segment Selector

**Requested Privilege Level (RPL)**

(Bits 0 and 1) — Specifies the privilege level of the selector. The privilege level can range from 0 to 3, with 0 being the most privileged level. See Section 5.5, "Privilege Levels", for a description of the relationship of the RPL to the CPL of the executing program (or task) and the descriptor privilege level (DPL) of the descriptor the segment selector points to.

The first entry of the GDT is not used by the processor. A segment selector that points to this entry of the GDT (that is, a segment selector with an index of 0 and the TI flag set to 0) is used as a "null segment selector." The processor does not generate an exception when a segment register (other than the CS or SS registers) is loaded with a null selector. It does, however, generate an exception when a segment register holding a null selector is used to access memory. A null selector can be used to initialize unused segment registers. Loading the CS or SS register with a null segment selector causes a general-protection exception (#GP) to be generated.

Segment selectors are visible to application programs as part of a pointer variable, but the values of selectors are usually assigned or modified by link editors or linking loaders, not application programs.

**3.4.3 Segment Registers**

To reduce address translation time and coding complexity, the processor provides registers for holding up to 6 segment selectors (see Figure 3-7). Each of these segment registers support a specific kind of memory reference (code, stack, or data). For virtually any kind of program execution to take place, at least the code-segment (CS), data-segment (DS), and stack-segment (SS) registers must be loaded with valid segment selectors. The processor also provides three additional data-segment registers (ES, FS, and GS), which can be used to make additional data segments available to the currently executing program (or task).

For a program to access a segment, the segment selector for the segment must have been loaded in one of the segment registers. So, although a system can define thousands of segments, only 6 can be available for immediate use. Other segments can be made available by loading their segment selectors into these registers during program execution.

Visible Part		Hidden Part	
Segment Selector	Base Address, Limit, Access Information		
			CS
			SS
			DS
			ES
			FS
			GS

**Figure 3-7. Segment Registers**

Every segment register has a "visible" part and a "hidden" part. (The hidden part is sometimes referred to as a "descriptor cache" or a "shadow register.") When a segment selector is loaded into the visible part of a segment register, the processor also loads the hidden part of the segment register with the base address, segment limit, and access control information from the segment descriptor pointed to by the segment selector. The information cached in the segment register (visible and hidden) allows the processor to translate addresses without taking extra bus cycles to read the base address and limit from the segment descriptor. In systems in which multiple processors have access to the same descriptor tables, it is the responsibility of software to reload the segment registers when the descriptor tables are modified. If this is not done, an old segment descriptor cached in a segment register might be used after its memory-resident version has been modified.

Two kinds of load instructions are provided for loading the segment registers:

1. Direct load instructions such as the MOV, POP, LDS, LES, LSS, LGS, and LFS instructions. These instructions explicitly reference the segment registers.

2. Implied load instructions such as the far pointer versions of the CALL, JMP, and RET instructions, the SYSENTER and SYSEXIT instructions, and the IRET, INT $n$ , INTO and INT3 instructions. These instructions change the contents of the CS register (and sometimes other segment registers) as an incidental part of their operation.

The MOV instruction can also be used to store the visible part of a segment register in a general-purpose register.

### 3.4.4 Segment Loading Instructions in IA-32e Mode

Because ES, DS, and SS segment registers are not used in 64-bit mode, their fields (base, limit, and attribute) in segment descriptor registers are ignored. Some forms of segment load instructions are also invalid (for example, LDS, POP ES). Address calculations that reference the ES, DS, or SS segments are treated as if the segment base is zero.

The processor checks that all linear-address references are in canonical form instead of performing limit checks. Mode switching does not change the contents of the segment registers or the associated descriptor registers. These registers are also not changed during 64-bit mode execution, unless explicit segment loads are performed.

In order to set up compatibility mode for an application, segment-load instructions (MOV to Sreg, POP Sreg) work normally in 64-bit mode. An entry is read from the system descriptor table (GDT or LDT) and is loaded in the hidden portion of the segment register. The descriptor-register base, limit, and attribute fields are all loaded. However, the contents of the data and stack segment selector and the descriptor registers are ignored.

When FS and GS segment overrides are used in 64-bit mode, their respective base addresses are used in the linear address calculation: (FS or GS).base + index + displacement. FS.base and GS.base are then expanded to the full linear-address size supported by the implementation. The resulting effective address calculation can wrap across positive and negative addresses; the resulting linear address must be canonical.

In 64-bit mode, memory accesses using FS-segment and GS-segment overrides are not checked for a runtime limit nor subjected to attribute-checking. Normal segment loads (MOV to Sreg and POP Sreg) into FS and GS load a standard 32-bit base value in the hidden portion of the segment register. The base address bits above the standard 32 bits are cleared to 0 to allow consistency for implementations that use less than 64 bits.

The hidden descriptor register fields for FS.base and GS.base are physically mapped to MSRs in order to load all address bits supported by a 64-bit implementation. Software with CPL = 0 (privileged software) can load all supported linear-address bits into FS.base or GS.base using WRMSR. Addresses written into the 64-bit FS.base and GS.base registers must be in canonical form. A WRMSR instruction that attempts to write a non-canonical address to those registers causes a #GP fault.

When in compatibility mode, FS and GS overrides operate as defined by 32-bit mode behavior regardless of the value loaded into the upper 32 linear-address bits of the hidden descriptor register base field. Compatibility mode ignores the upper 32 bits when calculating an effective address.

A new 64-bit mode instruction, SWAPGS, can be used to load GS base. SWAPGS exchanges the kernel data structure 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 the kernel data structures. An attempt to write a non-canonical value (using WRMSR) to the IA32\_KERNEL\_GS\_BASE MSR causes a #GP fault.

### 3.4.5 Segment Descriptors

A segment descriptor is a data structure in a GDT or LDT that provides the processor with the size and location of a segment, as well as access control and status information. Segment descriptors are typically created by compilers, linkers, loaders, or the operating system or executive, but not application programs. Figure 3-8 illustrates the general descriptor format for all types of segment descriptors.

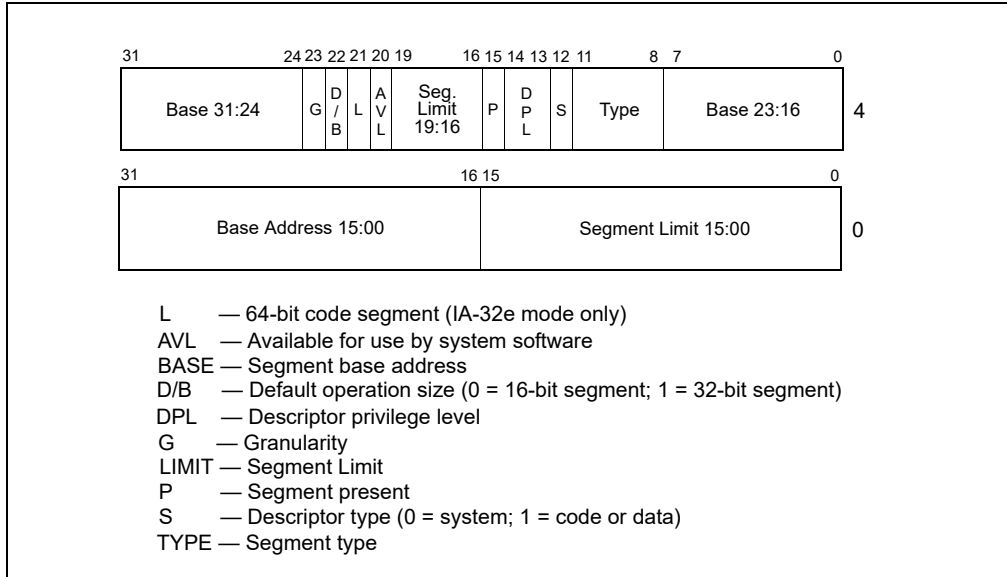


Figure 3-8. Segment Descriptor

The flags and fields in a segment descriptor are as follows:

**Segment limit field**

Specifies the size of the segment. The processor puts together the two segment limit fields to form a 20-bit value. The processor interprets the segment limit in one of two ways, depending on the setting of the G (granularity) flag:

- If the granularity flag is clear, the segment size can range from 1 byte to 1 MByte, in byte increments.
- If the granularity flag is set, the segment size can range from 4 KBytes to 4 GBytes, in 4-KByte increments.

The processor uses the segment limit in two different ways, depending on whether the segment is an expand-up or an expand-down segment. See Section 3.4.5.1, "Code- and Data-Segment Descriptor Types", for more information about segment types. For expand-up segments, the offset in a logical address can range from 0 to the segment limit. Offsets greater than the segment limit generate general-protection exceptions (#GP, for all segments other than SS) or stack-fault exceptions (#SS for the SS segment). For expand-down segments, the segment limit has the reverse function; the offset can range from the segment limit plus 1 to FFFFFFFFH or FFFFH, depending on the setting of the B flag. Offsets less than or equal to the segment limit generate general-protection exceptions or stack-fault exceptions. Decreasing the value in the segment limit field for an expand-down segment allocates new memory at the bottom of the segment's address space, rather than at the top. IA-32 architecture stacks always grow downwards, making this mechanism convenient for expandable stacks.

**Base address fields**

Defines the location of byte 0 of the segment within the 4-GByte linear address space. The processor puts together the three base address fields to form a single 32-bit value. Segment base addresses should be aligned to 16-byte boundaries. Although 16-byte alignment is not required, this alignment allows programs to maximize performance by aligning code and data on 16-byte boundaries.

**Type field**

Indicates the segment or gate type and specifies the kinds of access that can be made to the segment and the direction of growth. The interpretation of this field depends on whether the descriptor type flag specifies an application (code or data) descriptor or a system descriptor. The encoding of the type field is different for code, data, and system descriptors (see Figure 5-1). See Section 3.4.5.1, "Code- and Data-Segment Descriptor Types", for a description of how this field is used to specify code and data-segment types.

**S (descriptor type) flag**

Specifies whether the segment descriptor is for a system segment (S flag is clear) or a code or data segment (S flag is set).

**DPL (descriptor privilege level) field**

Specifies the privilege level of the segment. The privilege level can range from 0 to 3, with 0 being the most privileged level. The DPL is used to control access to the segment. See Section 5.5, "Privilege Levels", for a description of the relationship of the DPL to the CPL of the executing code segment and the RPL of a segment selector.

**P (segment-present) flag**

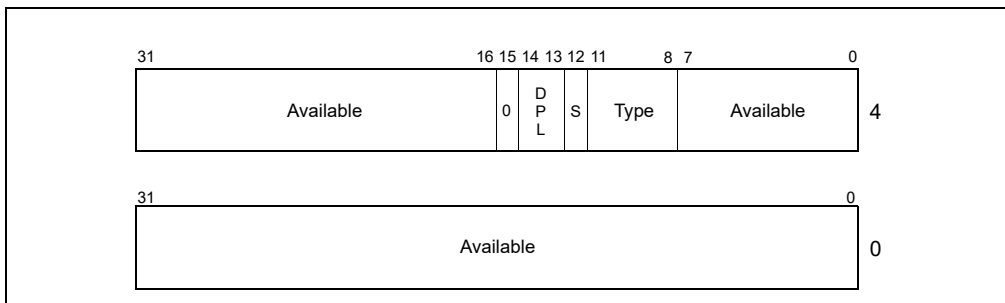
Indicates whether the segment is present in memory (set) or not present (clear). If this flag is clear, the processor generates a segment-not-present exception (#NP) when a segment selector that points to the segment descriptor is loaded into a segment register. Memory management software can use this flag to control which segments are actually loaded into physical memory at a given time. It offers a control in addition to paging for managing virtual memory.

Figure 3-9 shows the format of a segment descriptor when the segment-present flag is clear. When this flag is clear, the operating system or executive is free to use the locations marked "Available" to store its own data, such as information regarding the whereabouts of the missing segment.

**D/B (default operation size/default stack pointer size and/or upper bound) flag**

Performs different functions depending on whether the segment descriptor is an executable code segment, an expand-down data segment, or a stack segment. (This flag should always be set to 1 for 32-bit code and data segments and to 0 for 16-bit code and data segments.)

- **Executable code segment.** The flag is called the D flag and it indicates the default length for effective addresses and operands referenced by instructions in the segment. If the flag is set, 32-bit addresses and 32-bit or 8-bit operands are assumed; if it is clear, 16-bit addresses and 16-bit or 8-bit operands are assumed. The instruction prefix 66H can be used to select an operand size other than the default, and the prefix 67H can be used select an address size other than the default.
- **Stack segment (data segment pointed to by the SS register).** The flag is called the B (big) flag and it specifies the size of the stack pointer used for implicit stack operations (such as pushes, pops, and calls). If the flag is set, a 32-bit stack pointer is used, which is stored in the 32-bit ESP register; if the flag is clear, a 16-bit stack pointer is used, which is stored in the 16-bit SP register. If the stack segment is set up to be an expand-down data segment (described in the next paragraph), the B flag also specifies the upper bound of the stack segment.
- **Expand-down data segment.** The flag is called the B flag and it specifies the upper bound of the segment. If the flag is set, the upper bound is FFFFFFFH (4 GBytes); if the flag is clear, the upper bound is FFFFH (64 KBytes).



**Figure 3-9. Segment Descriptor When Segment-Present Flag Is Clear**

**G (granularity) flag**

Determines the scaling of the segment limit field. When the granularity flag is clear, the segment limit is interpreted in byte units; when flag is set, the segment limit is interpreted in 4-KByte units. (This flag does not affect the granularity of the base address; it is always byte granular.) When the granularity flag is set, the twelve least significant bits of an offset are not tested when checking the

offset against the segment limit. For example, when the granularity flag is set, a limit of 0 results in valid offsets from 0 to 4095.

**L (64-bit code segment) flag**

In IA-32e mode, bit 21 of the second doubleword of the segment descriptor indicates whether a code segment contains native 64-bit code. A value of 1 indicates instructions in this code segment are executed in 64-bit mode. A value of 0 indicates the instructions in this code segment are executed in compatibility mode. If L-bit is set, then D-bit must be cleared. When not in IA-32e mode or for non-code segments, bit 21 is reserved and should always be set to 0.

**Available and reserved bits**

Bit 20 of the second doubleword of the segment descriptor is available for use by system software.

**3.4.5.1 Code- and Data-Segment Descriptor Types**

When the S (descriptor type) flag in a segment descriptor is set, the descriptor is for either a code or a data segment. The highest order bit of the type field (bit 11 of the second double word of the segment descriptor) then determines whether the descriptor is for a data segment (clear) or a code segment (set).

For data segments, the three low-order bits of the type field (bits 8, 9, and 10) are interpreted as accessed (A), write-enabled (W), and expansion-direction (E). See Table 3-1 for a description of the encoding of the bits in the type field for code and data segments. Data segments can be read-only or read/write segments, depending on the setting of the write-enabled bit.

**Table 3-1. Code- and Data-Segment Types**

Decimal	Type Field				Descriptor Type	Description
	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		<b>C</b>	<b>R</b>	<b>A</b>		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

Stack segments are data segments which must be read/write segments. Loading the SS register with a segment selector for a nonwritable data segment generates a general-protection exception (#GP). If the size of a stack segment needs to be changed dynamically, the stack segment can be an expand-down data segment (expansion-direction flag set). Here, dynamically changing the segment limit causes stack space to be added to the bottom of

the stack. If the size of a stack segment is intended to remain static, the stack segment may be either an expand-up or expand-down type.

The accessed bit indicates whether the segment has been accessed since the last time the operating-system or executive cleared the bit. The processor sets this bit whenever it loads a segment selector for the segment into a segment register, assuming that the type of memory that contains the segment descriptor supports processor writes. The bit remains set until explicitly cleared. This bit can be used both for virtual memory management and for debugging.

For code segments, the three low-order bits of the type field are interpreted as accessed (A), read enable (R), and conforming (C). Code segments can be execute-only or execute/read, depending on the setting of the read-enable bit. An execute/read segment might be used when constants or other static data have been placed with instruction code in a ROM. Here, data can be read from the code segment either by using an instruction with a CS override prefix or by loading a segment selector for the code segment in a data-segment register (the DS, ES, FS, or GS registers). In protected mode, code segments are not writable.

Code segments can be either conforming or nonconforming. A transfer of execution into a more-privileged conforming segment allows execution to continue at the current privilege level. A transfer into a nonconforming segment at a different privilege level results in a general-protection exception (#GP), unless a call gate or task gate is used (see Section 5.8.1, "Direct Calls or Jumps to Code Segments", for more information on conforming and nonconforming code segments). System utilities that do not access protected facilities and handlers for some types of exceptions (such as, divide error or overflow) may be loaded in conforming code segments. Utilities that need to be protected from less privileged programs and procedures should be placed in nonconforming code segments.

#### NOTE

Execution cannot be transferred by a call or a jump to a less-privileged (numerically higher privilege level) code segment, regardless of whether the target segment is a conforming or nonconforming code segment. Attempting such an execution transfer will result in a general-protection exception.

All data segments are nonconforming, meaning that they cannot be accessed by less privileged programs or procedures (code executing at numerically higher privilege levels). Unlike code segments, however, data segments can be accessed by more privileged programs or procedures (code executing at numerically lower privilege levels) without using a special access gate.

If the segment descriptors in the GDT or an LDT are placed in ROM, the processor can enter an indefinite loop if software or the processor attempts to update (write to) the ROM-based segment descriptors. To prevent this problem, set the accessed bits for all segment descriptors placed in a ROM. Also, remove operating-system or executive code that attempts to modify segment descriptors located in ROM.

## 3.5 SYSTEM DESCRIPTOR TYPES

When the S (descriptor type) flag in a segment descriptor is clear, the descriptor type is a system descriptor. The processor recognizes the following types of system descriptors:

- Local descriptor-table (LDT) segment descriptor.
- Task-state segment (TSS) descriptor.
- Call-gate descriptor.
- Interrupt-gate descriptor.
- Trap-gate descriptor.
- Task-gate descriptor.

These descriptor types fall into two categories: system-segment descriptors and gate descriptors. System-segment descriptors point to system segments (LDT and TSS segments). Gate descriptors are in themselves "gates," which hold pointers to procedure entry points in code segments (call, interrupt, and trap gates) or which hold segment selectors for TSS's (task gates).

Table 3-2 shows the encoding of the type field for system-segment descriptors and gate descriptors. Note that system descriptors in IA-32e mode are 16 bytes instead of 8 bytes.

**Table 3-2. System-Segment and Gate-Descriptor Types**

Type Field					Description	
Decimal	11	10	9	8	32-Bit Mode	IA-32e Mode
0	0	0	0	0	Reserved	Reserved
1	0	0	0	1	16-bit TSS (Available)	Reserved
2	0	0	1	0	LDT	LDT
3	0	0	1	1	16-bit TSS (Busy)	Reserved
4	0	1	0	0	16-bit Call Gate	Reserved
5	0	1	0	1	Task Gate	Reserved
6	0	1	1	0	16-bit Interrupt Gate	Reserved
7	0	1	1	1	16-bit Trap Gate	Reserved
8	1	0	0	0	Reserved	Reserved
9	1	0	0	1	32-bit TSS (Available)	64-bit TSS (Available)
10	1	0	1	0	Reserved	Reserved
11	1	0	1	1	32-bit TSS (Busy)	64-bit TSS (Busy)
12	1	1	0	0	32-bit Call Gate	64-bit Call Gate
13	1	1	0	1	Reserved	Reserved
14	1	1	1	0	32-bit Interrupt Gate	64-bit Interrupt Gate
15	1	1	1	1	32-bit Trap Gate	64-bit Trap Gate

See also: Section 3.5.1, “Segment Descriptor Tables”, and Section 7.2.2, “TSS Descriptor” (for more information on the system-segment descriptors); see Section 5.8.3, “Call Gates”, Section 6.11, “IDT Descriptors”, and Section 7.2.5, “Task-Gate Descriptor” (for more information on the gate descriptors).

### 3.5.1 Segment Descriptor Tables

A segment descriptor table is an array of segment descriptors (see Figure 3-10). A descriptor table is variable in length and can contain up to 8192 ( $2^{13}$ ) 8-byte descriptors. There are two kinds of descriptor tables:

- The global descriptor table (GDT)
- The local descriptor tables (LDT)



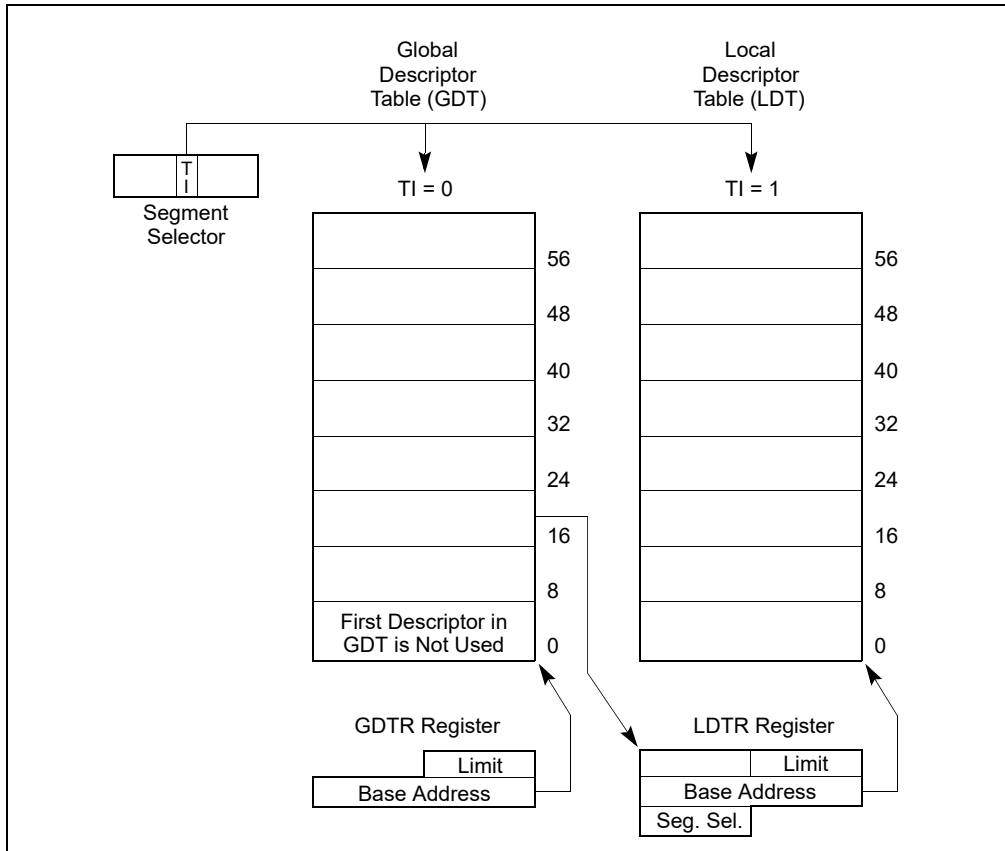


Figure 3-10. Global and Local Descriptor Tables

Each system must have one GDT defined, which may be used for all programs and tasks in the system. Optionally, one or more LDTs can be defined. For example, an LDT can be defined for each separate task being run, or some or all tasks can share the same LDT.

The GDT is not a segment itself; instead, it is a data structure in linear address space. The base linear address and limit of the GDT must be loaded into the GDTR register (see Section 2.4, "Memory-Management Registers"). The base address of the GDT should be aligned on an eight-byte boundary to yield the best processor performance. The limit value for the GDT is expressed in bytes. As with segments, the limit value is added to the base address to get the address of the last valid byte. A limit value of 0 results in exactly one valid byte. Because segment descriptors are always 8 bytes long, the GDT limit should always be one less than an integral multiple of eight (that is,  $8N - 1$ ).

The first descriptor in the GDT is not used by the processor. A segment selector to this "null descriptor" does not generate an exception when loaded into a data-segment register (DS, ES, FS, or GS), but it always generates a general-protection exception (#GP) when an attempt is made to access memory using the descriptor. By initializing the segment registers with this segment selector, accidental reference to unused segment registers can be guaranteed to generate an exception.

The LDT is located in a system segment of the LDT type. The GDT must contain a segment descriptor for the LDT segment. If the system supports multiple LDTs, each must have a separate segment selector and segment descriptor in the GDT. The segment descriptor for an LDT can be located anywhere in the GDT. See Section 3.5, "System Descriptor Types", for information on the LDT segment-descriptor type.

An LDT is accessed with its segment selector. To eliminate address translations when accessing the LDT, the segment selector, base linear address, limit, and access rights of the LDT are stored in the LDTR register (see Section 2.4, "Memory-Management Registers").

When the GDTR register is stored (using the SGDT instruction), a 48-bit "pseudo-descriptor" is stored in memory (see top diagram in Figure 3-11). To avoid alignment check faults in user mode (privilege level 3), the pseudo-descriptor should be located at an odd word address (that is, address MOD 4 is equal to 2). This causes the

processor to store an aligned word, followed by an aligned doubleword. User-mode programs normally do not store pseudo-descriptors, but the possibility of generating an alignment check fault can be avoided by aligning pseudo-descriptors in this way. The same alignment should be used when storing the IDTR register using the SIDT instruction. When storing the LDTR or task register (using the SLDT or STR instruction, respectively), the pseudo-descriptor should be located at a doubleword address (that is, address MOD 4 is equal to 0).

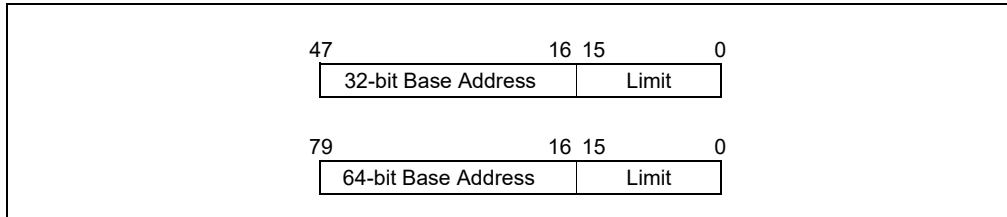


Figure 3-11. Pseudo-Descriptor Formats

### 3.5.2 Segment Descriptor Tables in IA-32e Mode

In IA-32e mode, a segment descriptor table can contain up to 8192 ( $2^{13}$ ) 8-byte descriptors. An entry in the segment descriptor table can be 8 bytes. System descriptors are expanded to 16 bytes (occupying the space of two entries).

GDTR and LDTR registers are expanded to hold 64-bit base address. The corresponding pseudo-descriptor is 80 bits. (see the bottom diagram in Figure 3-11).

The following system descriptors expand to 16 bytes:

- Call gate descriptors (see Section 5.8.3.1, "IA-32e Mode Call Gates")
- IDT gate descriptors (see Section 6.14.1, "64-Bit Mode IDT")
- LDT and TSS descriptors (see Section 7.2.3, "TSS Descriptor in 64-bit mode").

## 11. Updates to Chapter 10, Volume 3A

Change bars show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

-----  
Changes to this chapter: Corrected minor typos and updated text for clarity in various sections throughout chapter.

# CHAPTER 10

## ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER (APIC)

---

The Advanced Programmable Interrupt Controller (APIC), referred to in the following sections as the local APIC, was introduced into the IA-32 processors with the Pentium processor (see Section 22.27, “Advanced Programmable Interrupt Controller (APIC)”) and is included in the P6 family, Pentium 4, Intel Xeon processors, and other more recent Intel 64 and IA-32 processor families (see Section 10.4.2, “Presence of the Local APIC”). The local APIC performs two primary functions for the processor:

- It receives interrupts from the processor’s interrupt pins, from internal sources and from an external I/O APIC (or other external interrupt controller). It sends these to the processor core for handling.
- In multiple processor (MP) systems, it sends and receives interprocessor interrupt (IPI) messages to and from other logical processors on the system bus. IPI messages can be used to distribute interrupts among the processors in the system or to execute system wide functions (such as, booting up processors or distributing work among a group of processors).

The external I/O APIC is part of Intel’s system chip set. Its primary function is to receive external interrupt events from the system and its associated I/O devices and relay them to the local APIC as interrupt messages. In MP systems, the I/O APIC also provides a mechanism for distributing external interrupts to the local APICs of selected processors or groups of processors on the system bus.

This chapter provides a description of the local APIC and its programming interface. It also provides an overview of the interface between the local APIC and the I/O APIC. Contact Intel for detailed information about the I/O APIC.

When a local APIC has sent an interrupt to its processor core for handling, the processor uses the interrupt and exception handling mechanism described in Chapter 6, “Interrupt and Exception Handling.” See Section 6.1, “Interrupt and Exception Overview,” for an introduction to interrupt and exception handling.

### 10.1 LOCAL AND I/O APIC OVERVIEW

Each local APIC consists of a set of APIC registers (see Table 10-1) and associated hardware that control the delivery of interrupts to the processor core and the generation of IPI messages. The APIC registers are memory mapped and can be read and written to using the MOV instruction.

Local APICs can receive interrupts from the following sources:

- **Locally connected I/O devices** — These interrupts originate as an edge or level asserted by an I/O device that is connected directly to the processor’s local interrupt pins (LINT0 and LINT1). The I/O devices may also be connected to an 8259-type interrupt controller that is in turn connected to the processor through one of the local interrupt pins.
- **Externally connected I/O devices** — These interrupts originate as an edge or level asserted by an I/O device that is connected to the interrupt input pins of an I/O APIC. Interrupts are sent as I/O interrupt messages from the I/O APIC to one or more of the processors in the system.
- **Inter-processor interrupts (IPIs)** — An Intel 64 or IA-32 processor can use the IPI mechanism to interrupt another processor or group of processors on the system bus. IPIs are used for software self-interrupts, interrupt forwarding, or preemptive scheduling.
- **APIC timer generated interrupts** — The local APIC timer can be programmed to send a local interrupt to its associated processor when a programmed count is reached (see Section 10.5.4, “APIC Timer”).
- **Performance monitoring counter interrupts** — P6 family, Pentium 4, and Intel Xeon processors provide the ability to send an interrupt to its associated processor when a performance-monitoring counter overflows (see Section 18.6.3.5.8, “Generating an Interrupt on Overflow”).
- **Thermal Sensor interrupts** — Pentium 4 and Intel Xeon processors provide the ability to send an interrupt to themselves when the internal thermal sensor has been tripped (see Section 14.7.2, “Thermal Monitor”).

- **APIC internal error interrupts** — When an error condition is recognized within the local APIC (such as an attempt to access an unimplemented register), the APIC can be programmed to send an interrupt to its associated processor (see Section 10.5.3, “Error Handling”).

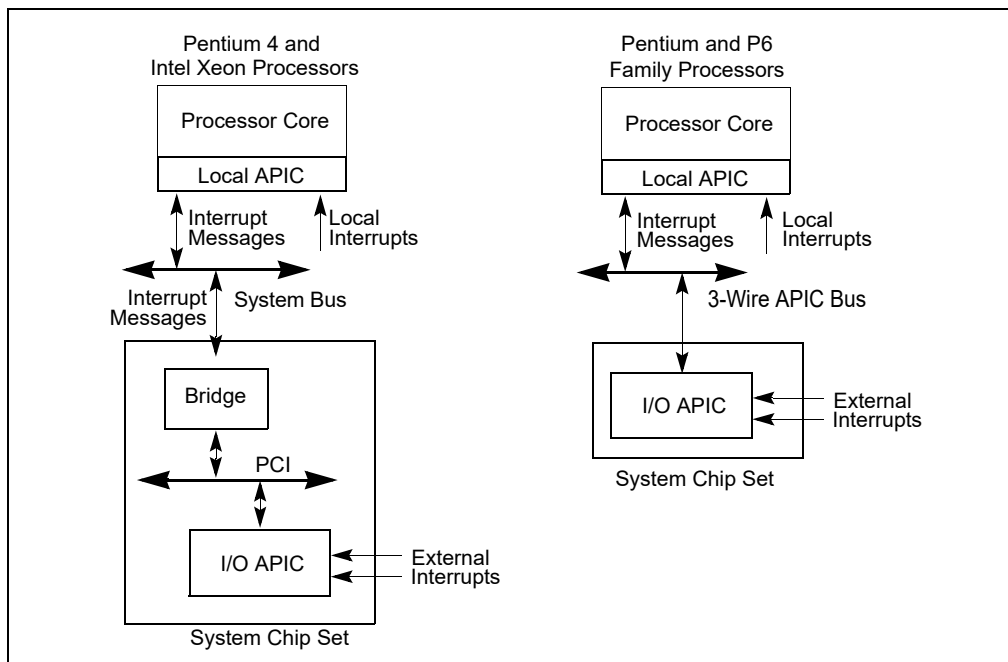
Of these interrupt sources: the processor’s LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, the thermal sensor, and the internal APIC error detector are referred to as **local interrupt sources**. Upon receiving a signal from a local interrupt source, the local APIC delivers the interrupt to the processor core using an interrupt delivery protocol that has been set up through a group of APIC registers called the **local vector table** or **LVT** (see Section 10.5.1, “Local Vector Table”). A separate entry is provided in the local vector table for each local interrupt source, which allows a specific interrupt delivery protocol to be set up for each source. For example, if the LINT1 pin is going to be used as an NMI pin, the LINT1 entry in the local vector table can be set up to deliver an interrupt with vector number 2 (NMI interrupt) to the processor core.

The local APIC handles interrupts from the other two interrupt sources (externally connected I/O devices and IPIs) through its IPI message handling facilities.

A processor can generate IPIs by programming the interrupt command register (ICR) in its local APIC (see Section 10.6.1, “Interrupt Command Register (ICR)”). The act of writing to the ICR causes an IPI message to be generated and issued on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for Pentium and P6 family processors). See Section 10.2, “System Bus Vs. APIC Bus.”

IPIs can be sent to other processors in the system or to the originating processor (self-interrupts). When the target processor receives an IPI message, its local APIC handles the message automatically (using information included in the message such as vector number and trigger mode). See Section 10.6, “Issuing Interprocessor Interrupts,” for a detailed explanation of the local APIC’s IPI message delivery and acceptance mechanism.

The local APIC can also receive interrupts from externally connected devices through the I/O APIC (see Figure 10-1). The I/O APIC is responsible for receiving interrupts generated by system hardware and I/O devices and forwarding them to the local APIC as interrupt messages.



**Figure 10-1. Relationship of Local APIC and I/O APIC In Single-Processor Systems**

Individual pins on the I/O APIC can be programmed to generate a specific interrupt vector when asserted. The I/O APIC also has a “virtual wire mode” that allows it to communicate with a standard 8259A-style external interrupt controller. Note that the local APIC can be disabled (see Section 10.4.3, “Enabling or Disabling the Local APIC”). This allows an associated processor core to receive interrupts directly from an 8259A interrupt controller.

Both the local APIC and the I/O APIC are designed to operate in MP systems (see Figures 10-2 and 10-3). Each local APIC handles interrupts from the I/O APIC, IPIs from processors on the system bus, and self-generated interrupts. Interrupts can also be delivered to the individual processors through the local interrupt pins; however, this mechanism is commonly not used in MP systems.

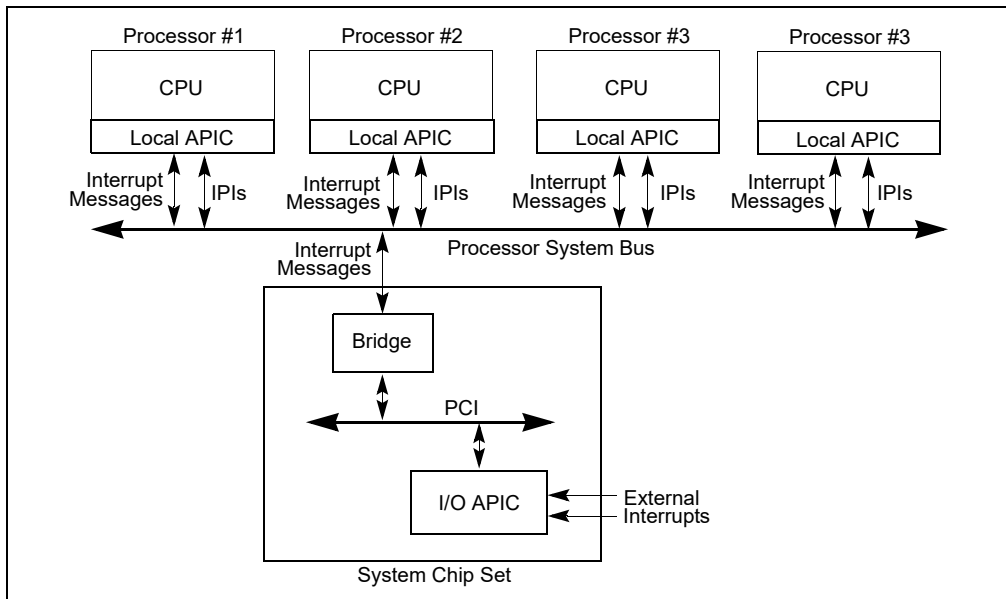


Figure 10-2. Local APICs and I/O APIC When Intel Xeon Processors Are Used in Multiple-Processor Systems

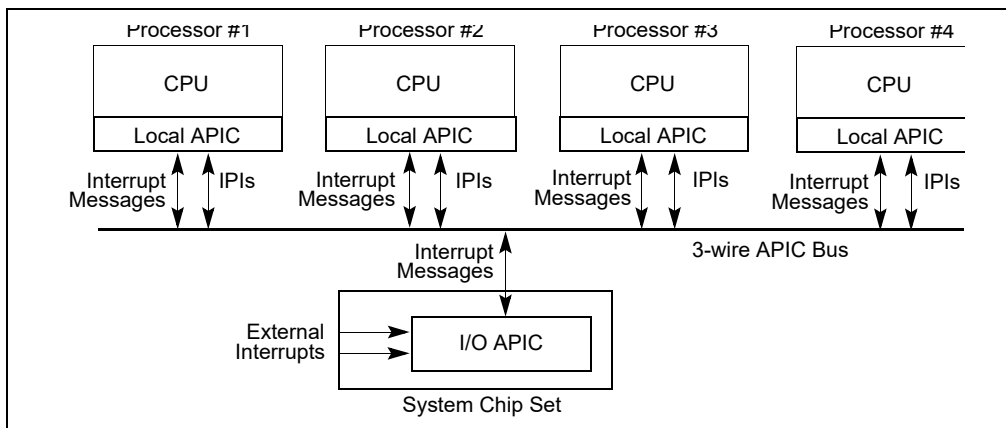


Figure 10-3. Local APICs and I/O APIC When P6 Family Processors Are Used in Multiple-Processor Systems

The IPI mechanism is typically used in MP systems to send fixed interrupts (interrupts for a specific vector number) and special-purpose interrupts to processors on the system bus. For example, a local APIC can use an IPI to forward a fixed interrupt to another processor for servicing. Special-purpose IPIs (including NMI, INIT, SMI and SIPI IPIs) allow one or more processors on the system bus to perform system-wide boot-up and control functions.

The following sections focus on the local APIC and its implementation in the Pentium 4, Intel Xeon, and P6 family processors. In these sections, the terms “local APIC” and “I/O APIC” refer to local and I/O APICs used with the P6 family processors and to local and I/O xAPICs used with the Pentium 4 and Intel Xeon processors (see Section 10.3, “The Intel® 82489DX External APIC, the APIC, the xAPIC, and the X2APIC”).

## 10.2 SYSTEM BUS VS. APIC BUS

For the P6 family and Pentium processors, the I/O APIC and local APICs communicate through the 3-wire inter-APIC bus (see Figure 10-3). Local APICs also use the APIC bus to send and receive IPIs. The APIC bus and its messages are invisible to software and are not classed as architectural.

Beginning with the Pentium 4 and Intel Xeon processors, the I/O APIC and local APICs (using the xAPIC architecture) communicate through the system bus (see Figure 10-2). The I/O APIC sends interrupt requests to the processors on the system bus through bridge hardware that is part of the Intel chip set. The bridge hardware generates the interrupt messages that go to the local APICs. IPIs between local APICs are transmitted directly on the system bus.

## 10.3 THE INTEL® 82489DX EXTERNAL APIC, THE APIC, THE XAPIC, AND THE X2APIC

The local APIC in the P6 family and Pentium processors is an architectural subset of the Intel® 82489DX external APIC. See Section 22.27.1, “Software Visible Differences Between the Local APIC and the 82489DX.”

The APIC architecture used in the Pentium 4 and Intel Xeon processors (called the xAPIC architecture) is an extension of the APIC architecture found in the P6 family processors. The primary difference between the APIC and xAPIC architectures is that with the xAPIC architecture, the local APICs and the I/O APIC communicate through the system bus. With the APIC architecture, they communicate through the APIC bus (see Section 10.2, “System Bus Vs. APIC Bus”). Also, some APIC architectural features have been extended and/or modified in the xAPIC architecture. These extensions and modifications are described in Section 10.4 through Section 10.10.

The basic operating mode of the xAPIC is **xAPIC mode**. The x2APIC architecture is an extension of the xAPIC architecture, primarily to increase processor addressability. The x2APIC architecture provides backward compatibility to the xAPIC architecture and forward extendability for future Intel platform innovations. These extensions and modifications are supported by a new mode of execution (**x2APIC mode**) are detailed in Section 10.12.

## 10.4 LOCAL APIC

The following sections describe the architecture of the local APIC and how to detect it, identify it, and determine its status. Descriptions of how to program the local APIC are given in Section 10.5.1, “Local Vector Table,” and Section 10.6.1, “Interrupt Command Register (ICR).”

### 10.4.1 The Local APIC Block Diagram

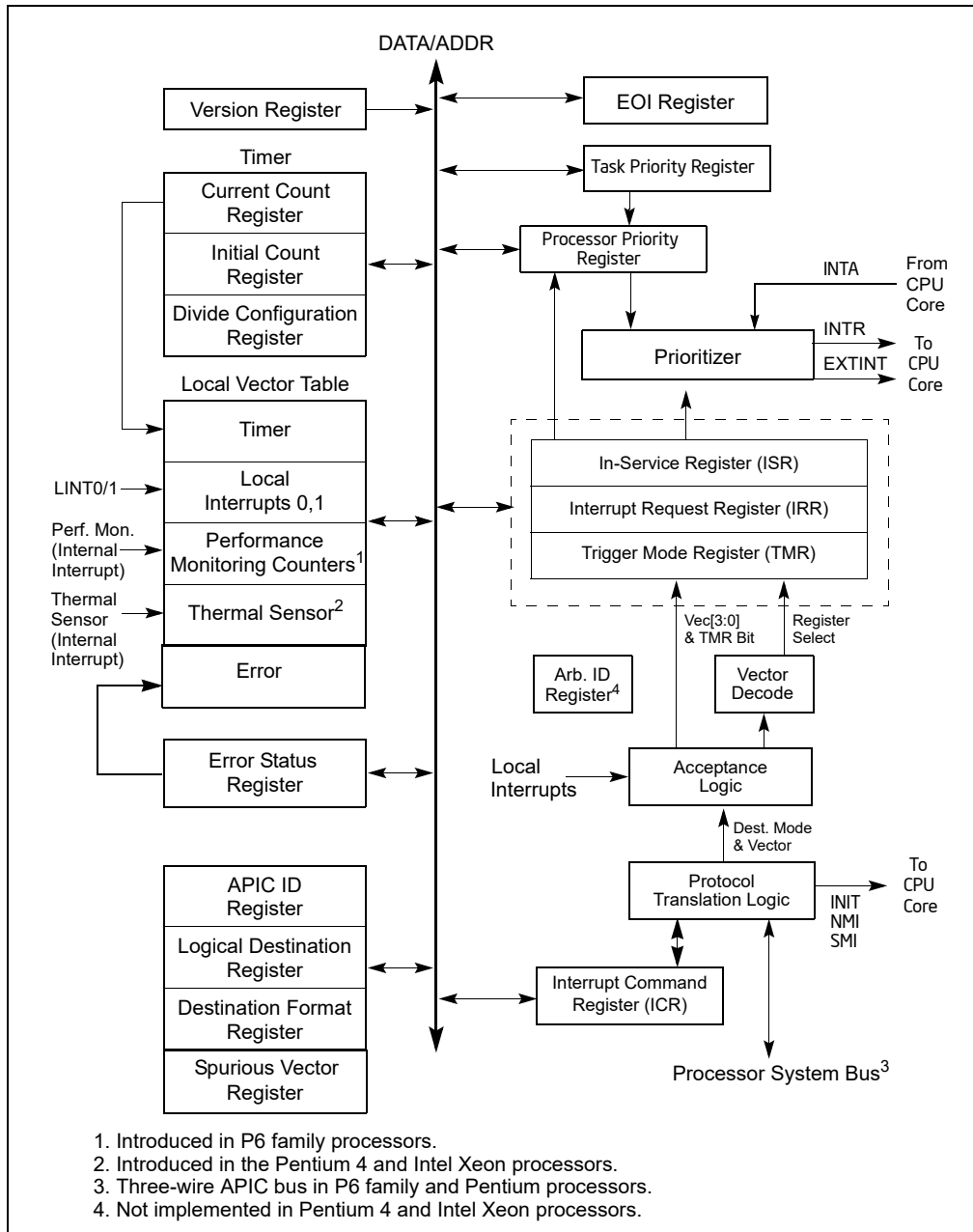
Figure 10-4 gives a functional block diagram for the local APIC. Software interacts with the local APIC by reading and writing its registers. APIC registers are memory-mapped to a 4-KByte region of the processor’s physical address space with an initial starting address of FEE00000H. For correct APIC operation, this address space must be mapped to an area of memory that has been designated as strong uncacheable (UC). See Section 11.3, “Methods of Caching Available.”

In MP system configurations, the APIC registers for Intel 64 or IA-32 processors on the system bus are initially mapped to the same 4-KByte region of the physical address space. Software has the option of changing initial mapping to a different 4-KByte region for all the local APICs or of mapping the APIC registers for each local APIC to its own 4-KByte region. Section 10.4.5, “Relocating the Local APIC Registers,” describes how to relocate the base address for APIC registers.

On processors supporting x2APIC architecture (indicated by CPUID.01H:ECX[21] = 1), the local APIC supports operation both in xAPIC mode and (if enabled by software) in x2APIC mode. x2APIC mode provides extended processor addressability (see Section 10.12).

**NOTE**

For P6 family, Pentium 4, and Intel Xeon processors, the APIC handles all memory accesses to addresses within the 4-KByte APIC register space internally and no external bus cycles are produced. For the Pentium processors with an on-chip APIC, bus cycles are produced for accesses to the APIC register space. Thus, for software intended to run on Pentium processors, system software should explicitly not map the APIC register space to regular system memory. Doing so can result in an invalid opcode exception (#UD) being generated or unpredictable execution.



**Figure 10-4. Local APIC Structure**

Table 10-1 shows how the APIC registers are mapped into the 4-KByte APIC register space. Registers are 32 bits, 64 bits, or 256 bits in width; all are aligned on 128-bit boundaries. All 32-bit registers should be accessed using 128-bit aligned 32-bit loads or stores. Some processors may support loads and stores of less than 32 bits to some of the APIC registers. This is model specific behavior and is not guaranteed to work on all processors. Any



FP/MMX/SSE access to an APIC register, or any access that touches bytes 4 through 15 of an APIC register may cause undefined behavior and must not be executed. This undefined behavior could include hangs, incorrect results or unexpected exceptions, including machine checks, and may vary between implementations. Wider registers (64-bit or 256-bit) must be accessed using multiple 32-bit loads or stores, with all accesses being 128-bit aligned. The local APIC registers listed in Table 10-1 are not MSRs. The only MSR associated with the programming of the local APIC is the IA32\_APIC\_BASE MSR (see Section 10.4.3, “Enabling or Disabling the Local APIC”).

**NOTE**

In processors based on Intel microarchitecture code name Nehalem<sup>1</sup> the Local APIC ID Register is no longer Read/Write; it is Read Only.

**Table 10-1 Local APIC Register Address Map**

Address	Register Name	Software Read/Write
FEE0 0000H	Reserved	
FEE0 0010H	Reserved	
FEE0 0020H	Local APIC ID Register	Read/Write.
FEE0 0030H	Local APIC Version Register	Read Only.
FEE0 0040H	Reserved	
FEE0 0050H	Reserved	
FEE0 0060H	Reserved	
FEE0 0070H	Reserved	
FEE0 0080H	Task Priority Register (TPR)	Read/Write.
FEE0 0090H	Arbitration Priority Register <sup>1</sup> (APR)	Read Only.
FEE0 00A0H	Processor Priority Register (PPR)	Read Only.
FEE0 00B0H	EOI Register	Write Only.
FEE0 00C0H	Remote Read Register <sup>1</sup> (RRD)	Read Only
FEE0 00D0H	Logical Destination Register	Read/Write.
FEE0 00E0H	Destination Format Register	Read/Write (see Section 10.6.2.2).
FEE0 00F0H	Spurious Interrupt Vector Register	Read/Write (see Section 10.9.
FEE0 0100H	In-Service Register (ISR); bits 31:0	Read Only.
FEE0 0110H	In-Service Register (ISR); bits 63:32	Read Only.
FEE0 0120H	In-Service Register (ISR); bits 95:64	Read Only.
FEE0 0130H	In-Service Register (ISR); bits 127:96	Read Only.
FEE0 0140H	In-Service Register (ISR); bits 159:128	Read Only.
FEE0 0150H	In-Service Register (ISR); bits 191:160	Read Only.
FEE0 0160H	In-Service Register (ISR); bits 223:192	Read Only.
FEE0 0170H	In-Service Register (ISR); bits 255:224	Read Only.
FEE0 0180H	Trigger Mode Register (TMR); bits 31:0	Read Only.
FEE0 0190H	Trigger Mode Register (TMR); bits 63:32	Read Only.
FEE0 01A0H	Trigger Mode Register (TMR); bits 95:64	Read Only.

1. See Table 2-1, “CPUID Signature Values of DisplayFamily\_DisplayModel,” on page 1, and Section 2.7, “MSRs In the Intel® Microarchitecture Code Name Nehalem” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* to determine which processors are based on Nehalem microarchitecture.

**Table 10-1 Local APIC Register Address Map (Contd.)**

Address	Register Name	Software Read/Write
FEE0 01B0H	Trigger Mode Register (TMR); bits 127:96	Read Only.
FEE0 01C0H	Trigger Mode Register (TMR); bits 159:128	Read Only.
FEE0 01D0H	Trigger Mode Register (TMR); bits 191:160	Read Only.
FEE0 01E0H	Trigger Mode Register (TMR); bits 223:192	Read Only.
FEE0 01F0H	Trigger Mode Register (TMR); bits 255:224	Read Only.
FEE0 0200H	Interrupt Request Register (IRR); bits 31:0	Read Only.
FEE0 0210H	Interrupt Request Register (IRR); bits 63:32	Read Only.
FEE0 0220H	Interrupt Request Register (IRR); bits 95:64	Read Only.
FEE0 0230H	Interrupt Request Register (IRR); bits 127:96	Read Only.
FEE0 0240H	Interrupt Request Register (IRR); bits 159:128	Read Only.
FEE0 0250H	Interrupt Request Register (IRR); bits 191:160	Read Only.
FEE0 0260H	Interrupt Request Register (IRR); bits 223:192	Read Only.
FEE0 0270H	Interrupt Request Register (IRR); bits 255:224	Read Only.
FEE0 0280H	Error Status Register	Read Only.
FEE0 0290H through FEE0 02E0H	Reserved	
FEE0 02F0H	LVT Corrected Machine Check Interrupt (CMCI) Register	Read/Write.
FEE0 0300H	Interrupt Command Register (ICR); bits 0-31	Read/Write.
FEE0 0310H	Interrupt Command Register (ICR); bits 32-63	Read/Write.
FEE0 0320H	LVT Timer Register	Read/Write.
FEE0 0330H	LVT Thermal Sensor Register <sup>2</sup>	Read/Write.
FEE0 0340H	LVT Performance Monitoring Counters Register <sup>3</sup>	Read/Write.
FEE0 0350H	LVT LINT0 Register	Read/Write.
FEE0 0360H	LVT LINT1 Register	Read/Write.
FEE0 0370H	LVT Error Register	Read/Write.
FEE0 0380H	Initial Count Register (for Timer)	Read/Write.
FEE0 0390H	Current Count Register (for Timer)	Read Only.
FEE0 03A0H through FEE0 03D0H	Reserved	
FEE0 03E0H	Divide Configuration Register (for Timer)	Read/Write.
FEE0 03F0H	Reserved	

**NOTES:**

1. Not supported in the Pentium 4 and Intel Xeon processors. The Illegal Register Access bit (7) of the ESR will not be set when writing to these registers.
2. Introduced in the Pentium 4 and Intel Xeon processors. This APIC register and its associated function are implementation dependent and may not be present in future IA-32 or Intel 64 processors.
3. Introduced in the Pentium Pro processor. This APIC register and its associated function are implementation dependent and may not be present in future IA-32 or Intel 64 processors.

## 10.4.2 Presence of the Local APIC

Beginning with the P6 family processors, the presence or absence of an on-chip local APIC can be detected using the CPUID instruction. When the CPUID instruction is executed with a source operand of 1 in the EAX register, bit 9 of the CPUID feature flags returned in the EDX register indicates the presence (set) or absence (clear) of a local APIC.

## 10.4.3 Enabling or Disabling the Local APIC

The local APIC can be enabled or disabled in either of two ways:

- Using the APIC global enable/disable flag in the IA32\_APIC\_BASE MSR (MSR address 1BH; see Figure 10-5):
  - When IA32\_APIC\_BASE[11] is 0, the processor is functionally equivalent to an IA-32 processor without an on-chip APIC. The CPUID feature flag for the APIC (see Section 10.4.2, "Presence of the Local APIC") is also set to 0.
  - When IA32\_APIC\_BASE[11] is set to 0, processor APICs based on the 3-wire APIC bus cannot be generally re-enabled until a system hardware reset. The 3-wire bus loses track of arbitration that would be necessary for complete re-enabling. Certain APIC functionality can be enabled (for example: performance and thermal monitoring interrupt generation).
  - For processors that use Front Side Bus (FSB) delivery of interrupts, software may disable or enable the APIC by setting and resetting IA32\_APIC\_BASE[11]. A hardware reset is not required to re-start APIC functionality, if software guarantees no interrupt will be sent to the APIC as IA32\_APIC\_BASE[11] is cleared.
  - When IA32\_APIC\_BASE[11] is set to 0, prior initialization to the APIC may be lost and the APIC may return to the state described in Section 10.4.7.1, "Local APIC State After Power-Up or Reset."
- Using the APIC software enable/disable flag in the spurious-interrupt vector register (see Figure 10-23):
  - If IA32\_APIC\_BASE[11] is 1, software can temporarily disable a local APIC at any time by clearing the APIC software enable/disable flag in the spurious-interrupt vector register (see Figure 10-23). The state of the local APIC when in this software-disabled state is described in Section 10.4.7.2, "Local APIC State After It Has Been Software Disabled."
  - When the local APIC is in the software-disabled state, it can be re-enabled at any time by setting the APIC software enable/disable flag to 1.

For the Pentium processor, the APICEN pin (which is shared with the PICD1 pin) is used during power-up or reset to disable the local APIC.

Note that each entry in the LVT has a mask bit that can be used to inhibit interrupts from being delivered to the processor from selected local interrupt sources (the LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, the thermal sensor, and/or the internal APIC error detector).

## 10.4.4 Local APIC Status and Location

The status and location of the local APIC are contained in the IA32\_APIC\_BASE MSR (see Figure 10-5). MSR bit functions are described below:

- BSP flag, bit 8** — Indicates if the processor is the bootstrap processor (BSP). See Section 8.4, "Multiple-Processor (MP) Initialization." Following a power-up or reset, this flag is set to 1 for the processor selected as the BSP and set to 0 for the remaining processors (APs).
- APIC Global Enable flag, bit 11** — Enables or disables the local APIC (see Section 10.4.3, "Enabling or Disabling the Local APIC"). This flag is available in the Pentium 4, Intel Xeon, and P6 family processors. It is not guaranteed to be available or available at the same location in future Intel 64 or IA-32 processors.
- APIC Base field, bits 12 through 35** — Specifies the base address of the APIC registers. This 24-bit value is extended by 12 bits at the low end to form the base address. This automatically aligns the address on a 4-KByte boundary. Following a power-up or reset, the field is set to FEE0 0000H.
- Bits 0 through 7, bits 9 and 10, and bits MAXPHYADDR<sup>1</sup> through 63 in the IA32\_APIC\_BASE MSR are reserved.

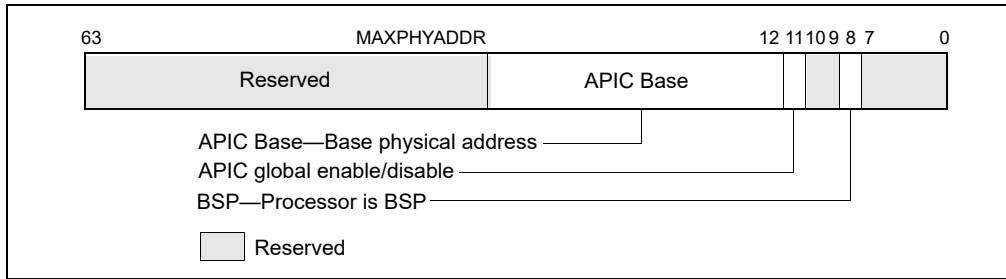


Figure 10-5. IA32\_APIC\_BASE MSR (APIC\_BASE\_MSR in P6 Family)

### 10.4.5 Relocating the Local APIC Registers

The Pentium 4, Intel Xeon, and P6 family processors permit the starting address of the APIC registers to be relocated from FEE00000H to another physical address by modifying the value in the base address field of the IA32\_APIC\_BASE MSR. This extension of the APIC architecture is provided to help resolve conflicts with memory maps of existing systems and to allow individual processors in an MP system to map their APIC registers to different locations in physical memory.

### 10.4.6 Local APIC ID

At power up, system hardware assigns a unique APIC ID to each local APIC on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for P6 family and Pentium processors). The hardware assigned APIC ID is based on system topology and includes encoding for socket position and cluster information (see Figure 8-2 and Section 8.9.1, "Hierarchical Mapping of Shared Resources").

In MP systems, the local APIC ID is also used as a processor ID by the BIOS and the operating system. Some processors permit software to modify the APIC ID. However, the ability of software to modify the APIC ID is processor model specific. Because of this, operating system software should avoid writing to the local APIC ID register. The value returned by bits 31-24 of the EBX register (when the CPUID instruction is executed with a source operand value of 1 in the EAX register) is always the Initial APIC ID (determined by the platform initialization). This is true even if software has changed the value in the Local APIC ID register.

The processor receives the hardware assigned APIC ID (or Initial APIC ID) by sampling pins A11# and A12# and pins BR0# through BR3# (for the Pentium 4, Intel Xeon, and P6 family processors) and pins BE0# through BE3# (for the Pentium processor). The APIC ID latched from these pins is stored in the APIC ID field of the local APIC ID register (see Figure 10-6), and is used as the Initial APIC ID for the processor.

1. The MAXPHYADDR is 36 bits for processors that do not support CPUID leaf 80000008H, or indicated by CPUID.80000008H:EAX[bits 7:0] for processors that support CPUID leaf 80000008H.

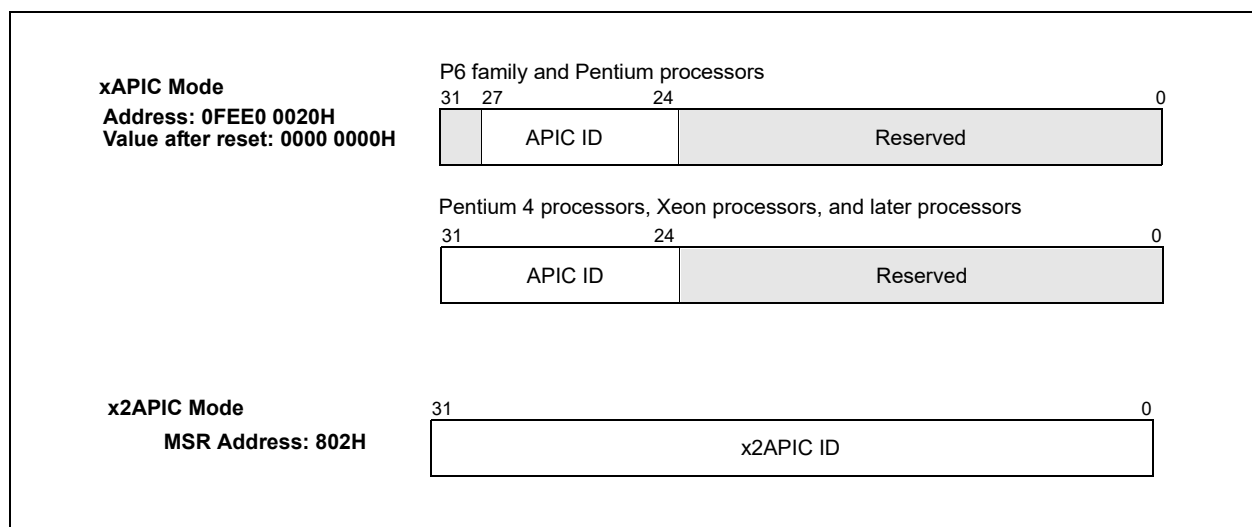


Figure 10-6. Local APIC ID Register

For the P6 family and Pentium processors, the local APIC ID field in the local APIC ID register is 4 bits. Encodings 0H through EH can be used to uniquely identify 15 different processors connected to the APIC bus. For the Pentium 4 and Intel Xeon processors, the xAPIC specification extends the local APIC ID field to 8 bits. These can be used to identify up to 255 processors in the system.

### 10.4.7 Local APIC State

The following sections describe the state of the local APIC and its registers following a power-up or reset, after the local APIC has been software disabled, following an INIT reset, and following an INIT-deassert message.

x2APIC will introduce 32-bit ID; see Section 10.12.

#### 10.4.7.1 Local APIC State After Power-Up or Reset

Following a power-up or reset of the processor, the state of local APIC and its registers are as follows:

- The following registers are reset to all 0s.
  - IRR, ISR, TMR, ICR, LDR, and TPR.
  - Timer initial count and timer current count registers.
  - Divide configuration register.
- The DFR register is reset to all 1s.
- The LVT register is reset to 0s except for the mask bits; these are set to 1s.
- The local APIC version register is not affected.
- The local APIC ID register is set to a unique APIC ID. (Pentium and P6 family processors only). The Arb ID register is set to the value in the APIC ID register.
- The spurious-interrupt vector register is initialized to 000000FFH. By setting bit 8 to 0, software disables the local APIC.
- If the processor is the only processor in the system or it is the BSP in an MP system (see Section 8.4.1, “BSP and AP Processors”); the local APIC will respond normally to INIT and NMI messages, to INIT# signals and to STPCLK# signals. If the processor is in an MP system and has been designated as an AP; the local APIC will respond the same as for the BSP. In addition, it will respond to SIPI messages. For P6 family processors only, an AP will not respond to a STPCLK# signal.

### 10.4.7.2 Local APIC State After It Has Been Software Disabled

When the APIC software enable/disable flag in the spurious interrupt vector register has been explicitly cleared (as opposed to being cleared during a power up or reset), the local APIC is temporarily disabled (see Section 10.4.3, “Enabling or Disabling the Local APIC”). The operation and response of a local APIC while in this software-disabled state is as follows:

- The local APIC will respond normally to INIT, NMI, SMI, and SIPI messages.
- Pending interrupts in the IRR and ISR registers are held and require masking or handling by the CPU.
- The local APIC can still issue IPIs. It is software’s responsibility to avoid issuing IPIs through the IPI mechanism and the ICR register if sending interrupts through this mechanism is not desired.
- The reception of any interrupt or transmission of any IPIs that are in progress when the local APIC is disabled are completed before the local APIC enters the software-disabled state.
- The mask bits for all the LVT entries are set. Attempts to reset these bits will be ignored.
- (For Pentium and P6 family processors) The local APIC continues to listen to all bus messages in order to keep its arbitration ID synchronized with the rest of the system.

### 10.4.7.3 Local APIC State After an INIT Reset (“Wait-for-SIPI” State)

An INIT reset of the processor can be initiated in either of two ways:

- By asserting the processor’s INIT# pin.
- By sending the processor an INIT IPI (an IPI with the delivery mode set to INIT).

Upon receiving an INIT through either of these mechanisms, the processor responds by beginning the initialization process of the processor core and the local APIC. The state of the local APIC following an INIT reset is the same as it is after a power-up or hardware reset, except that the APIC ID and arbitration ID registers are not affected. This state is also referred to at the “wait-for-SIPI” state (see also: Section 8.4.2, “MP Initialization Protocol Requirements and Restrictions”).

### 10.4.7.4 Local APIC State After It Receives an INIT-Deassert IPI

Only the Pentium and P6 family processors support the INIT-deassert IPI. An INIT-deassert IPI has no affect on the state of the APIC, other than to reload the arbitration ID register with the value in the APIC ID register.

## 10.4.8 Local APIC Version Register

The local APIC contains a hardwired version register. Software can use this register to identify the APIC version (see Figure 10-7). In addition, the register specifies the number of entries in the local vector table (LVT) for a specific implementation.

The fields in the local APIC version register are as follows:

<b>Version</b>	The version numbers of the local APIC:
	0XH                    82489DX discrete APIC.
	10H - 15H            Integrated APIC.
	Other values reserved.
<b>Max LVT Entry</b>	Shows the number of LVT entries minus 1. For the Pentium 4 and Intel Xeon processors (which have 6 LVT entries), the value returned in the Max LVT field is 5; for the P6 family processors (which have 5 LVT entries), the value returned is 4; for the Pentium processor (which has 4 LVT entries), the value returned is 3. For processors based on the Intel microarchitecture code name Nehalem (which has 7 LVT entries) and onward, the value returned is 6.
<b>Suppress EOI-broadcasts</b>	Indicates whether software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register; see Section 10.8.5 and Section 10.9.

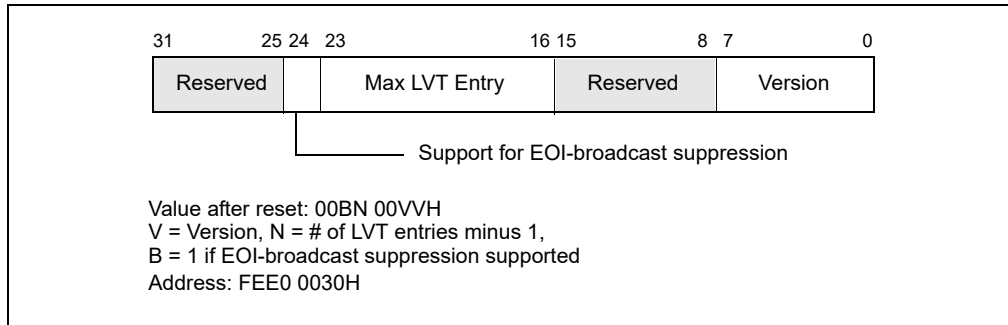


Figure 10-7. Local APIC Version Register

## 10.5 HANDLING LOCAL INTERRUPTS

The following sections describe facilities that are provided in the local APIC for handling local interrupts. These include: the processor’s LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, the thermal sensor, and the internal APIC error detector. Local interrupt handling facilities include: the LVT, the error status register (ESR), the divide configuration register (DCR), and the initial count and current count registers.

### 10.5.1 Local Vector Table

The local vector table (LVT) allows software to specify the manner in which the local interrupts are delivered to the processor core. It consists of the following 32-bit APIC registers (see Figure 10-8), one for each local interrupt:

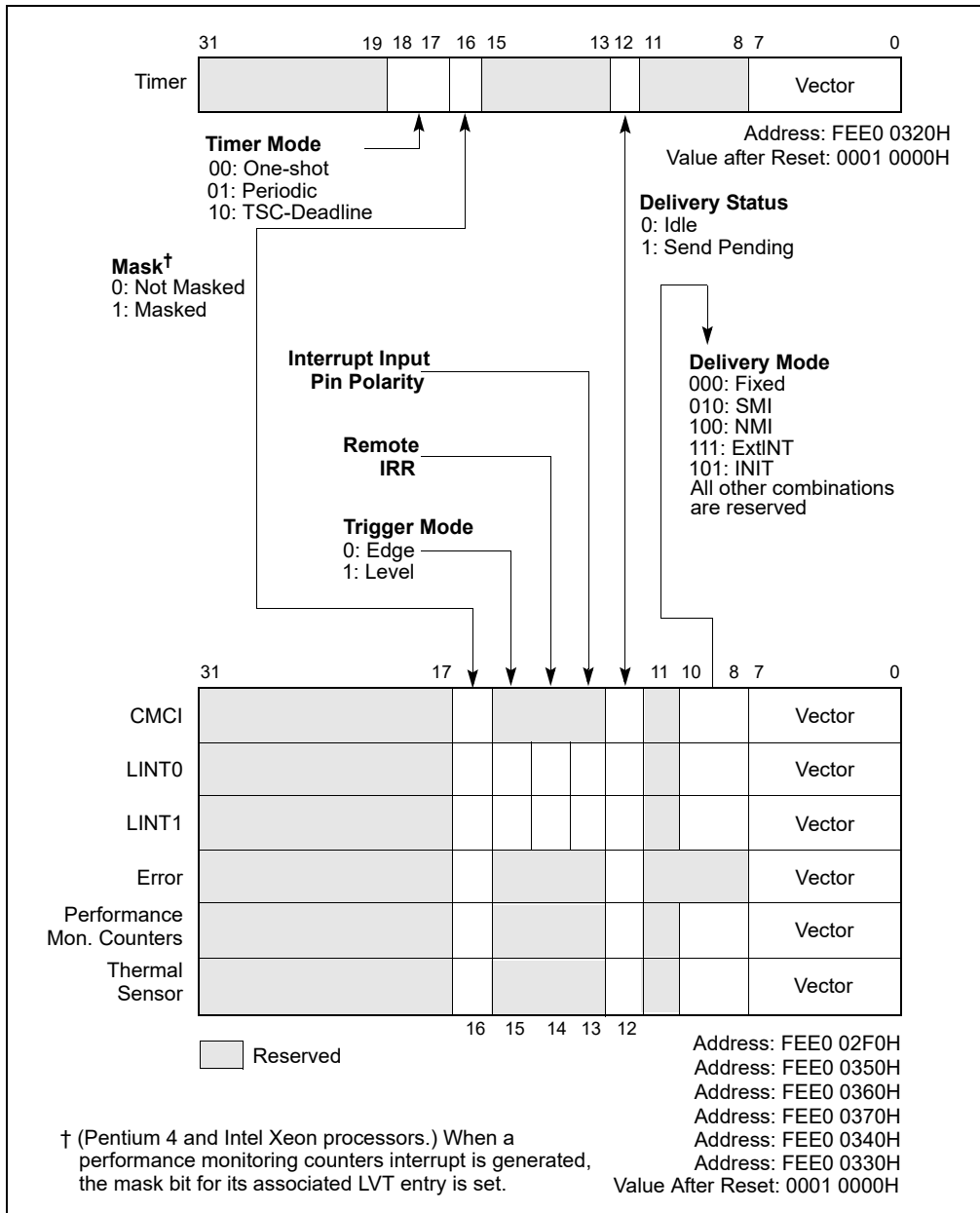
- **LVT CMCI Register (FEE0 02FOH)** — Specifies interrupt delivery when an overflow condition of corrected machine check error count reaching a threshold value occurred in a machine check bank supporting CMCI (see Section 15.5.1, “CMCI Local APIC Interface”).
- **LVT Timer Register (FEE0 0320H)** — Specifies interrupt delivery when the APIC timer signals an interrupt (see Section 10.5.4, “APIC Timer”).
- **LVT Thermal Monitor Register (FEE0 0330H)** — Specifies interrupt delivery when the thermal sensor generates an interrupt (see Section 14.7.2, “Thermal Monitor”). This LVT entry is implementation specific, not architectural. If implemented, it will always be at base address FEE0 0330H.
- **LVT Performance Counter Register (FEE0 0340H)** — Specifies interrupt delivery when a performance counter generates an interrupt on overflow (see Section 18.6.3.5.8, “Generating an Interrupt on Overflow”). This LVT entry is implementation specific, not architectural. If implemented, it is not guaranteed to be at base address FEE0 0340H.
- **LVT LINT0 Register (FEE0 0350H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT0 pin.
- **LVT LINT1 Register (FEE0 0360H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT1 pin.
- **LVT Error Register (FEE0 0370H)** — Specifies interrupt delivery when the APIC detects an internal error (see Section 10.5.3, “Error Handling”).

The LVT performance counter register and its associated interrupt were introduced in the P6 processors and are also present in the Pentium 4 and Intel Xeon processors. The LVT thermal monitor register and its associated interrupt were introduced in the Pentium 4 and Intel Xeon processors. The LVT CMCI register and its associated interrupt were introduced in the Intel Xeon 5500 processors.

As shown in Figures 10-8, some of these fields and flags are not available (and reserved) for some entries.

The setup information that can be specified in the registers of the LVT table is as follows:

**Vector**                      Interrupt vector number.



**Figure 10-8. Local Vector Table (LVT)**

- Delivery Mode** Specifies the type of interrupt to be sent to the processor. Some delivery modes will only operate as intended when used in conjunction with a specific trigger mode. The allowable delivery modes are as follows:
- 000 (Fixed)** Delivers the interrupt specified in the vector field.
  - 010 (SMI)** Delivers an SMI interrupt to the processor core through the processor’s local SMI signal path. When using this delivery mode, the vector field should be set to 00H for future compatibility.
  - 100 (NMI)** Delivers an NMI interrupt to the processor. The vector information is ignored.
  - 101 (INIT)** Delivers an INIT request to the processor core, which causes the processor to perform an INIT. When using this delivery mode, the vector field should



be set to 00H for future compatibility. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.

**110** Reserved; not supported for any LVT register.

**111 (ExtINT)** Causes the processor to respond to the interrupt as if the interrupt originated in an externally connected (8259A-compatible) interrupt controller. A special INTA bus cycle corresponding to ExtINT, is routed to the external controller. The external controller is expected to supply the vector information. The APIC architecture supports only one ExtINT source in a system, usually contained in the compatibility bridge. Only one processor in the system should have an LVT entry configured to use the ExtINT delivery mode. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.

**Delivery Status (Read Only)**

Indicates the interrupt delivery status, as follows:

**0 (Idle)** There is currently no activity for this interrupt source, or the previous interrupt from this source was delivered to the processor core and accepted.

**1 (Send Pending)** Indicates that an interrupt from this source has been delivered to the processor core but has not yet been accepted (see Section 10.5.5, "Local Interrupt Acceptance").

**Interrupt Input Pin Polarity**

Specifies the polarity of the corresponding interrupt pin: (0) active high or (1) active low.

**Remote IRR Flag (Read Only)**

For fixed mode, level-triggered interrupts; this flag is set when the local APIC accepts the interrupt for servicing and is reset when an EOI command is received from the processor. The meaning of this flag is undefined for edge-triggered interrupts and other delivery modes.

**Trigger Mode**

Selects the trigger mode for the local LINT0 and LINT1 pins: (0) edge sensitive and (1) level sensitive. This flag is only used when the delivery mode is Fixed. When the delivery mode is NMI, SMI, or INIT, the trigger mode is always edge sensitive. When the delivery mode is ExtINT, the trigger mode is always level sensitive. The timer and error interrupts are always treated as edge sensitive.

If the local APIC is not used in conjunction with an I/O APIC and fixed delivery mode is selected; the Pentium 4, Intel Xeon, and P6 family processors will always use level-sensitive triggering, regardless if edge-sensitive triggering is selected.

Software should always set the trigger mode in the LVT LINT1 register to 0 (edge sensitive). Level-sensitive interrupts are not supported for LINT1.

**Mask**

Interrupt mask: (0) enables reception of the interrupt and (1) inhibits reception of the interrupt. When the local APIC handles a performance-monitoring counters interrupt, it automatically sets the mask flag in the LVT performance counter register. This flag is set to 1 on reset. It can be cleared only by software.

**Timer Mode**

Bits 18:17 selects the timer mode (see Section 10.5.4):

(00b) one-shot mode using a count-down value,

(01b) periodic mode reloading a count-down value,

(10b) TSC-Deadline mode using absolute target value in IA32\_TSC\_DEADLINE MSR (see Section 10.5.4.1),

(11b) is reserved.

## 10.5.2 Valid Interrupt Vectors

The Intel 64 and IA-32 architectures define 256 vector numbers, ranging from 0 through 255 (see Section 6.2, "Exception and Interrupt Vectors"). Local and I/O APICs support 240 of these vectors (in the range of 16 to 255) as valid interrupts.

When an interrupt vector in the range of 0 to 15 is sent or received through the local APIC, the APIC indicates an illegal vector in its Error Status Register (see Section 10.5.3, "Error Handling"). The Intel 64 and IA-32 architectures reserve vectors 16 through 31 for predefined interrupts, exceptions, and Intel-reserved encodings (see Table 6-1). However, the local APIC does not treat vectors in this range as illegal.

When an illegal vector value (0 to 15) is written to an LVT entry and the delivery mode is Fixed (bits 8-11 equal 0), the APIC may signal an illegal vector error, without regard to whether the mask bit is set or whether an interrupt is actually seen on the input.

### 10.5.3 Error Handling

The local APIC records errors detected during interrupt handling in the error status register (ESR). The format of the ESR is given in Figure 10-9; it contains the following flags:

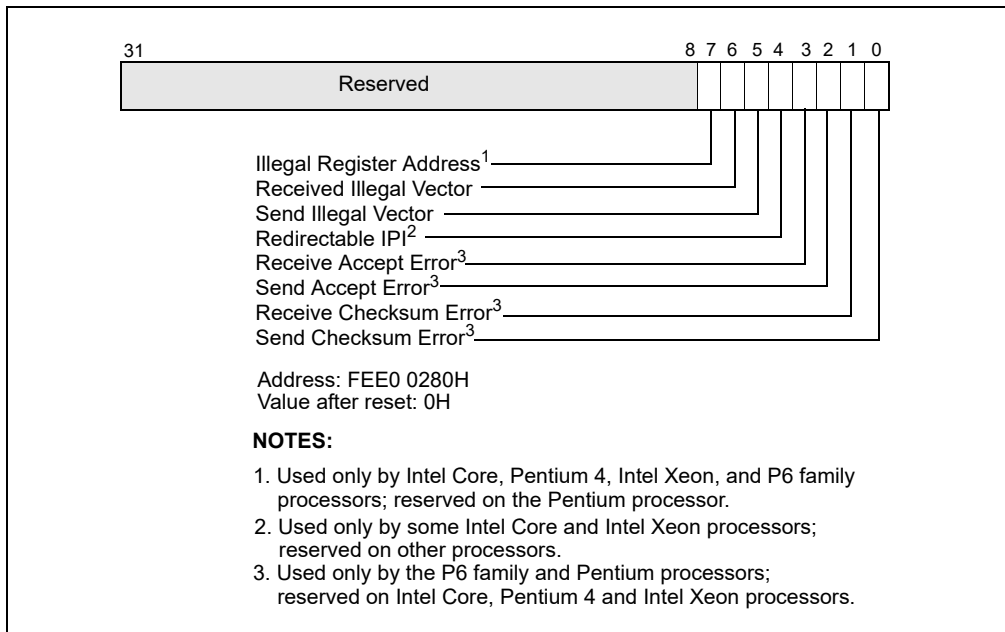


Figure 10-9. Error Status Register (ESR)

- **Bit 0: Send Checksum Error.**  
Set when the local APIC detects a checksum error for a message that it sent on the APIC bus. Used only on P6 family and Pentium processors.
- **Bit 1: Receive Checksum Error.**  
Set when the local APIC detects a checksum error for a message that it received on the APIC bus. Used only on P6 family and Pentium processors.
- **Bit 2: Send Accept Error.**  
Set when the local APIC detects that a message it sent was not accepted by any APIC on the APIC bus. Used only on P6 family and Pentium processors.
- **Bit 3: Receive Accept Error.**  
Set when the local APIC detects that the message it received was not accepted by any APIC on the APIC bus, including itself. Used only on P6 family and Pentium processors.
- **Bit 4: Redirectable IPI.**  
Set when the local APIC detects an attempt to send an IPI with the lowest-priority delivery mode and the local APIC does not support the sending of such IPIs. This bit is used on some Intel Core and Intel Xeon processors. As noted in Section 10.6.2, the ability of a processor to send a lowest-priority IPI is model-specific and should be avoided.

- Bit 5: Send Illegal Vector.**  
 Set when the local APIC detects an illegal vector (one in the range 0 to 15) in the message that it is sending. This occurs as the result of a write to the ICR (in both xAPIC and x2APIC modes) or to SELF IPI register (x2APIC mode only) with an illegal vector.

If the local APIC does not support the sending of lowest-priority IPIs and software writes the ICR to send a lowest-priority IPI with an illegal vector, the local APIC sets only the “redirectable IPI” error bit. The interrupt is not processed and hence the “Send Illegal Vector” bit is not set in the ESR.
- Bit 6: Receive Illegal Vector.**  
 Set when the local APIC detects an illegal vector (one in the range 0 to 15) in an interrupt message it receives or in an interrupt generated locally from the local vector table or via a self IPI. Such interrupts are not delivered to the processor; the local APIC will never set an IRR bit in the range 0 to 15.
- Bit 7: Illegal Register Address**  
 Set when the local APIC is in xAPIC mode and software attempts to access a register that is reserved in the processor’s local-APIC register-address space; see Table 10-1. (The local-APIC register-address space comprises the 4 KBytes at the physical address specified in the IA32\_APIC\_BASE MSR.) Used only on Intel Core, Intel Atom™, Pentium 4, Intel Xeon, and P6 family processors.

In x2APIC mode, software accesses the APIC registers using the RDMSR and WRMSR instructions. Use of one of these instructions to access a reserved register cause a general-protection exception (see Section 10.12.1.3). They do not set the “Illegal Register Access” bit in the ESR.

The ESR is a write/read register. Before attempt to read from the ESR, software should first write to it. (The value written does not affect the values read subsequently; only zero may be written in x2APIC mode.) This write clears any previously logged errors and updates the ESR with any errors detected since the last write to the ESR. This write also rearms the APIC error interrupt triggering mechanism.

The LVT Error Register (see Section 10.5.1) allows specification of the vector of the interrupt to be delivered to the processor core when APIC error is detected. The register also provides a means of masking an APIC-error interrupt. This masking only prevents delivery of APIC-error interrupts; the APIC continues to record errors in the ESR.

### 10.5.4 APIC Timer

The local APIC unit contains a 32-bit programmable timer that is available to software to time events or operations. This timer is set up by programming four registers: the divide configuration register (see Figure 10-10), the initial-count and current-count registers (see Figure 10-11), and the LVT timer register (see Figure 10-8).

If CPUID.06H:EAX.ARAT[bit 2] = 1, the processor’s APIC timer runs at a constant rate regardless of P-state transitions and it continues to run at the same rate in deep C-states.

If CPUID.06H:EAX.ARAT[bit 2] = 0 or if CPUID 06H is not supported, the APIC timer may temporarily stop while the processor is in deep C-states or during transitions caused by Enhanced Intel SpeedStep® Technology.

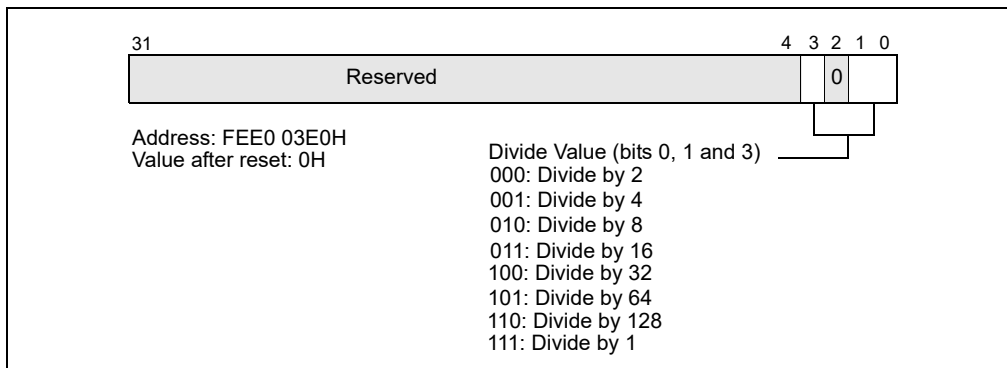
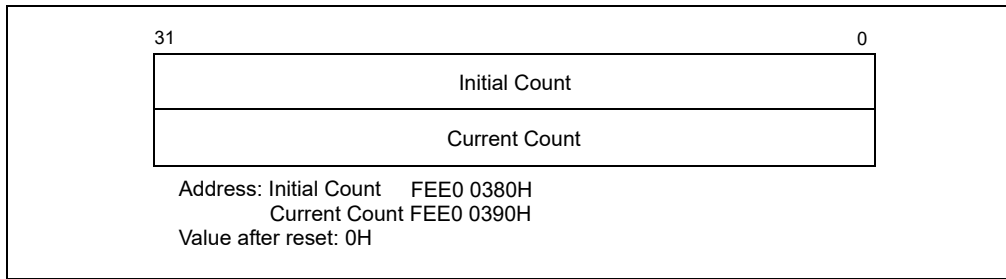


Figure 10-10. Divide Configuration Register

The APIC timer frequency will be the processor’s bus clock or core crystal clock frequency (when TSC/core crystal clock ratio is enumerated in CPUID leaf 0x15) divided by the value specified in the divide configuration register.



**Figure 10-11. Initial Count and Current Count Registers**

The timer can be configured through the timer LVT entry for one-shot or periodic operation. In one-shot mode, the timer is started by programming its initial-count register. The initial count value is then copied into the current-count register and count-down begins. After the timer reaches zero, an timer interrupt is generated and the timer remains at its 0 value until reprogrammed.

In periodic mode, the current-count register is automatically reloaded from the initial-count register when the count reaches 0 and a timer interrupt is generated, and the count-down is repeated. If during the count-down process the initial-count register is set, counting will restart, using the new initial-count value. The initial-count register is a read-write register; the current-count register is read only.

A write of 0 to the initial-count register effectively stops the local APIC timer, in both one-shot and periodic mode.

The LVT timer register determines the vector number that is delivered to the processor with the timer interrupt that is generated when the timer count reaches zero. The mask flag in the LVT timer register can be used to mask the timer interrupt.

### 10.5.4.1 TSC-Deadline Mode

The mode of operation of the local-APIC timer is determined by the LVT Timer Register. Specifically:

- If CPUID.01H:ECX.TSC\_Deadline[bit 24] = 0, the mode is determined by bit 17 of the register.
- If CPUID.01H:ECX.TSC\_Deadline[bit 24] = 1, the mode is determined by bits 18:17. See Figure 10-8. (If CPUID.01H:ECX.TSC\_Deadline[bit 24] = 0, bit 18 of the register is reserved.)

A write to the LVT Timer Register that changes the timer mode disarms the local APIC timer. The supported timer modes are given in Table 10-2. The three modes of the local APIC timer are mutually exclusive.

**Table 10-2. Local APIC Timer Modes**

LVT Bits [18:17]	Timer Mode
00b	One-shot mode, program count-down value in an initial-count register. See Section 10.5.4
01b	Periodic mode, program interval value in an initial-count register. See Section 10.5.4
10b	TSC-Deadline mode, program target value in IA32_TSC_DEADLINE MSR.
11b	Reserved

TSC-deadline mode allows software to use the local APIC timer to signal an interrupt at an absolute time. In TSC-deadline mode, writes to the initial-count register are ignored; and current-count register always reads 0. Instead, timer behavior is controlled using the IA32\_TSC\_DEADLINE MSR.

The IA32\_TSC\_DEADLINE MSR (MSR address 6E0H) is a per-logical processor MSR that specifies the time at which a timer interrupt should occur. Writing a non-zero 64-bit value into IA32\_TSC\_DEADLINE arms the timer. An interrupt is generated when the logical processor’s time-stamp counter equals or exceeds the target value in the IA32\_TSC\_DEADLINE MSR.<sup>1</sup> When the timer generates an interrupt, it disarms itself and clears the IA32\_TSC\_DEADLINE MSR. Thus, each write to the IA32\_TSC\_DEADLINE MSR generates at most one timer interrupt.

In TSC-deadline mode, writing 0 to the IA32\_TSC\_DEADLINE MSR disarms the local-APIC timer. Transitioning between TSC-deadline mode and other timer modes also disarms the timer.

The hardware reset value of the IA32\_TSC\_DEADLINE MSR is 0. In other timer modes (LVT bit 18 = 0), the IA32\_TSC\_DEADLINE MSR reads zero and writes are ignored.

Software can configure the TSC-deadline timer to deliver a single interrupt using the following algorithm:

1. Detect support for TSC-deadline mode by verifying `CPUID.1:ECX.24 = 1`.
2. Select the TSC-deadline mode by programming bits 18:17 of the LVT Timer register with 10b.
3. Program the IA32\_TSC\_DEADLINE MSR with the target TSC value at which the timer interrupt is desired. This causes the processor to arm the timer.
4. The processor generates a timer interrupt when the value of time-stamp counter is greater than or equal to that of IA32\_TSC\_DEADLINE. It then disarms the timer and clear the IA32\_TSC\_DEADLINE MSR. (Both the time-stamp counter and the IA32\_TSC\_DEADLINE MSR are 64-bit unsigned integers.)
5. Software can re-arm the timer by repeating step 3.

The following are usage guidelines for TSC-deadline mode:

- Writes to the IA32\_TSC\_DEADLINE MSR are not serialized. Therefore, system software should not use WRMSR to the IA32\_TSC\_DEADLINE MSR as a serializing instruction. Read and write accesses to the IA32\_TSC\_DEADLINE and other MSR registers will occur in program order.
- Software can disarm the timer at any time by writing 0 to the IA32\_TSC\_DEADLINE MSR.
- If timer is armed, software can change the deadline (forward or backward) by writing a new value to the IA32\_TSC\_DEADLINE MSR.
- If software disarms the timer or postpones the deadline, race conditions may result in the delivery of a spurious timer interrupt. Software is expected to detect such spurious interrupts by checking the current value of the time-stamp counter to confirm that the interrupt was desired.<sup>1</sup>
- In xAPIC mode (in which the local-APIC registers are memory-mapped), software must order the memory-mapped write to the LVT entry that enables TSC-deadline mode and any subsequent WRMSR to the IA32\_TSC\_DEADLINE MSR. Software can assure proper ordering by executing the MFENCE instruction after the memory-mapped write and before any WRMSR. (In x2APIC mode, the WRMSR instruction is used to write to the LVT entry. The processor ensures the ordering of this write and any subsequent WRMSR to the deadline; no fencing is required.)

### 10.5.5 Local Interrupt Acceptance

When a local interrupt is sent to the processor core, it is subject to the acceptance criteria specified in the interrupt acceptance flow chart in Figure 10-17. If the interrupt is accepted, it is logged into the IRR register and handled by the processor according to its priority (see Section 10.8.4, "Interrupt Acceptance for Fixed Interrupts"). If the interrupt is not accepted, it is sent back to the local APIC and retried.

## 10.6 ISSUING INTERPROCESSOR INTERRUPTS

The following sections describe the local APIC facilities that are provided for issuing interprocessor interrupts (IPIs) from software. The primary local APIC facility for issuing IPIs is the interrupt command register (ICR). The ICR can be used for the following functions:

- 
1. If the logical processor is in VMX non-root operation, a read of the time-stamp counter (using either RDMSR, RDTSC, or RDTSCP) may not return the actual value of the time-stamp counter; see Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. It is the responsibility of software operating in VMX root operation to coordinate the virtualization of the time-stamp counter and the IA32\_TSC\_DEADLINE MSR.
  1. If the logical processor is in VMX non-root operation, a read of the time-stamp counter (using either RDMSR, RDTSC, or RDTSCP) may not return the actual value of the time-stamp counter; see Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. It is the responsibility of software operating in VMX root operation to coordinate the virtualization of the time-stamp counter and the IA32\_TSC\_DEADLINE MSR.

- To send an interrupt to another processor.
- To allow a processor to forward an interrupt that it received but did not service to another processor for servicing.
- To direct the processor to interrupt itself (perform a self interrupt).
- To deliver special IPIs, such as the start-up IPI (SIPI) message, to other processors.

Interrupts generated with this facility are delivered to the other processors in the system through the system bus (for Pentium 4 and Intel Xeon processors) or the APIC bus (for P6 family and Pentium processors). The ability for a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.

### 10.6.1 Interrupt Command Register (ICR)

The interrupt command register (ICR) is a 64-bit<sup>1</sup> local APIC register (see Figure 10-12) that allows software running on the processor to specify and send interprocessor interrupts (IPIs) to other processors in the system.

To send an IPI, software must set up the ICR to indicate the type of IPI message to be sent and the destination processor or processors. (All fields of the ICR are read-write by software with the exception of the delivery status field, which is read-only.) The act of writing to the low doubleword of the ICR causes the IPI to be sent.

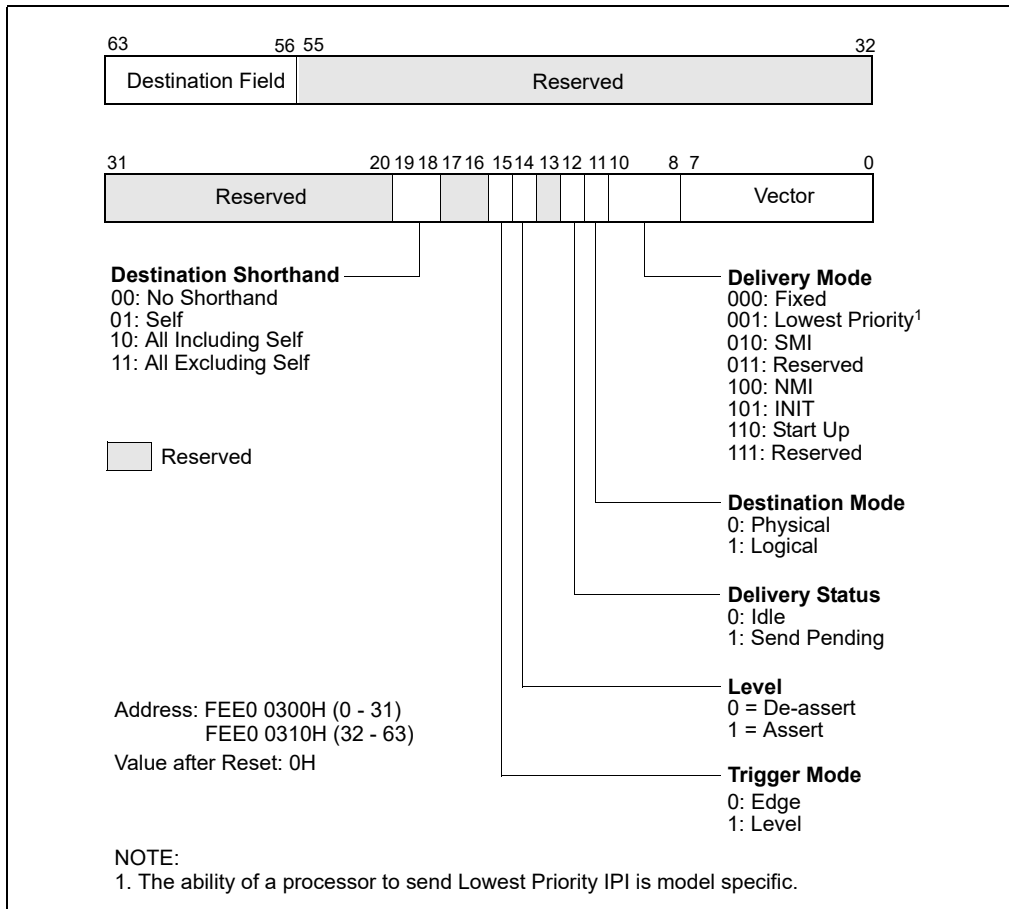


Figure 10-12. Interrupt Command Register (ICR)

1. In XAPIC mode the ICR is addressed as two 32-bit registers, ICR\_LOW (FEE0 0300H) and ICR\_HIGH (FEE0 0310H). In x2APIC mode, the ICR uses MSR 830H.

The ICR consists of the following fields.

<b>Vector</b>	The vector number of the interrupt being sent.
<b>Delivery Mode</b>	Specifies the type of IPI to be sent. This field is also know as the IPI message type field. <ul style="list-style-type: none"> <li><b>000 (Fixed)</b> Delivers the interrupt specified in the vector field to the target processor or processors.</li> <li><b>001 (Lowest Priority)</b> Same as fixed mode, except that the interrupt is delivered to the processor executing at the lowest priority among the set of processors specified in the destination field. The ability for a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.</li> <li><b>010 (SMI)</b> Delivers an SMI interrupt to the target processor or processors. The vector field must be programmed to 00H for future compatibility.</li> <li><b>011 (Reserved)</b></li> <li><b>100 (NMI)</b> Delivers an NMI interrupt to the target processor or processors. The vector information is ignored.</li> <li><b>101 (INIT)</b> Delivers an INIT request to the target processor or processors, which causes them to perform an INIT. As a result of this IPI message, all the target processors perform an INIT. The vector field must be programmed to 00H for future compatibility.</li> <li><b>101 (INIT Level De-assert)</b> (Not supported in the Pentium 4 and Intel Xeon processors.) Sends a synchronization message to all the local APICs in the system to set their arbitration IDs (stored in their Arb ID registers) to the values of their APIC IDs (see Section 10.7, "System and APIC Bus Arbitration"). For this delivery mode, the level flag must be set to 0 and trigger mode flag to 1. This IPI is sent to all processors, regardless of the value in the destination field or the destination shorthand field; however, software should specify the "all including self" shorthand.</li> <li><b>110 (Start-Up)</b> Sends a special "start-up" IPI (called a SIPI) to the target processor or processors. The vector typically points to a start-up routine that is part of the BIOS boot-strap code (see Section 8.4, "Multiple-Processor (MP) Initialization"). IPIs sent with this delivery mode are not automatically retried if the source APIC is unable to deliver it. It is up to the software to determine if the SIPI was not successfully delivered and to reissue the SIPI if necessary.</li> </ul>
<b>Destination Mode</b>	Selects either physical (0) or logical (1) destination mode (see Section 10.6.2, "Determining IPI Destination").
<b>Delivery Status (Read Only)</b>	Indicates the IPI delivery status, as follows: <ul style="list-style-type: none"> <li><b>0 (Idle)</b> Indicates that this local APIC has completed sending any previous IPIs.</li> <li><b>1 (Send Pending)</b> Indicates that this local APIC has not completed sending the last IPI.</li> </ul>
<b>Level</b>	For the INIT level de-assert delivery mode this flag must be set to 0; for all other delivery modes it must be set to 1. (This flag has no meaning in Pentium 4 and Intel Xeon processors, and will always be issued as a 1.)



**Trigger Mode** Selects the trigger mode when using the INIT level de-assert delivery mode: edge (0) or level (1). It is ignored for all other delivery modes. (This flag has no meaning in Pentium 4 and Intel Xeon processors, and will always be issued as a 0.)

#### Destination Shorthand

Indicates whether a shorthand notation is used to specify the destination of the interrupt and, if so, which shorthand is used. Destination shorthands are used in place of the 8-bit destination field, and can be sent by software using a single write to the low doubleword of the ICR. Shorthands are defined for the following cases: software self interrupt, IPIs to all processors in the system including the sender, IPIs to all processors in the system excluding the sender.

##### 00: (No Shorthand)

The destination is specified in the destination field.

##### 01: (Self)

The issuing APIC is the one and only destination of the IPI. This destination shorthand allows software to interrupt the processor on which it is executing. An APIC implementation is free to deliver the self-interrupt message internally or to issue the message to the bus and “snoop” it as with any other IPI message.

##### 10: (All Including Self)

The IPI is sent to all processors in the system including the processor sending the IPI. The APIC will broadcast an IPI message with the destination field set to FH for Pentium and P6 family processors and to FFH for Pentium 4 and Intel Xeon processors.

##### 11: (All Excluding Self)

The IPI is sent to all processors in a system with the exception of the processor sending the IPI. The APIC broadcasts a message with the physical destination mode and destination field set to FH for Pentium and P6 family processors and to FFH for Pentium 4 and Intel Xeon processors. Support for this destination shorthand in conjunction with the lowest-priority delivery mode is model specific. For Pentium 4 and Intel Xeon processors, when this shorthand is used together with lowest priority delivery mode, the IPI may be redirected back to the issuing processor.

**Destination** Specifies the target processor or processors. This field is only used when the destination shorthand field is set to 00B. If the destination mode is set to physical, then bits 56 through 59 contain the APIC ID of the target processor for Pentium and P6 family processors and bits 56 through 63 contain the APIC ID of the target processor the for Pentium 4 and Intel Xeon processors. If the destination mode is set to logical, the interpretation of the 8-bit destination field depends on the settings of the DFR and LDR registers of the local APICs in all the processors in the system (see Section 10.6.2, “Determining IPI Destination”).

Not all combinations of options for the ICR are valid. Table 10-3 shows the valid combinations for the fields in the ICR for the Pentium 4 and Intel Xeon processors; Table 10-4 shows the valid combinations for the fields in the ICR for the P6 family processors. Also note that the lower half of the ICR may not be preserved over transitions to the deepest C-States.

ICR operation in x2APIC mode is discussed in Section 10.12.9.



**Table 10-3 Valid Combinations for the Pentium 4 and Intel Xeon Processors’ Local xAPIC Interrupt Command Register**

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
No Shorthand	Valid	Edge	All Modes <sup>1</sup>	Physical or Logical
No Shorthand	Invalid <sup>2</sup>	Level	All Modes	Physical or Logical
Self	Valid	Edge	Fixed	X <sup>3</sup>
Self	Invalid <sup>2</sup>	Level	Fixed	X
Self	Invalid	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All Including Self	Valid	Edge	Fixed	X
All Including Self	Invalid <sup>2</sup>	Level	Fixed	X
All Including Self	Invalid	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All Excluding Self	Valid	Edge	Fixed, Lowest Priority <sup>1,4</sup> , NMI, INIT, SMI, Start-Up	X
All Excluding Self	Invalid <sup>2</sup>	Level	Fixed, Lowest Priority <sup>4</sup> , NMI, INIT, SMI, Start-Up	X

**NOTES:**

1. The ability of a processor to send a lowest priority IPI is model specific.
2. For these interrupts, if the trigger mode bit is 1 (Level), the local xAPIC will override the bit setting and issue the interrupt as an edge triggered interrupt.
3. X means the setting is ignored.
4. When using the “lowest priority” delivery mode and the “all excluding self” destination, the IPI can be redirected back to the issuing APIC, which is essentially the same as the “all including self” destination mode.

**Table 10-4 Valid Combinations for the P6 Family Processors’ Local APIC Interrupt Command Register**

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
No Shorthand	Valid	Edge	All Modes <sup>1</sup>	Physical or Logical
No Shorthand	Valid <sup>2</sup>	Level	Fixed, Lowest Priority <sup>1</sup> , NMI	Physical or Logical
No Shorthand	Valid <sup>3</sup>	Level	INIT	Physical or Logical
Self	Valid	Edge	Fixed	X <sup>4</sup>
Self	Valid <sup>2</sup>	Level	Fixed	X
Self	Invalid <sup>5</sup>	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All including Self	Valid	Edge	Fixed	X
All including Self	Valid <sup>2</sup>	Level	Fixed	X
All including Self	Invalid <sup>5</sup>	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All excluding Self	Valid	Edge	All Modes <sup>1</sup>	X
All excluding Self	Valid <sup>2</sup>	Level	Fixed, Lowest Priority <sup>1</sup> , NMI	X
All excluding Self	Invalid <sup>5</sup>	Level	SMI, Start-Up	X
All excluding Self	Valid <sup>3</sup>	Level	INIT	X
X	Invalid <sup>5</sup>	Level	SMI, Start-Up	X

**NOTES:**

1. The ability of a processor to send a lowest priority IPI is model specific.
2. Treated as edge triggered if level bit is set to 1, otherwise ignored.
3. Treated as edge triggered when Level bit is set to 1; treated as “INIT Level Deassert” message when level bit is set to 0 (deassert). Only INIT level deassert messages are allowed to have the level bit set to 0. For all other messages the level bit must be set to 1.
4. X means the setting is ignored.
5. The behavior of the APIC is undefined.

## 10.6.2 Determining IPI Destination

The destination of an IPI<sup>1</sup> can be one, all, or a subset (group) of the processors on the system bus. The sender of the IPI specifies the destination of an IPI with the following APIC registers and fields within the registers:

- **ICR Register** — The following fields in the ICR register are used to specify the destination of an IPI.
  - **Destination Mode** — Selects one of two destination modes (physical or logical).
  - **Destination Field** — In physical destination mode, used to specify the APIC ID of the destination processor; in logical destination mode, used to specify a message destination address (MDA) that can be used to select specific processors in clusters.
  - **Destination Shorthand** — A quick method of specifying all processors, all excluding self, or self as the destination.
  - **Delivery mode, Lowest Priority** — Architecturally specifies that a lowest-priority arbitration mechanism be used to select a destination processor from a specified group of processors. The ability of a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.
- **Local destination register (LDR)** — Used in conjunction with the logical destination mode and MDAs to select the destination processors.
- **Destination format register (DFR)** — Used in conjunction with the logical destination mode and MDAs to select the destination processors.

How the ICR, LDR, and DFR are used to select an IPI destination depends on the destination mode used: physical, logical, broadcast/self, or lowest-priority delivery mode. These destination modes are described in the following sections.

### 10.6.2.1 Physical Destination Mode

In physical destination mode, the destination processor is specified by its local APIC ID (see Section 10.4.6, “Local APIC ID”). For Pentium 4 and Intel Xeon processors, either a single destination (local APIC IDs 00H through FEH) or a broadcast to all APICs (the APIC ID is FFH) may be specified in physical destination mode.

A broadcast IPI (bits 28-31 of the MDA are 1's) or I/O subsystem initiated interrupt with lowest priority delivery mode is not supported in physical destination mode and must not be configured by software. Also, for any non-broadcast IPI or I/O subsystem initiated interrupt with lowest priority delivery mode, software must ensure that APICs defined in the interrupt address are present and enabled to receive interrupts.

For the P6 family and Pentium processors, a single destination is specified in physical destination mode with a local APIC ID of 0H through 0EH, allowing up to 15 local APICs to be addressed on the APIC bus. A broadcast to all local APICs is specified with 0FH.

#### NOTE

The number of local APICs that can be addressed on the system bus may be restricted by hardware.

### 10.6.2.2 Logical Destination Mode

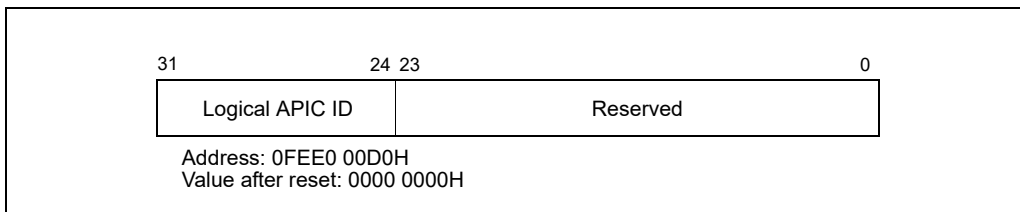
In logical destination mode, IPI destination is specified using an 8-bit message destination address (MDA), which is entered in the destination field of the ICR. Upon receiving an IPI message that was sent using logical destination mode, a local APIC compares the MDA in the message with the values in its LDR and DFR to determine if it should accept and handle the IPI. For both configurations of logical destination mode, when combined with lowest priority delivery mode, software is responsible for ensuring that all of the local APICs included in or addressed by the IPI or I/O subsystem interrupt are present and enabled to receive the interrupt.

Figure 10-13 shows the layout of the logical destination register (LDR). The 8-bit logical APIC ID field in this register is used to create an identifier that can be compared with the MDA.

1. Determination of IPI destinations in x2APIC mode is discussed in Section 10.12.10.

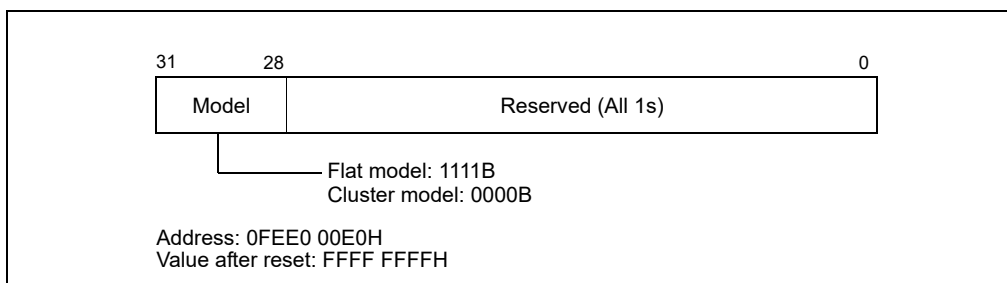
**NOTE**

The logical APIC ID should not be confused with the local APIC ID that is contained in the local APIC ID register.



**Figure 10-13. Logical Destination Register (LDR)**

Figure 10-14 shows the layout of the destination format register (DFR). The 4-bit model field in this register selects one of two models (flat or cluster) that can be used to interpret the MDA when using logical destination mode.



**Figure 10-14. Destination Format Register (DFR)**

The interpretation of MDA for the two models is described in the following paragraphs.

1. **Flat Model** — This model is selected by programming DFR bits 28 through 31 to 1111. Here, a unique logical APIC ID can be established for up to 8 local APICs by setting a different bit in the logical APIC ID field of the LDR for each local APIC. A group of local APICs can then be selected by setting one or more bits in the MDA. Each local APIC performs a bit-wise AND of the MDA and its logical APIC ID. If a true condition (non-zero) is detected, the local APIC accepts the IPI message. A broadcast to all APICs is achieved by setting the MDA to 1s.
2. **Cluster Model** — This model is selected by programming DFR bits 28 through 31 to 0000. This model supports two basic destination schemes: flat cluster and hierarchical cluster.

The flat cluster destination model is only supported for P6 family and Pentium processors. Using this model, all APICs are assumed to be connected through the APIC bus. Bits 60 through 63 of the MDA contains the encoded address of the destination cluster and bits 56 through 59 identify up to four local APICs within the cluster (each bit is assigned to one local APIC in the cluster, as in the flat connection model). To identify one or more local APICs, bits 60 through 63 of the MDA are compared with bits 28 through 31 of the LDR to determine if a local APIC is part of the cluster. Bits 56 through 59 of the MDA are compared with Bits 24 through 27 of the LDR to identify a local APICs within the cluster.

Sets of processors within a cluster can be specified by writing the target cluster address in bits 60 through 63 of the MDA and setting selected bits in bits 56 through 59 of the MDA, corresponding to the chosen members of the cluster. In this mode, 15 clusters (with cluster addresses of 0 through 14) each having 4 local APICs can be specified in the message. For the P6 and Pentium processor’s local APICs, however, the APIC arbitration ID supports only 15 APIC agents. Therefore, the total number of processors and their local APICs supported in this mode is limited to 15. Broadcast to all local APICs is achieved by setting all destination bits to one. This guarantees a match on all clusters and selects all APICs in each cluster. A broadcast IPI or I/O subsystem broadcast interrupt with lowest priority delivery mode is not supported in cluster mode and must not be configured by software.

The hierarchical cluster destination model can be used with Pentium 4, Intel Xeon, P6 family, or Pentium processors. With this model, a hierarchical network can be created by connecting different flat clusters via

independent system or APIC buses. This scheme requires a cluster manager within each cluster, which is responsible for handling message passing between system or APIC buses. One cluster contains up to 4 agents. Thus 15 cluster managers, each with 4 agents, can form a network of up to 60 APIC agents. Note that hierarchical APIC networks requires a special cluster manager device, which is not part of the local or the I/O APIC units.

### NOTES

All processors that have their APIC software enabled (using the spurious vector enable/disable bit) must have their DFRs (Destination Format Registers) programmed identically.

The default mode for DFR is flat mode. If you are using cluster mode, DFRs must be programmed before the APIC is software enabled. Since some chipsets do not accurately track a system view of the logical mode, program DFRs as soon as possible after starting the processor.

#### 10.6.2.3 Broadcast/Self Delivery Mode

The destination shorthand field of the ICR allows the delivery mode to be by-passed in favor of broadcasting the IPI to all the processors on the system bus and/or back to itself (see Section 10.6.1, "Interrupt Command Register (ICR)"). Three destination shorthands are supported: self, all excluding self, and all including self. The destination mode is ignored when a destination shorthand is used.

#### 10.6.2.4 Lowest Priority Delivery Mode

With lowest priority delivery mode, the ICR is programmed to send an IPI to several processors on the system bus, using the logical or shorthand destination mechanism for selecting the processor. The selected processors then arbitrate with one another over the system bus or the APIC bus, with the lowest-priority processor accepting the IPI.

For systems based on the Intel Xeon processor, the chipset bus controller accepts messages from the I/O APIC agents in the system and directs interrupts to the processors on the system bus. When using the lowest priority delivery mode, the chipset chooses a target processor to receive the interrupt out of the set of possible targets. The Pentium 4 processor provides a special bus cycle on the system bus that informs the chipset of the current task priority for each logical processor in the system. The chipset saves this information and uses it to choose the lowest priority processor when an interrupt is received.

For systems based on P6 family processors, the processor priority used in lowest-priority arbitration is contained in the arbitration priority register (APR) in each local APIC. Figure 10-15 shows the layout of the APR.

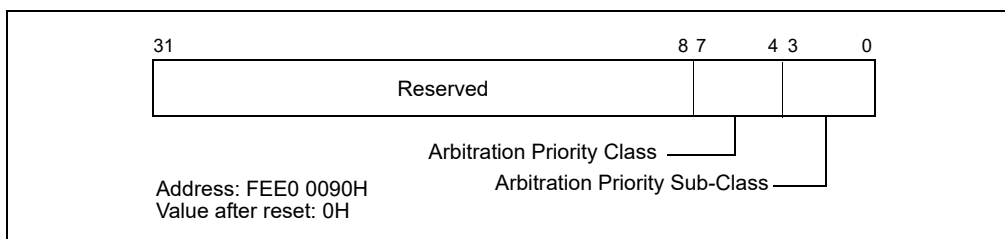


Figure 10-15. Arbitration Priority Register (APR)

The APR value is computed as follows:

```

IF (TPR[7:4] ≥ IRRV[7:4]) AND (TPR[7:4] > ISRV[7:4])
  THEN
    APR[7:0] ← TPR[7:0]
  ELSE
    APR[7:4] ← max(TPR[7:4] AND ISRV[7:4], IRRV[7:4])
    APR[3:0] ← 0.
    
```

Here, the TPR value is the task priority value in the TPR (see Figure 10-18), the IRRV value is the vector number for the highest priority bit that is set in the IRR (see Figure 10-20) or 00H (if no IRR bit is set), and the ISRV value is the vector number for the highest priority bit that is set in the ISR (see Figure 10-20). Following arbitration among the destination processors, the processor with the lowest value in its APR handles the IPI and the other processors ignore it.

(P6 family and Pentium processors.) For these processors, if a **focus processor** exists, it may accept the interrupt, regardless of its priority. A processor is said to be the focus of an interrupt if it is currently servicing that interrupt or if it has a pending request for that interrupt. For Intel Xeon processors, the concept of a focus processor is not supported.

In operating systems that use the lowest priority delivery mode but do not update the TPR, the TPR information saved in the chipset will potentially cause the interrupt to be always delivered to the same processor from the logical set. This behavior is functionally backward compatible with the P6 family processor but may result in unexpected performance implications.

### 10.6.3 IPI Delivery and Acceptance

When the low double-word of the ICR is written to, the local APIC creates an IPI message from the information contained in the ICR and sends the message out on the system bus (Pentium 4 and Intel Xeon processors) or the APIC bus (P6 family and Pentium processors). The manner in which these IPIs are handled after being issues in described in Section 10.8, "Handling Interrupts."

## 10.7 SYSTEM AND APIC BUS ARBITRATION

When several local APICs and the I/O APIC are sending IPI and interrupt messages on the system bus (or APIC bus), the order in which the messages are sent and handled is determined through bus arbitration.

For the Pentium 4 and Intel Xeon processors, the local and I/O APICs use the arbitration mechanism defined for the system bus to determine the order in which IPIs are handled. This mechanism is non-architectural and cannot be controlled by software.

For the P6 family and Pentium processors, the local and I/O APICs use an APIC-based arbitration mechanism to determine the order in which IPIs are handled. Here, each local APIC is given an arbitration priority of from 0 to 15, which the I/O APIC uses during arbitration to determine which local APIC should be given access to the APIC bus. The local APIC with the highest arbitration priority always wins bus access. Upon completion of an arbitration round, the winning local APIC lowers its arbitration priority to 0 and the losing local APICs each raise theirs by 1.

The current arbitration priority for a local APIC is stored in a 4-bit, software-transparent arbitration ID (Arb ID) register. During reset, this register is initialized to the APIC ID number (stored in the local APIC ID register). The INIT level-deassert IPI, which is issued with an ICR command, can be used to resynchronize the arbitration priorities of the local APICs by resetting Arb ID register of each agent to its current APIC ID value. (The Pentium 4 and Intel Xeon processors do not implement the Arb ID register.)

Section 10.10, "APIC Bus Message Passing Mechanism and Protocol (P6 Family, Pentium Processors)," describes the APIC bus arbitration protocols and bus message formats, while Section 10.6.1, "Interrupt Command Register (ICR)," describes the INIT level de-assert IPI message.

Note that except for the SIPI IPI (see Section 10.6.1, "Interrupt Command Register (ICR)"), all bus messages that fail to be delivered to their specified destination or destinations are automatically retried. Software should avoid situations in which IPIs are sent to disabled or nonexistent local APICs, causing the messages to be resent repeatedly. Additionally, interrupt sources that target the APIC should be masked or changed to no longer target the APIC.

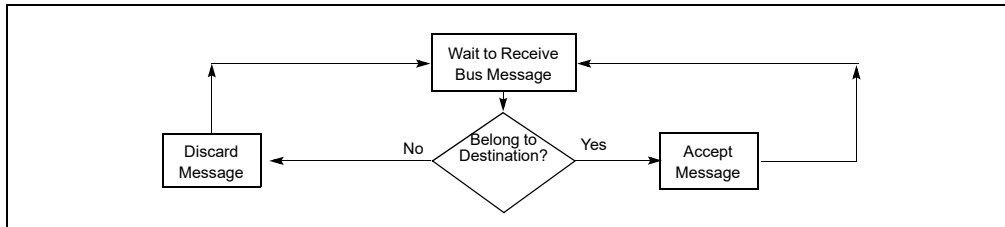
## 10.8 HANDLING INTERRUPTS

When a local APIC receives an interrupt from a local source, an interrupt message from an I/O APIC, or an IPI, the manner in which it handles the message depends on processor implementation, as described in the following sections.

## 10.8.1 Interrupt Handling with the Pentium 4 and Intel Xeon Processors

With the Pentium 4 and Intel Xeon processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows:

1. It determines if it is the specified destination or not (see Figure 10-16). If it is the specified destination, it accepts the message; if it is not, it discards the message.



**Figure 10-16. Interrupt Acceptance Flow Chart for the Local APIC (Pentium 4 and Intel Xeon Processors)**

2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or SIPI, the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC sets the appropriate bit in the IRR.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 10.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI) register in the local APIC (see Section 10.8.5, "Signaling Interrupt Servicing Completion"). The act of writing to the EOI register causes the local APIC to delete the interrupt from its ISR queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

## 10.8.2 Interrupt Handling with the P6 Family and Pentium Processors

With the P6 family and Pentium processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows (see Figure 10-17).

1. (IPIs only) The local APIC examines the IPI message to determine if it is the specified destination for the IPI as described in Section 10.6.2, "Determining IPI Destination." If it is the specified destination, it continues its acceptance procedure; if it is not the destination, it discards the IPI message. When the message specifies lowest-priority delivery mode, the local APIC will arbitrate with the other processors that were designated as recipients of the IPI message (see Section 10.6.2.4, "Lowest Priority Delivery Mode").
2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or INIT-deassert interrupt, or one of the MP protocol IPI messages (BIPI, FIPI, and SIPI), the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC looks for an open slot in one of its two pending interrupt queues contained in the IRR and ISR registers (see Figure 10-20). If a slot is available (see Section 10.8.4, "Interrupt Acceptance for Fixed Interrupts"), places the interrupt in the slot. If a slot is not available, it rejects the interrupt request and sends it back to the sender with a retry message.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 10.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI) register in the local APIC.

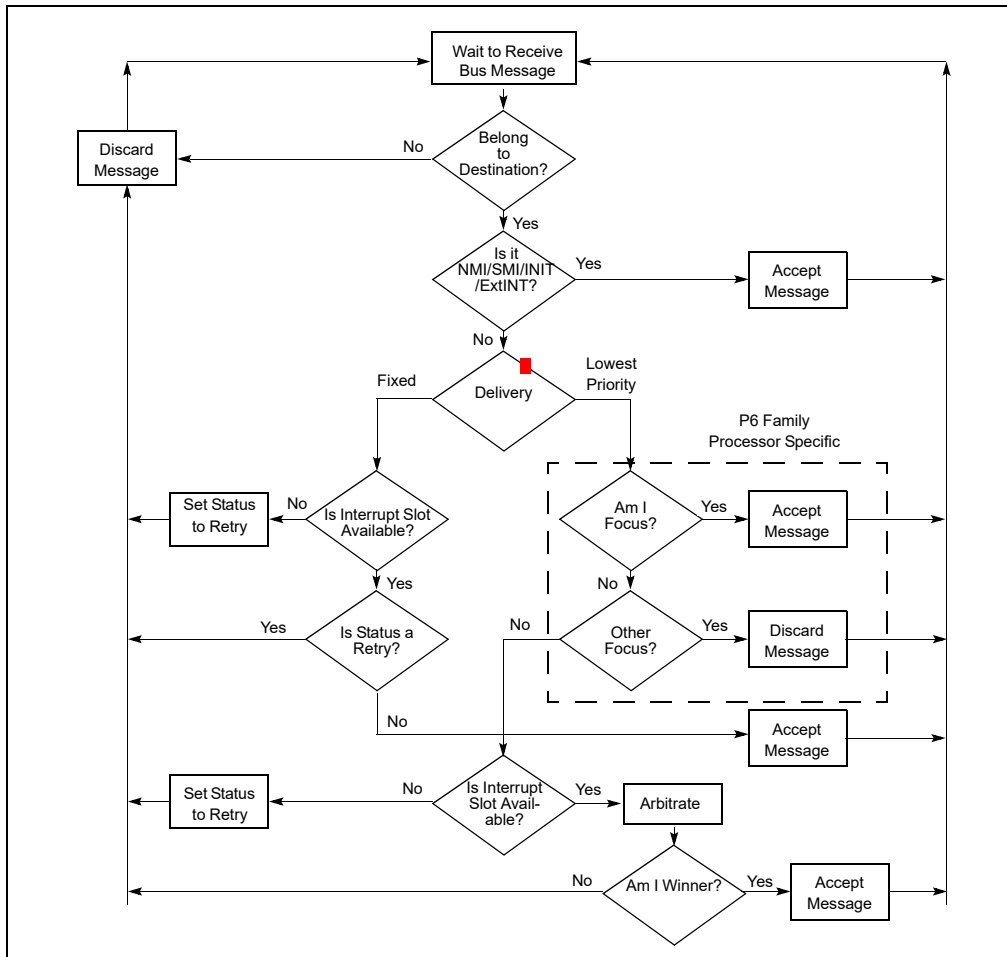


Figure 10-17. Interrupt Acceptance Flow Chart for the Local APIC (P6 Family and Pentium Processors)

register in the local APIC (see Section 10.8.5, “Signaling Interrupt Servicing Completion”). The act of writing to the EOI register causes the local APIC to delete the interrupt from its queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

The following sections describe the acceptance of interrupts and their handling by the local APIC and processor in greater detail.

### 10.8.3 Interrupt, Task, and Processor Priority

Each interrupt delivered to the processor through the local APIC has a priority based on its vector number. The local APIC uses this priority to determine when to service the interrupt relative to the other activities of the processor, including the servicing of other interrupts.

Each interrupt vector is an 8-bit value. The **interrupt-priority class** is the value of bits 7:4 of the interrupt vector. The lowest interrupt-priority class is 1 and the highest is 15; interrupts with vectors in the range 0–15 (with interrupt-priority class 0) are illegal and are never delivered. Because vectors 0–31 are reserved for dedicated uses by the Intel 64 and IA-32 architectures, software should configure interrupt vectors to use interrupt-priority classes in the range 2–15.

Each interrupt-priority class encompasses 16 vectors. The relative priority of interrupts within an interrupt-priority class is determined by the value of bits 3:0 of the vector number. The higher the value of those bits, the higher the



priority within that interrupt-priority class. Thus, each interrupt vector comprises two parts, with the high 4 bits indicating its interrupt-priority class and the low 4 bits indicating its ranking within the interrupt-priority class.

### 10.8.3.1 Task and Processor Priorities

The local APIC also defines a **task priority** and a **processor priority** that determine the order in which interrupts are handled. The **task-priority class** is the value of bits 7:4 of the task-priority register (TPR), which can be written by software (TPR is a read/write register); see Figure 10-18.

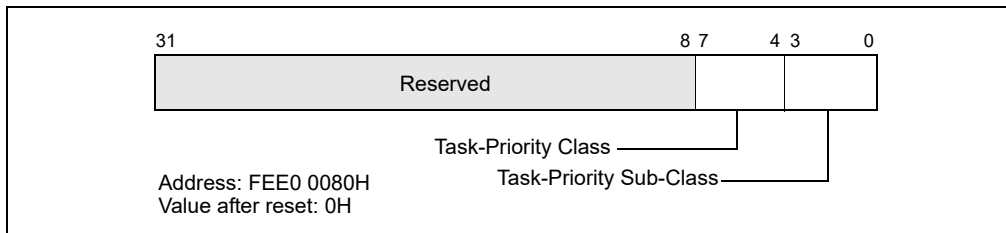


Figure 10-18. Task-Priority Register (TPR)

#### NOTE

In this discussion, the term “task” refers to a software defined task, process, thread, program, or routine that is dispatched to run on the processor by the operating system. It does not refer to an IA-32 architecture defined task as described in Chapter 7, “Task Management.”

The task priority allows software to set a priority threshold for interrupting the processor. This mechanism enables the operating system to temporarily block low priority interrupts from disturbing high-priority work that the processor is doing. The ability to block such interrupts using task priority results from the way that the TPR controls the value of the processor-priority register (PPR).<sup>1</sup>

The **processor-priority class** is a value in the range 0–15 that is maintained in bits 7:4 of the processor-priority register (PPR); see Figure 10-19. The PPR is a read-only register. The processor-priority class represents the current priority at which the processor is executing.

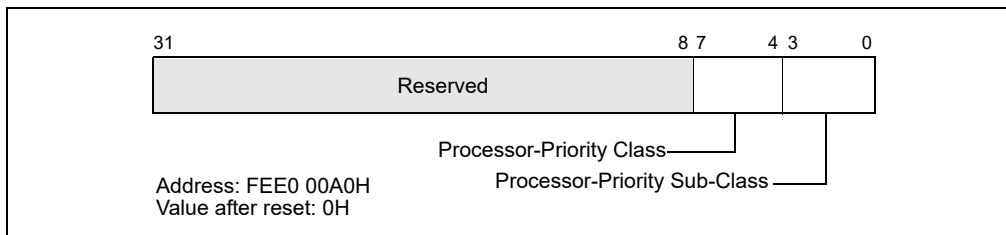


Figure 10-19. Processor-Priority Register (PPR)

The value of the PPR is based on the value of TPR and the value ISRV; ISRV is the vector number of the highest priority bit that is set in the ISR or 00H if no bit is set in the ISR. (See Section 10.8.4 for more details on the ISR.) The value of PPR is determined as follows:

- PPR[7:4] (the processor-priority class) the maximum of TPR[7:4] (the task- priority class) and ISRV[7:4] (the priority of the highest priority interrupt in service).
- PPR[3:0] (the processor-priority sub-class) is determined as follows:
  - If TPR[7:4] > ISRV[7:4], PPR[3:0] is TPR[3:0] (the task-priority sub-class).
  - If TPR[7:4] < ISRV[7:4], PPR[3:0] is 0.
  - If TPR[7:4] = ISRV[7:4], PPR[3:0] may be either TPR[3:0] or 0. The actual behavior is model-specific.

1. The TPR also determines the arbitration priority of the local processor; see Section 10.6.2.4, “Lowest Priority Delivery Mode.”



The processor-priority class determines the priority threshold for interrupting the processor. The processor will deliver only those interrupts that have an interrupt-priority class higher than the processor-priority class in the PPR. If the processor-priority class is 0, the PPR does not inhibit the delivery any interrupt; if it is 15, the processor inhibits the delivery of all interrupts. (The processor-priority mechanism does not affect the delivery of interrupts with the NMI, SMI, INIT, ExtINT, INIT-deassert, and start-up delivery modes.)

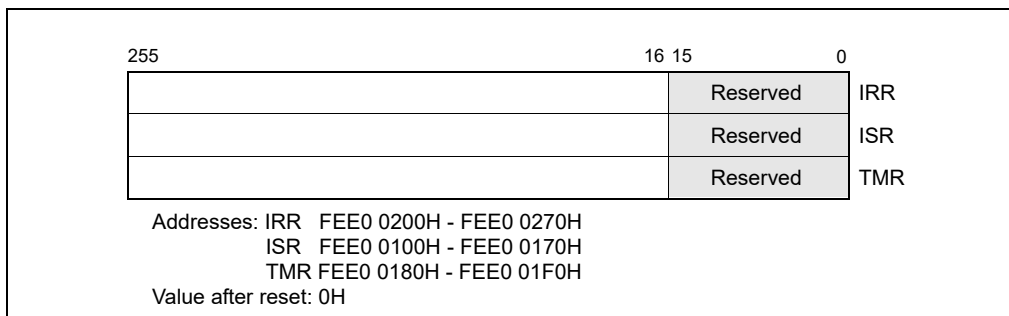
The processor does not use the processor-priority sub-class to determine which interrupts to delivery and which to inhibit. (The processor uses the processor-priority sub-class only to satisfy reads of the PPR.)

### 10.8.4 Interrupt Acceptance for Fixed Interrupts

The local APIC queues the fixed interrupts that it accepts in one of two interrupt pending registers: the interrupt request register (IRR) or in-service register (ISR). These two 256-bit read-only registers are shown in Figure 10-20. The 256 bits in these registers represent the 256 possible vectors; vectors 0 through 15 are reserved by the APIC (see also: Section 10.5.2, "Valid Interrupt Vectors").

**NOTE**

All interrupts with an NMI, SMI, INIT, ExtINT, start-up, or INIT-deassert delivery mode bypass the IRR and ISR registers and are sent directly to the processor core for servicing.



**Figure 10-20. IRR, ISR and TMR Registers**

The IRR contains the active interrupt requests that have been accepted, but not yet dispatched to the processor for servicing. When the local APIC accepts an interrupt, it sets the bit in the IRR that corresponds the vector of the accepted interrupt. When the processor core is ready to handle the next interrupt, the local APIC clears the highest priority IRR bit that is set and sets the corresponding ISR bit. The vector for the highest priority bit set in the ISR is then dispatched to the processor core for servicing.

While the processor is servicing the highest priority interrupt, the local APIC can send additional fixed interrupts by setting bits in the IRR. When the interrupt service routine issues a write to the EOI register (see Section 10.8.5, "Signaling Interrupt Servicing Completion"), the local APIC responds by clearing the highest priority ISR bit that is set. It then repeats the process of clearing the highest priority bit in the IRR and setting the corresponding bit in the ISR. The processor core then begins executing the service routing for the highest priority bit set in the ISR.

If more than one interrupt is generated with the same vector number, the local APIC can set the bit for the vector both in the IRR and the ISR. This means that for the Pentium 4 and Intel Xeon processors, the IRR and ISR can queue two interrupts for each interrupt vector: one in the IRR and one in the ISR. Any additional interrupts issued for the same interrupt vector are collapsed into the single bit in the IRR.

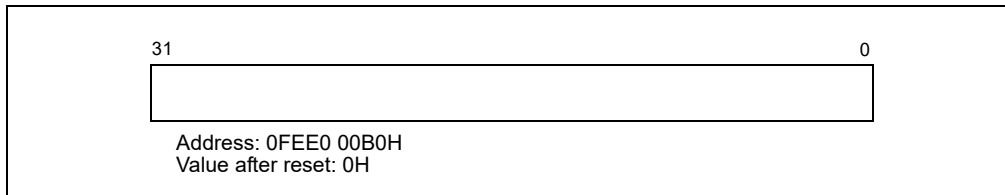
For the P6 family and Pentium processors, the IRR and ISR registers can queue no more than two interrupts per interrupt vector and will reject other interrupts that are received within the same vector.

If the local APIC receives an interrupt with an interrupt-priority class higher than that of the interrupt currently in service, and interrupts are enabled in the processor core, the local APIC dispatches the higher priority interrupt to the processor immediately (without waiting for a write to the EOI register). The currently executing interrupt handler is then interrupted so the higher-priority interrupt can be handled. When the handling of the higher-priority interrupt has been completed, the servicing of the interrupted interrupt is resumed.

The trigger mode register (TMR) indicates the trigger mode of the interrupt (see Figure 10-20). Upon acceptance of an interrupt into the IRR, the corresponding TMR bit is cleared for edge-triggered interrupts and set for level-triggered interrupts. If a TMR bit is set when an EOI cycle for its corresponding interrupt vector is generated, an EOI message is sent to all I/O APICs.

### 10.8.5 Signaling Interrupt Servicing Completion

For all interrupts except those delivered with the NMI, SMI, INIT, ExtINT, the start-up, or INIT-Deassert delivery mode, the interrupt handler must include a write to the end-of-interrupt (EOI) register (see Figure 10-21). This write must occur at the end of the handler routine, sometime before the IRET instruction. This action indicates that the servicing of the current interrupt is complete and the local APIC can issue the next interrupt from the ISR.



**Figure 10-21. EOI Register**

Upon receiving an EOI, the APIC clears the highest priority bit in the ISR and dispatches the next highest priority interrupt to the processor. If the terminated interrupt was a level-triggered interrupt, the local APIC also sends an end-of-interrupt message to all I/O APICs.

System software may prefer to direct EOIs to specific I/O APICs rather than having the local APIC send end-of-interrupt messages to all I/O APICs.

Software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register (see Section 10.9). If this bit is set, a broadcast EOI is not generated on an EOI cycle even if the associated TMR bit indicates that the current interrupt was level-triggered. The default value for the bit is 0, indicating that EOI broadcasts are performed.

Bit 12 of the Spurious Interrupt Vector Register is reserved to 0 if the processor does not support suppression of EOI broadcasts. Support for EOI-broadcast suppression is reported in bit 24 in the Local APIC Version Register (see Section 10.4.8); the feature is supported if that bit is set to 1. When supported, the feature is available in both xAPIC mode and x2APIC mode.

System software desiring to perform directed EOIs for level-triggered interrupts should set bit 12 of the Spurious Interrupt Vector Register and follow each the EOI to the local xAPIC for a level triggered interrupt with a directed EOI to the I/O APIC generating the interrupt (this is done by writing to the I/O APIC's EOI register). System software performing directed EOIs must retain a mapping associating level-triggered interrupts with the I/O APICs in the system.

### 10.8.6 Task Priority in IA-32e Mode

In IA-32e mode, operating systems can manage the 16 interrupt-priority classes (see Section 10.8.3, "Interrupt, Task, and Processor Priority") explicitly using the task priority register (TPR). Operating systems can use the TPR to temporarily block specific (low-priority) interrupts from interrupting a high-priority task. This is done by loading TPR with a value in which the task-priority class corresponds to the highest interrupt-priority class that is to be blocked. For example:

- Loading the TPR with a task-priority class of 8 (01000B) blocks all interrupts with an interrupt-priority class of 8 or less while allowing all interrupts with an interrupt-priority class of 9 or more to be recognized.
- Loading the TPR with a task-priority class of 0 enables all external interrupts.
- Loading the TPR with a task-priority class of 0FH (01111B) disables all external interrupts.

The TPR (shown in Figure 10-18) is cleared to 0 on reset. In 64-bit mode, software can read and write the TPR using an alternate interface, MOV CR8 instruction. The new task-priority class is established when the MOV CR8

instruction completes execution. Software does not need to force serialization after loading the TPR using MOV CR8.

Use of the MOV CRn instruction requires a privilege level of 0. Programs running at privilege level greater than 0 cannot read or write the TPR. An attempt to do so causes a general-protection exception. The TPR is abstracted from the interrupt controller (IC), which prioritizes and manages external interrupt delivery to the processor. The IC can be an external device, such as an APIC or 8259. Typically, the IC provides a priority mechanism similar or identical to the TPR. The IC, however, is considered implementation-dependent with the under-lying priority mechanisms subject to change. CR8, by contrast, is part of the Intel 64 architecture. Software can depend on this definition remaining unchanged.

Figure 10-22 shows the layout of CR8; only the low four bits are used. The remaining 60 bits are reserved and must be written with zeros. Failure to do this causes a general-protection exception.

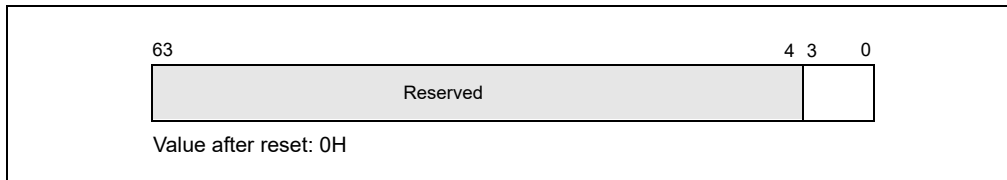


Figure 10-22. CR8 Register

### 10.8.6.1 Interaction of Task Priorities between CR8 and APIC

The first implementation of Intel 64 architecture includes a local advanced programmable interrupt controller (APIC) that is similar to the APIC used with previous IA-32 processors. Some aspects of the local APIC affect the operation of the architecturally defined task priority register and the programming interface using CR8.

Notable CR8 and APIC interactions are:

- The processor powers up with the local APIC enabled.
- The APIC must be enabled for CR8 to function as the TPR. Writes to CR8 are reflected into the APIC Task Priority Register.
- $APIC.TPR[\text{bits } 7:4] = CR8[\text{bits } 3:0]$ ,  $APIC.TPR[\text{bits } 3:0] = 0$ . A read of CR8 returns a 64-bit value which is the value of  $TPR[\text{bits } 7:4]$ , zero extended to 64 bits.

There are no ordering mechanisms between direct updates of the APIC.TPR and CR8. Operating software should implement either direct APIC TPR updates or CR8 style TPR updates but not mix them. Software can use a serializing instruction (for example, CPUID) to serialize updates between MOV CR8 and stores to the APIC.

## 10.9 SPURIOUS INTERRUPT

A special situation may occur when a processor raises its task priority to be greater than or equal to the level of the interrupt for which the processor INTR signal is currently being asserted. If at the time the INTA cycle is issued, the interrupt that was to be dispensed has become masked (programmed by software), the local APIC will deliver a spurious-interrupt vector. Dispensing the spurious-interrupt vector does not affect the ISR, so the handler for this vector should return without an EOI.

The vector number for the spurious-interrupt vector is specified in the spurious-interrupt vector register (see Figure 10-23). The functions of the fields in this register are as follows:

- Spurious Vector** Determines the vector number to be delivered to the processor when the local APIC generates a spurious vector.
- (Pentium 4 and Intel Xeon processors.) Bits 0 through 7 of the this field are programmable by software.
  - (P6 family and Pentium processors). Bits 4 through 7 of the this field are programmable by software, and bits 0 through 3 are hardwired to logical ones. Software writes to bits 0 through 3 have no effect.

### APIC Software Enable/Disable

Allows software to temporarily enable (1) or disable (0) the local APIC (see Section 10.4.3, "Enabling or Disabling the Local APIC").

**Focus Processor Checking**

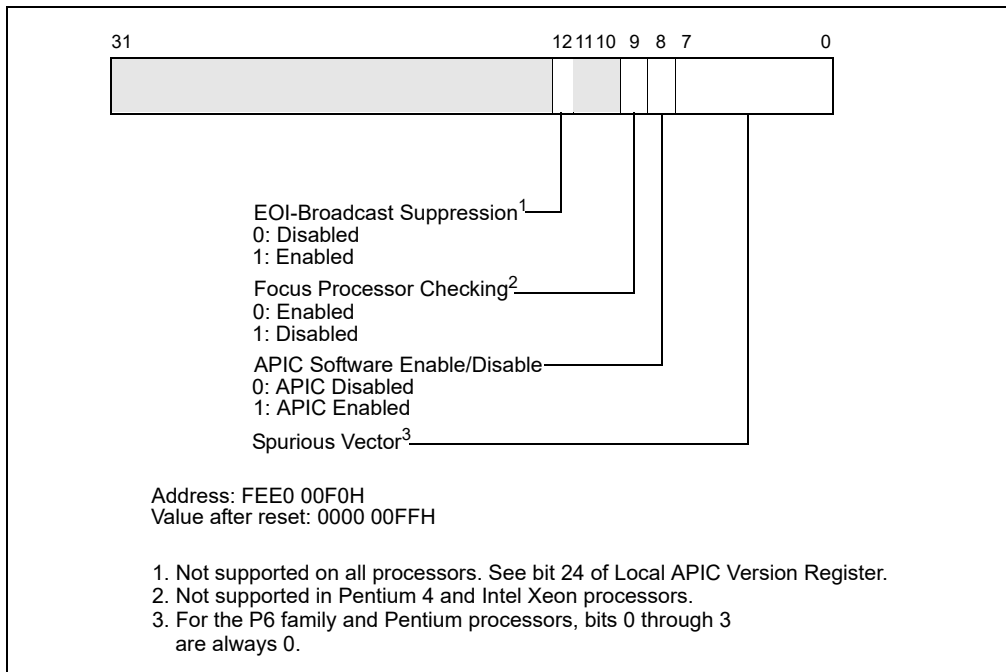
Determines if focus processor checking is enabled (0) or disabled (1) when using the lowest-priority delivery mode. In Pentium 4 and Intel Xeon processors, this bit is reserved and should be cleared to 0.

**Suppress EOI Broadcasts**

Determines whether an EOI for a level-triggered interrupt causes EOI messages to be broadcast to the I/O APICs (0) or not (1). See Section 10.8.5. The default value for this bit is 0, indicating that EOI broadcasts are performed. This bit is reserved to 0 if the processor does not support EOI-broadcast suppression.

**NOTE**

Do not program an LVT or IOAPIC RTE with a spurious vector even if you set the mask bit. A spurious vector ISR does not do an EOI. If for some reason an interrupt is generated by an LVT or RTE entry, the bit in the in-service register will be left set for the spurious vector. This will mask all interrupts at the same or lower priority



**Figure 10-23. Spurious-Interrupt Vector Register (SVR)**

**10.10 APIC BUS MESSAGE PASSING MECHANISM AND PROTOCOL (P6 FAMILY, PENTIUM PROCESSORS)**

The Pentium 4 and Intel Xeon processors pass messages among the local and I/O APICs on the system bus, using the system bus message passing mechanism and protocol.

The P6 family and Pentium processors, pass messages among the local and I/O APICs on the serial APIC bus, as follows. Because only one message can be sent at a time on the APIC bus, the I/O APIC and local APICs employ a "rotating priority" arbitration protocol to gain permission to send a message on the APIC bus. One or more APICs may start sending their messages simultaneously. At the beginning of every message, each APIC presents the type of the message it is sending and its current arbitration priority on the APIC bus. This information is used for arbitration. After each arbitration cycle (within an arbitration round), only the potential winners keep driving the bus.

By the time all arbitration cycles are completed, there will be only one APIC left driving the bus. Once a winner is selected, it is granted exclusive use of the bus, and will continue driving the bus to send its actual message.

After each successfully transmitted message, all APICs increase their arbitration priority by 1. The previous winner (that is, the one that has just successfully transmitted its message) assumes a priority of 0 (lowest). An agent whose arbitration priority was 15 (highest) during arbitration, but did not send a message, adopts the previous winner's arbitration priority, incremented by 1.

Note that the arbitration protocol described above is slightly different if one of the APICs issues a special End-Of-Interrupt (EOI). This high-priority message is granted the bus regardless of its sender's arbitration priority, unless more than one APIC issues an EOI message simultaneously. In the latter case, the APICs sending the EOI messages arbitrate using their arbitration priorities.

If the APICs are set up to use "lowest priority" arbitration (see Section 10.6.2.4, "Lowest Priority Delivery Mode") and multiple APICs are currently executing at the lowest priority (the value in the APR register), the arbitration priorities (unique values in the Arb ID register) are used to break ties. All 8 bits of the APR are used for the lowest priority arbitration.

### 10.10.1 Bus Message Formats

See Section 10.13, "APIC Bus Message Formats," for a description of bus message formats used to transmit messages on the serial APIC bus.

## 10.11 MESSAGE SIGNALLED INTERRUPTS

The *PCI Local Bus Specification, Rev 2.2* ([www.pcisig.com](http://www.pcisig.com)) introduces the concept of message signalled interrupts. As the specification indicates:

"Message signalled interrupts (MSI) is an optional feature that enables PCI devices to request service by writing a system-specified message to a system-specified address (PCI DWORD memory write transaction). The transaction address specifies the message destination while the transaction data specifies the message. System software is expected to initialize the message destination and message during device configuration, allocating one or more non-shared messages to each MSI capable function."

The capabilities mechanism provided by the *PCI Local Bus Specification* is used to identify and configure MSI capable PCI devices. Among other fields, this structure contains a Message Data Register and a Message Address Register. To request service, the PCI device function writes the contents of the Message Data Register to the address contained in the Message Address Register (and the Message Upper Address register for 64-bit message addresses).

Section 10.11.1 and Section 10.11.2 provide layout details for the Message Address Register and the Message Data Register. The operation issued by the device is a PCI write command to the Message Address Register with the Message Data Register contents. The operation follows semantic rules as defined for PCI write operations and is a DWORD operation.

### 10.11.1 Message Address Register Format

The format of the Message Address Register (lower 32-bits) is shown in Figure 10-24.

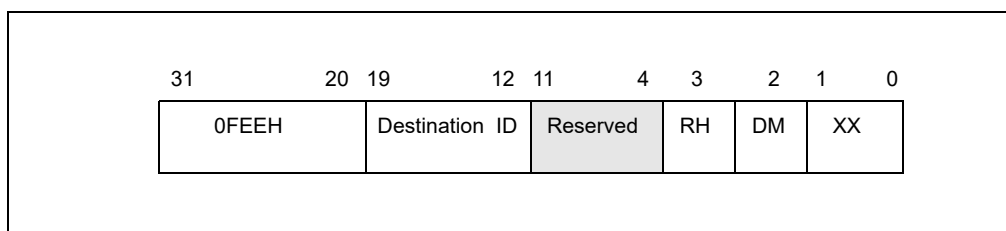


Figure 10-24. Layout of the MSI Message Address Register

Fields in the Message Address Register are as follows:

1. **Bits 31-20** — These bits contain a fixed value for interrupt messages (0FEEH). This value locates interrupts at the 1-MByte area with a base address of 4G – 18M. All accesses to this region are directed as interrupt messages. Care must be taken to ensure that no other device claims the region as I/O space.
2. **Destination ID** — This field contains an 8-bit destination ID. It identifies the message's target processor(s). The destination ID corresponds to bits 63:56 of the I/O APIC Redirection Table Entry if the IOAPIC is used to dispatch the interrupt to the processor(s).
3. **Redirection hint indication (RH)** — When this bit is set, the message is directed to the processor with the lowest interrupt priority among processors that can receive the interrupt.
  - When RH is 0, the interrupt is directed to the processor listed in the Destination ID field.
  - When RH is 1 and the physical destination mode is used, the Destination ID field must not be set to FFH; it must point to a processor that is present and enabled to receive the interrupt.
  - When RH is 1 and the logical destination mode is active in a system using a flat addressing model, the Destination ID field must be set so that bits set to 1 identify processors that are present and enabled to receive the interrupt.
  - If RH is set to 1 and the logical destination mode is active in a system using cluster addressing model, then Destination ID field must not be set to FFH; the processors identified with this field must be present and enabled to receive the interrupt.
4. **Destination mode (DM)** — This bit indicates whether the Destination ID field should be interpreted as logical or physical APIC ID for delivery of the lowest priority interrupt.
  - If RH is 1 and DM is 0, the Destination ID field is in physical destination mode and only the processor in the system that has the matching APIC ID is considered for delivery of that interrupt (this means no redirection).
  - If RH is 1 and DM is 1, the Destination ID Field is interpreted as in logical destination mode and the redirection is limited to only those processors that are part of the logical group of processors based on the processor's logical APIC ID and the Destination ID field in the message. The logical group of processors consists of those identified by matching the 8-bit Destination ID with the logical destination identified by the Destination Format Register and the Logical Destination Register in each local APIC. The details are similar to those described in Section 10.6.2, "Determining IPI Destination."
  - If RH is 0, then the DM bit is ignored and the message is sent ahead independent of whether the physical or logical destination mode is used.

### 10.11.2 Message Data Register Format

The layout of the Message Data Register is shown in Figure 10-25.

Reserved fields are not assumed to be any value. Software must preserve their contents on writes. Other fields in the Message Data Register are described below.

1. **Vector** — This 8-bit field contains the interrupt vector associated with the message. Values range from 010H to 0FEH. Software must guarantee that the field is not programmed with vector 00H to 0FH.
2. **Delivery Mode** — This 3-bit field specifies how the interrupt receipt is handled. Delivery Modes operate only in conjunction with specified Trigger Modes. Correct Trigger Modes must be guaranteed by software. Restrictions are indicated below:
  - a. **000B (Fixed Mode)** — Deliver the signal to all the agents listed in the destination. The Trigger Mode for fixed delivery mode can be edge or level.
  - b. **001B (Lowest Priority)** — Deliver the signal to the agent that is executing at the lowest priority of all agents listed in the destination field. The trigger mode can be edge or level.
  - c. **010B (System Management Interrupt or SMI)** — The delivery mode is edge only. For systems that rely on SMI semantics, the vector field is ignored but must be programmed to all zeroes for future compatibility.

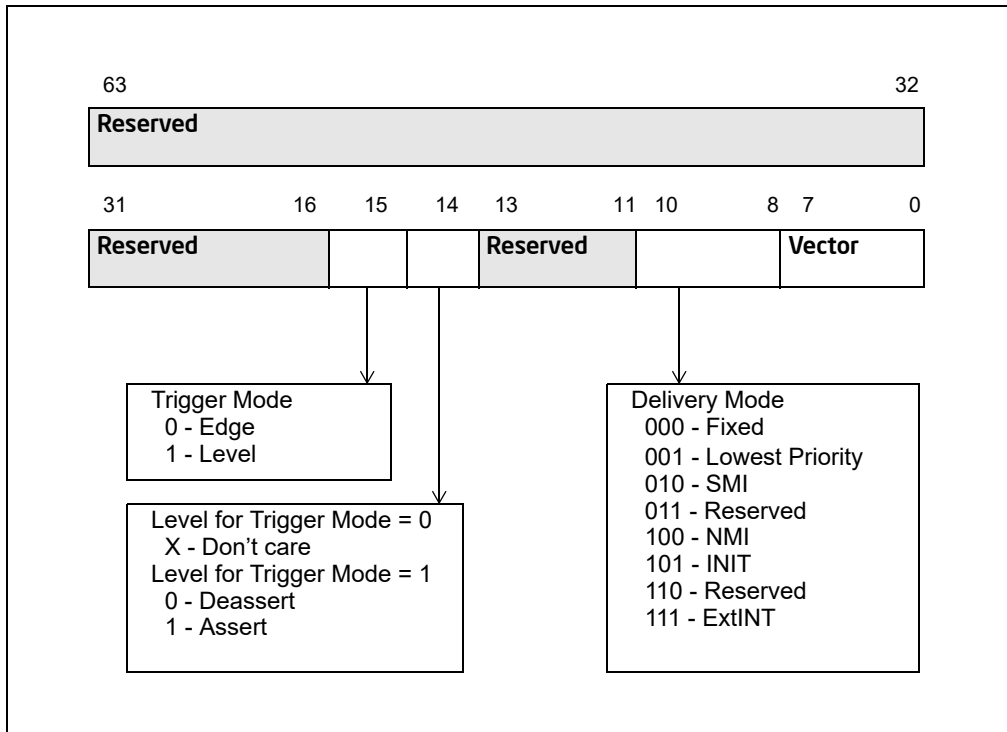


Figure 10-25. Layout of the MSI Message Data Register

- d. **100B (NMI)** — Deliver the signal to all the agents listed in the destination field. The vector information is ignored. NMI is an edge triggered interrupt regardless of the Trigger Mode Setting.
  - e. **101B (INIT)** — Deliver this signal to all the agents listed in the destination field. The vector information is ignored. INIT is an edge triggered interrupt regardless of the Trigger Mode Setting.
  - f. **111B (ExtINT)** — Deliver the signal to the INTR signal of all agents in the destination field (as an interrupt that originated from an 8259A compatible interrupt controller). The vector is supplied by the INTA cycle issued by the activation of the ExtINT. ExtINT is an edge triggered interrupt.
3. **Level** — Edge triggered interrupt messages are always interpreted as assert messages. For edge triggered interrupts this field is not used. For level triggered interrupts, this bit reflects the state of the interrupt input.
  4. **Trigger Mode** — This field indicates the signal type that will trigger a message.
    - a. **0** — Indicates edge sensitive.
    - b. **1** — Indicates level sensitive.

## 10.12 EXTENDED XAPIC (X2APIC)

The x2APIC architecture extends the xAPIC architecture (described in Section 10.4) in a backward compatible manner and provides forward extendability for future Intel platform innovations. Specifically, the x2APIC architecture does the following.

- Retains all key elements of compatibility to the xAPIC architecture.
  - Delivery modes.
  - Interrupt and processor priorities.
  - Interrupt sources.
  - Interrupt destination types.
- Provides extensions to scale processor addressability for both the logical and physical destination modes.



- Adds new features to enhance performance of interrupt delivery.
- Reduces complexity of logical destination mode interrupt delivery on link based platform architectures.
- Uses MSR programming interface to access APIC registers in x2APIC mode instead of memory-mapped interfaces. Memory-mapped interface is supported when operating in xAPIC mode.

### 10.12.1 Detecting and Enabling x2APIC Mode

Processor support for x2APIC mode can be detected by executing CPUID with EAX=1 and then checking ECX, bit 21 ECX. If CPUID.(EAX=1):ECX.21 is set, the processor supports the x2APIC capability and can be placed into the x2APIC mode.

System software can place the local APIC in the x2APIC mode by setting the x2APIC mode enable bit (bit 10) in the IA32\_APIC\_BASE MSR at MSR address 01BH. The layout for the IA32\_APIC\_BASE MSR is shown in Figure 10-26.

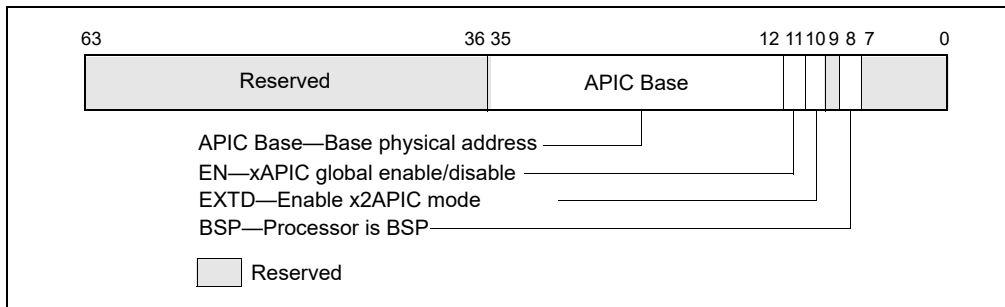


Figure 10-26. IA32\_APIC\_BASE MSR Supporting x2APIC

Table 10-5, “x2APIC operating mode configurations” describe the possible combinations of the enable bit (EN - bit 11) and the extended mode bit (EXTD - bit 10) in the IA32\_APIC\_BASE MSR.

Table 10-5. x2APIC Operating Mode Configurations

xAPIC global enable (IA32_APIC_BASE[11])	x2APIC enable (IA32_APIC_BASE[10])	Description
0	0	local APIC is disabled
0	1	Invalid
1	0	local APIC is enabled in xAPIC mode
1	1	local APIC is enabled in x2APIC mode

Once the local APIC has been switched to x2APIC mode (EN = 1, EXTD = 1), switching back to xAPIC mode would require system software to disable the local APIC unit. Specifically, attempting to write a value to the IA32\_APIC\_BASE MSR that has (EN= 1, EXTD = 0) when the local APIC is enabled and in x2APIC mode causes a general-protection exception. Once bit 10 in IA32\_APIC\_BASE MSR is set, the only way to leave x2APIC mode using IA32\_APIC\_BASE would require a WRMSR to set both bit 11 and bit 10 to zero. Section 10.12.5, “x2APIC State Transitions” provides a detailed state diagram for the state transitions allowed for the local APIC.

#### 10.12.1.1 Instructions to Access APIC Registers

In x2APIC mode, system software uses RDMSR and WRMSR to access the APIC registers. The MSR addresses for accessing the x2APIC registers are architecturally defined and specified in Section 10.12.1.2, “x2APIC Register Address Space”. Executing the RDMSR instruction with the APIC register address specified in ECX returns the content of bits 0 through 31 of the APIC registers in EAX. Bits 32 through 63 are returned in register EDX - these bits are reserved if the APIC register being read is a 32-bit register. Similarly executing the WRMSR instruction with the APIC register address in ECX, writes bits 0 to 31 of register EAX to bits 0 to 31 of the specified APIC register. If the register is a 64-bit register then bits 0 to 31 of register EDX are written to bits 32 to 63 of the APIC register. The



Interrupt Command Register is the only APIC register that is implemented as a 64-bit MSR. The semantics of handling reserved bits are defined in Section 10.12.1.3, "Reserved Bit Checking".

### 10.12.1.2 x2APIC Register Address Space

The MSR address range 800H through BFFH is architecturally reserved and dedicated for accessing APIC registers in x2APIC mode. Table 10-6 lists the APIC registers that are available in x2APIC mode. When appropriate, the table also gives the offset at which each register is available on the page referenced by IA32\_APIC\_BASE[35:12] in xAPIC mode.

There is a one-to-one mapping between the x2APIC MSRs and the legacy xAPIC register offsets with the following exceptions:

- The Destination Format Register (DFR): The DFR, supported at offset 0E0H in xAPIC mode, is not supported in x2APIC mode. There is no MSR with address 80EH.
- The Interrupt Command Register (ICR): The two 32-bit registers in xAPIC mode (at offsets 300H and 310H) are merged into a single 64-bit MSR in x2APIC mode (with MSR address 830H). There is no MSR with address 831H.
- The SELF IPI register. This register is available only in x2APIC mode at address 83FH. In xAPIC mode, there is no register defined at offset 3F0H.

MSR addresses in the range 800H–BFFH that are not listed in Table 10-6 (including 80EH and 831H) are reserved. Executions of RDMSR and WRMSR that attempt to access such addresses cause general-protection exceptions.

The MSR address space is compressed to allow for future growth. Every 32 bit register on a 128-bit boundary in the legacy MMIO space is mapped to a single MSR in the local x2APIC MSR address space. The upper 32-bits of all x2APIC MSRs (except for the ICR) are reserved.

**Table 10-6. Local APIC Register Address Map Supported by x2APIC**

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
802H	020H	Local APIC ID register	Read-only <sup>1</sup>	See Section 10.12.5.1 for initial values.
803H	030H	Local APIC Version register	Read-only	Same version used in xAPIC mode and x2APIC mode.
808H	080H	Task Priority Register (TPR)	Read/write	Bits 31:8 are reserved. <sup>2</sup>
80AH	0A0H	Processor Priority Register (PPR)	Read-only	
80BH	0B0H	EOI register	Write-only <sup>3</sup>	WRMSR of a non-zero value causes #GP(0).
80DH	0D0H	Logical Destination Register (LDR)	Read-only	Read/write in xAPIC mode.
80FH	0F0H	Spurious Interrupt Vector Register (SVR)	Read/write	See Section 10.9 for reserved bits.
810H	100H	In-Service Register (ISR); bits 31:0	Read-only	
811H	110H	ISR bits 63:32	Read-only	
812H	120H	ISR bits 95:64	Read-only	
813H	130H	ISR bits 127:96	Read-only	
814H	140H	ISR bits 159:128	Read-only	
815H	150H	ISR bits 191:160	Read-only	
816H	160H	ISR bits 223:192	Read-only	

**Table 10-6. Local APIC Register Address Map Supported by x2APIC (Contd.)**

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
817H	170H	ISR bits 255:224	Read-only	
818H	180H	Trigger Mode Register (TMR); bits 31:0	Read-only	
819H	190H	TMR bits 63:32	Read-only	
81AH	1A0H	TMR bits 95:64	Read-only	
81BH	1B0H	TMR bits 127:96	Read-only	
81CH	1C0H	TMR bits 159:128	Read-only	
81DH	1D0H	TMR bits 191:160	Read-only	
81EH	1E0H	TMR bits 223:192	Read-only	
81FH	1F0H	TMR bits 255:224	Read-only	
820H	200H	Interrupt Request Register (IRR); bits 31:0	Read-only	
821H	210H	IRR bits 63:32	Read-only	
822H	220H	IRR bits 95:64	Read-only	
823H	230H	IRR bits 127:96	Read-only	
824H	240H	IRR bits 159:128	Read-only	
825H	250H	IRR bits 191:160	Read-only	
826H	260H	IRR bits 223:192	Read-only	
827H	270H	IRR bits 255:224	Read-only	
828H	280H	Error Status Register (ESR)	Read/write	WRMSR of a non-zero value causes #GP(0). See Section 10.5.3.
82FH	2F0H	LVT CMCI register	Read/write	See Figure 10-8 for reserved bits.
830H <sup>4</sup>	300H and 310H	Interrupt Command Register (ICR)	Read/write	See Figure 10-28 for reserved bits
832H	320H	LVT Timer register	Read/write	See Figure 10-8 for reserved bits.
833H	330H	LVT Thermal Sensor register	Read/write	See Figure 10-8 for reserved bits.
834H	340H	LVT Performance Monitoring register	Read/write	See Figure 10-8 for reserved bits.
835H	350H	LVT LINT0 register	Read/write	See Figure 10-8 for reserved bits.
836H	360H	LVT LINT1 register	Read/write	See Figure 10-8 for reserved bits.
837H	370H	LVT Error register	Read/write	See Figure 10-8 for reserved bits.
838H	380H	Initial Count register (for Timer)	Read/write	
839H	390H	Current Count register (for Timer)	Read-only	
83EH	3E0H	Divide Configuration Register (DCR; for Timer)	Read/write	See Figure 10-10 for reserved bits.
83FH	Not available	SELF IPI <sup>5</sup>	Write-only	Available only in x2APIC mode.

**NOTES:**

1. WRMSR causes #GP(0) for read-only registers.

2. WRMSR causes #GP(0) for attempts to set a reserved bit to 1 in a read/write register (including bits 63:32 of each register).
3. RDMSR causes #GP(0) for write-only registers.
4. MSR 831H is reserved; read/write operations cause general-protection exceptions. The contents of the APIC register at MMIO offset 310H are accessible in x2APIC mode through the MSR at address 830H.
5. SELF IPI register is supported only in x2APIC mode.

### 10.12.1.3 Reserved Bit Checking

Section 10.12.1.2 and Table 10-6 specifies the reserved bit definitions for the APIC registers in x2APIC mode. Non-zero writes (by WRMSR instruction) to reserved bits to these registers will raise a general protection fault exception while reads return zeros (RsvdZ semantics).

In x2APIC mode, the local APIC ID register is increased to 32 bits wide. This enables  $2^{32}-1$  processors to be addressable in physical destination mode. This 32-bit value is referred to as “x2APIC ID”. A processor implementation may choose to support less than 32 bits in its hardware. System software should be agnostic to the actual number of bits that are implemented. All non-implemented bits will return zeros on reads by software.

The APIC ID value of FFFF\_FFFFH and the highest value corresponding to the implemented bit-width of the local APIC ID register in the system are reserved and cannot be assigned to any logical processor.

In x2APIC mode, the local APIC ID register is a read-only register to system software and will be initialized by hardware. It is accessed via the RDMSR instruction reading the MSR at address 0802H.

Each logical processor in the system (including clusters with a communication fabric) must be configured with a unique x2APIC ID to avoid collisions of x2APIC IDs. On DP and high-end MP processors targeted to specific market segments and depending on the system configuration, it is possible that logical processors in different and “un-connected” clusters power up initialized with overlapping x2APIC IDs. In these configurations, a model-specific means may be provided in those product segments to enable BIOS and/or platform firmware to re-configure the x2APIC IDs in some clusters to provide for unique and non-overlapping system wide IDs before configuring the disconnected components into a single system.

### 10.12.2 x2APIC Register Availability

The local APIC registers can be accessed via the MSR interface only when the local APIC has been switched to the x2APIC mode as described in Section 10.12.1. Accessing any APIC register in the MSR address range 0800H through 0BFFH via RDMSR or WRMSR when the local APIC is not in x2APIC mode causes a general-protection exception. In x2APIC mode, the memory mapped interface is not available and any access to the MMIO interface will behave similar to that of a legacy xAPIC in globally disabled state. Table 10-7 provides the interactions between the legacy & extended modes and the legacy and register interfaces.

**Table 10-7. MSR/MMIO Interface of a Local x2APIC in Different Modes of Operation**

	MMIO Interface	MSR Interface
xAPIC mode	Available	General-protection exception
x2APIC mode	Behavior identical to xAPIC in globally disabled state	Available

### 10.12.3 MSR Access in x2APIC Mode

To allow for efficient access to the APIC registers in x2APIC mode, the serializing semantics of WRMSR are relaxed when writing to the APIC registers. Thus, system software should not use “WRMSR to APIC registers in x2APIC mode” as a serializing instruction. Read and write accesses to the APIC registers will occur in program order. A WRMSR to an APIC register may complete before all preceding stores are globally visible; software can prevent this by inserting a serializing instruction, an SFENCE, or an MFENCE before the WRMSR.

The RDMSR instruction is not serializing and this behavior is unchanged when reading APIC registers in x2APIC mode. System software accessing the APIC registers using the RDMSR instruction should not expect a serializing behavior. (Note: The MMIO-based xAPIC interface is mapped by system software as an un-cached region. Consequently, read/writes to the xAPIC-MMIO interface have serializing semantics in the xAPIC mode.)

## 10.12.4 VM-Exit Controls for MSRs and x2APIC Registers

The VMX architecture allows a VMM to specify lists of MSRs to be loaded or stored on VMX transitions using the VMX-transition MSR areas (see VM-exit MSR-store address field, VM-exit MSR-load address field, and VM-entry MSR-load address field in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).

The X2APIC MSRs cannot to be loaded and stored on VMX transitions. A VMX transition fails if the VMM has specified that the transition should access any MSRs in the address range from 0000\_0800H to 0000\_08FFH (the range used for accessing the X2APIC registers). Specifically, processing of a 128-bit entry in any of the VMX-transition MSR areas fails if bits 31:0 of that entry (represented as ENTRY\_LOW\_DW) satisfies the expression: "ENTRY\_LOW\_DW & FFFF800H = 0000800H". Such a failure causes an associated VM entry to fail (by reloading host state) and causes an associated VM exit to lead to VMX abort.

## 10.12.5 x2APIC State Transitions

This section provides a detailed description of the x2APIC states of a local x2APIC unit, transitions between these states as well as interactions of these states with INIT and reset.

### 10.12.5.1 x2APIC States

The valid states for a local x2APIC unit are listed in Table 10-5.

- APIC disabled: IA32\_APIC\_BASE[EN]=0 and IA32\_APIC\_BASE[EXTD]=0.
- xAPIC mode: IA32\_APIC\_BASE[EN]=1 and IA32\_APIC\_BASE[EXTD]=0.
- x2APIC mode: IA32\_APIC\_BASE[EN]=1 and IA32\_APIC\_BASE[EXTD]=1.
- Invalid: IA32\_APIC\_BASE[EN]=0 and IA32\_APIC\_BASE[EXTD]=1.

The state corresponding to EXTD=1 and EN=0 is not valid and it is not possible to get into this state. An execution of WRMSR to the IA32\_APIC\_BASE\_MSR that attempts a transition from a valid state to this invalid state causes a general-protection exception. Figure 10-27 shows the comprehensive state transition diagram for a local x2APIC unit.

On coming out of reset, the local APIC unit is enabled and is in the xAPIC mode: IA32\_APIC\_BASE[EN]=1 and IA32\_APIC\_BASE[EXTD]=0. The APIC registers are initialized as follows.

- The local APIC ID is initialized by hardware with a 32 bit ID (x2APIC ID). The lowest 8 bits of the x2APIC ID are the legacy local xAPIC ID, and are stored in the upper 8 bits of the APIC register for access in xAPIC mode.
- The following APIC registers are reset to all zeros for those fields that are defined in the xAPIC mode.
  - IRR, ISR, TMR, ICR, LDR, TPR, Divide Configuration Register (See Section 10.4 through Section 10.6 for details of individual APIC registers).
  - Timer initial count and timer current count registers.
- The LVT registers are reset to 0s except for the mask bits; these are set to 1s.
- The local APIC version register is not affected.
- The Spurious Interrupt Vector Register is initialized to 000000FFH.
- The DFR (available only in xAPIC mode) is reset to all 1s.
- SELF IPI register is reset to zero.

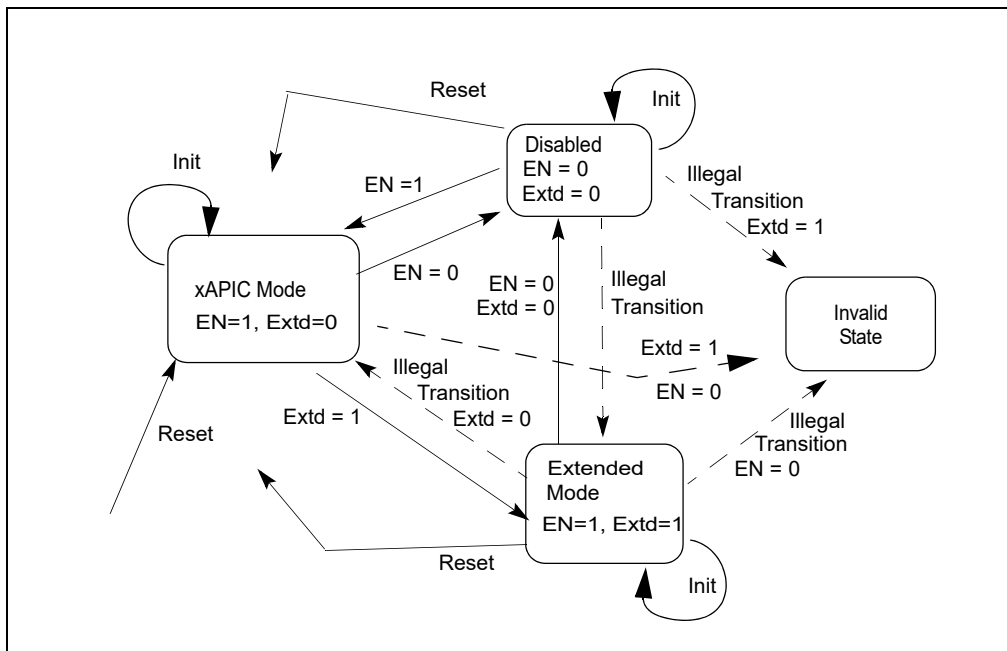


Figure 10-27. Local x2APIC State Transitions with IA32\_APIC\_BASE, INIT, and Reset

### x2APIC After Reset

The valid transitions from the xAPIC mode state are:

- to the x2APIC mode by setting EXT to 1 (resulting EN=1, EXTD= 1). The physical x2APIC ID (see Figure 10-6) is preserved across this transition and the logical x2APIC ID (see Figure 10-29) is initialized by hardware during this transition as documented in Section 10.12.10.2. The state of the extended fields in other APIC registers, which was not initialized at reset, is not architecturally defined across this transition and system software should explicitly initialize those programmable APIC registers.
- to the disabled state by setting EN to 0 (resulting EN=0, EXTD= 0).

The result of an INIT in the xAPIC state places the APIC in the state with EN= 1, EXTD= 0. The state of the local APIC ID register is preserved (the 8-bit xAPIC ID is in the upper 8 bits of the APIC ID register). All the other APIC registers are initialized as a result of INIT.

A reset in this state places the APIC in the state with EN= 1, EXTD= 0. The state of the local APIC ID register is initialized as described in Section 10.12.5.1. All the other APIC registers are initialized described in Section 10.12.5.1.

### x2APIC Transitions From x2APIC Mode

From the x2APIC mode, the only valid x2APIC transition using IA32\_APIC\_BASE is to the state where the x2APIC is disabled by setting EN to 0 and EXTD to 0. The x2APIC ID (32 bits) and the legacy local xAPIC ID (8 bits) are preserved across this transition. A transition from the x2APIC mode to xAPIC mode is not valid, and the corresponding WRMSR to the IA32\_APIC\_BASE MSR causes a general-protection exception.

A reset in this state places the x2APIC in xAPIC mode. All APIC registers (including the local APIC ID register) are initialized as described in Section 10.12.5.1.

An INIT in this state keeps the x2APIC in the x2APIC mode. The state of the local APIC ID register is preserved (all 32 bits). However, all the other APIC registers are initialized as a result of the INIT transition.

## x2APIC Transitions From Disabled Mode

From the disabled state, the only valid x2APIC transition using IA32\_APIC\_BASE is to the xAPIC mode (EN= 1, EXTD = 0). Thus the only means to transition from x2APIC mode to xAPIC mode is a two-step process:

- first transition from x2APIC mode to local APIC disabled mode (EN= 0, EXTD = 0),
- followed by another transition from disabled mode to xAPIC mode (EN= 1, EXTD= 0).

Consequently, all the APIC register states in the x2APIC, except for the x2APIC ID (32 bits), are not preserved across mode transitions.

A reset in the disabled state places the x2APIC in the xAPIC mode. All APIC registers (including the local APIC ID register) are initialized as described in Section 10.12.5.1.

An INIT in the disabled state keeps the x2APIC in the disabled state.

## State Changes From xAPIC Mode to x2APIC Mode

After APIC register states have been initialized by software in xAPIC mode, a transition from xAPIC mode to x2APIC mode does not affect most of the APIC register states, except the following:

- The Logical Destination Register is not preserved.
- Any APIC ID value written to the memory-mapped local APIC ID register is not preserved.
- The high half of the Interrupt Command Register is not preserved.

### 10.12.6 Routing of Device Interrupts in x2APIC Mode

The x2APIC architecture is intended to work with all existing IOxAPIC units as well as all PCI and PCI Express (PCIe) devices that support the capability for message-signaled interrupts (MSI). Support for x2APIC modifies only the following:

- the local APIC units;
- the interconnects joining IOxAPIC units to the local APIC units; and
- the interconnects joining MSI-capable PCI and PCIe devices to the local APIC units.

No modifications are required to MSI-capable PCI and PCIe devices. Similarly, no modifications are required to IOxAPIC units. This made possible through use of the interrupt-remapping architecture specified in the *Intel® Virtualization Technology for Directed I/O*, Revision 1.3 for the routing of interrupts from MSI-capable devices to local APIC units operating in x2APIC mode.

### 10.12.7 Initialization by System Software

Routing of device interrupts to local APIC units operating in x2APIC mode requires use of the interrupt-remapping architecture specified in the *Intel® Virtualization Technology for Directed I/O* (Revision 1.3 and/or later versions). Because of this, BIOS must enumerate support for and software must enable this interrupt remapping with Extended Interrupt Mode Enabled before it enabling x2APIC mode in the local APIC units.

The ACPI interfaces for the x2APIC are described in Section 5.2, "ACPI System Description Tables," of the *Advanced Configuration and Power Interface Specification*, Revision 4.0a (<http://www.acpi.info/spec.htm>). The default behavior for BIOS is to pass the control to the operating system with the local x2APICs in xAPIC mode if all APIC IDs reported by CPUID.0BH:EDX are less than 255, and in x2APIC mode if there are any logical processor reporting an APIC ID of 255 or greater.

### 10.12.8 CPUID Extensions And Topology Enumeration

For Intel 64 and IA-32 processors that support x2APIC, a value of 1 reported by CPUID.01H:ECX[21] indicates that the processor supports x2APIC and the extended topology enumeration leaf (CPUID.0BH).

The extended topology enumeration leaf can be accessed by executing CPUID with EAX = 0BH. Processors that do not support x2APIC may support CPUID leaf 0BH. Software can detect the availability of the extended topology enumeration leaf (0BH) by performing two steps:

- Check maximum input value for basic CPUID information by executing CPUID with EAX= 0. If CPUID.0H:EAX is greater than or equal to 11 (0BH), then proceed to next step
- Check CPUID.EAX=0BH, ECX=0H:EBX is non-zero.

If both of the above conditions are true, extended topology enumeration leaf is available. If available, the extended topology enumeration leaf is the preferred mechanism for enumerating topology. The presence of CPUID leaf 0BH in a processor does not guarantee support for x2APIC. If CPUID.EAX=0BH, ECX=0H:EBX returns zero and maximum input value for basic CPUID information is greater than 0BH, then CPUID.0BH leaf is not supported on that processor.

The extended topology enumeration leaf is intended to assist software with enumerating processor topology on systems that requires 32-bit x2APIC IDs to address individual logical processors. Details of CPUID leaf 0BH can be found in the reference pages of CPUID in Chapter 3 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

Processor topology enumeration algorithm for processors supporting the extended topology enumeration leaf of CPUID and processors that do not support CPUID leaf 0BH are treated in Section 8.9.4, "Algorithm for Three-Level Mappings of APIC\_ID".

### 10.12.8.1 Consistency of APIC IDs and CPUID

The consistency of physical x2APIC ID in MSR 802H in x2APIC mode and the 32-bit value returned in CPUID.0BH:EDX is facilitated by processor hardware.

CPUID.0BH:EDX will report the full 32 bit ID, in xAPIC and x2APIC mode. This allows BIOS to determine if a system has processors with IDs exceeding the 8-bit initial APIC ID limit (CPUID.01H:EBX[31:24]). Initial APIC ID (CPUID.01H:EBX[31:24]) is always equal to CPUID.0BH:EDX[7:0].

If the values of CPUID.0BH:EDX reported by all logical processors in a system are less than 255, BIOS can transfer control to OS in xAPIC mode.

If the values of CPUID.0BH:EDX reported by some logical processors in a system are greater than or equal to 255, BIOS must support two options to hand off to OS.

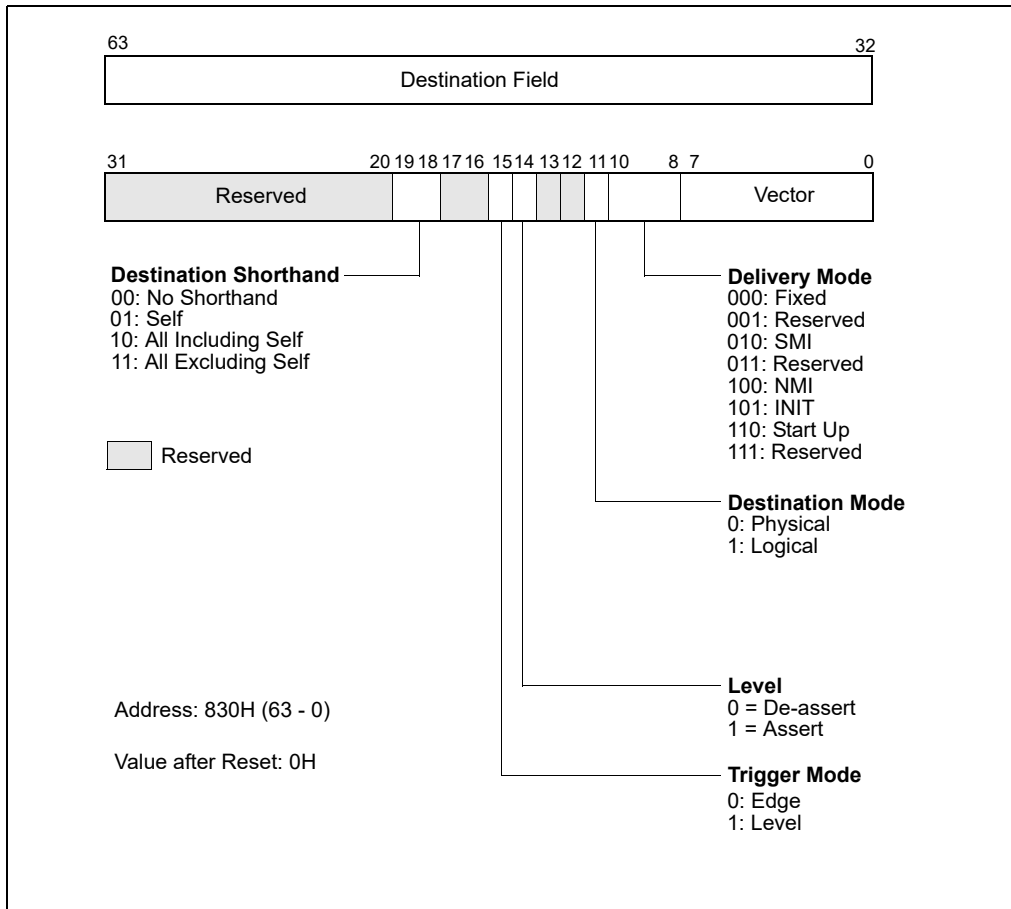
- If BIOS enables logical processors with x2APIC IDs greater than 255, then it should enable x2APIC in the Boot Strap Processor (BSP) and all Application Processors (AP) before passing control to the OS. Applications requiring processor topology information must use OS provided services based on x2APIC IDs or CPUID.0BH leaf.
- If a BIOS transfers control to OS in xAPIC mode, then the BIOS must ensure that only logical processors with CPUID.0BH:EDX value less than 255 are enabled. BIOS initialization on all logical processors with CPUID.0B:EDX values greater than or equal to 255 must (a) disable APIC and execute CLI in each logical processor, and (b) leave these logical processor in the lowest power state so that these processors do not respond to INIT IPI during OS boot. The BSP and all the enabled logical processor operate in xAPIC mode after BIOS passed control to OS. Application requiring processor topology information can use OS provided legacy services based on 8-bit initial APIC IDs or legacy topology information from CPUID.01H and CPUID 04H leaves. Even if the BIOS passes control in xAPIC mode, an OS can switch the processors to x2APIC mode later. BIOS SMM handler should always read the APIC\_BASE\_MSR, determine the APIC mode and use the corresponding access method.

### 10.12.9 ICR Operation in x2APIC Mode

In x2APIC mode, the layout of the Interrupt Command Register is shown in Figure 10-12. The lower 32 bits of ICR in x2APIC mode is identical to the lower half of the ICR in xAPIC mode, except the Delivery Status bit is removed since it is not needed in x2APIC mode. The destination ID field is expanded to 32 bits in x2APIC mode.

To send an IPI using the ICR, software must set up the ICR to indicate the type of IPI message to be sent and the destination processor or processors. Self IPIs can also be sent using the SELF IPI register (see Section 10.12.11).





**Figure 10-28. Interrupt Command Register (ICR) in x2APIC Mode**

A single MSR write to the Interrupt Command Register is required for dispatching an interrupt in x2APIC mode. With the removal of the Delivery Status bit, system software no longer has a reason to read the ICR. It remains readable only to aid in debugging; however, software should not assume the value returned by reading the ICR is the last written value.

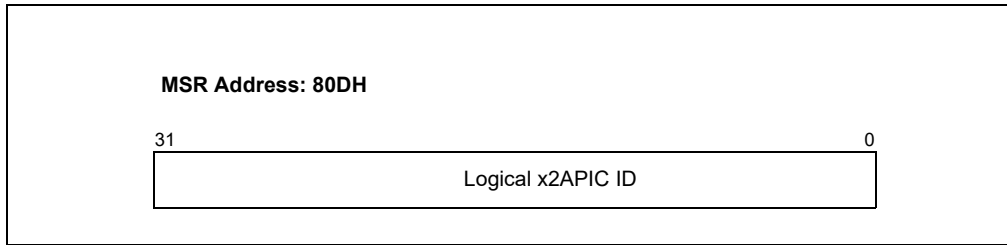
A destination ID value of FFFF\_FFFFH is used for broadcast of interrupts in both logical destination and physical destination modes.

## 10.12.10 Determining IPI Destination in x2APIC Mode

### 10.12.10.1 Logical Destination Mode in x2APIC Mode

In x2APIC mode, the Logical Destination Register (LDR) is increased to 32 bits wide. It is a read-only register to system software. This 32-bit value is referred to as "logical x2APIC ID". System software accesses this register via the RDMSR instruction reading the MSR at address 80DH. Figure 10-29 provides the layout of the Logical Destination Register in x2APIC mode.





**Figure 10-29. Logical Destination Register in x2APIC Mode**

In the xAPIC mode, the Destination Format Register (DFR) through the MMIO interface determines the choice of a flat logical mode or a clustered logical mode. Flat logical mode is not supported in the x2APIC mode. Hence the Destination Format Register (DFR) is eliminated in x2APIC mode.

The 32-bit logical x2APIC ID field of LDR is partitioned into two sub-fields:

- Cluster ID (LDR[31:16]): is the address of the destination cluster
- Logical ID (LDR[15:0]): defines a logical ID of the individual local x2APIC within the cluster specified by LDR[31:16].

This layout enables  $2^{16}-1$  clusters each with up to 16 unique logical IDs - effectively providing an addressability of  $((2^{20}) - 16)$  processors in logical destination mode.

It is likely that processor implementations may choose to support less than 16 bits of the cluster ID or less than 16-bits of the Logical ID in the Logical Destination Register. However system software should be agnostic to the number of bits implemented in the cluster ID and logical ID sub-fields. The x2APIC hardware initialization will ensure that the appropriately initialized logical x2APIC IDs are available to system software and reads of non-implemented bits return zero. This is a read-only register that software must read to determine the logical x2APIC ID of the processor. Specifically, software can apply a 16-bit mask to the lowest 16 bits of the logical x2APIC ID to identify the logical address of a processor within a cluster without needing to know the number of implemented bits in cluster ID and Logical ID sub-fields. Similarly, software can create a message destination address for cluster model, by bit-Oring the Logical X2APIC ID (31:0) of processors that have matching Cluster ID(31:16).

To enable cluster ID assignment in a fashion that matches the system topology characteristics and to enable efficient routing of logical mode lowest priority device interrupts in link based platform interconnects, the LDR are initialized by hardware based on the value of x2APIC ID upon x2APIC state transitions. Details of this initialization are provided in Section 10.12.10.2.

### 10.12.10.2 Deriving Logical x2APIC ID from the Local x2APIC ID

In x2APIC mode, the 32-bit logical x2APIC ID, which can be read from LDR, is derived from the 32-bit local x2APIC ID. Specifically, the 16-bit logical ID sub-field is derived by shifting 1 by the lowest 4 bits of the x2APIC ID, i.e. Logical ID =  $1 \ll x2APIC\ ID[3:0]$ . The remaining bits of the x2APIC ID then form the cluster ID portion of the logical x2APIC ID:

$$\text{Logical x2APIC ID} = [(x2APIC\ ID[19:4] \ll 16) | (1 \ll x2APIC\ ID[3:0])]$$

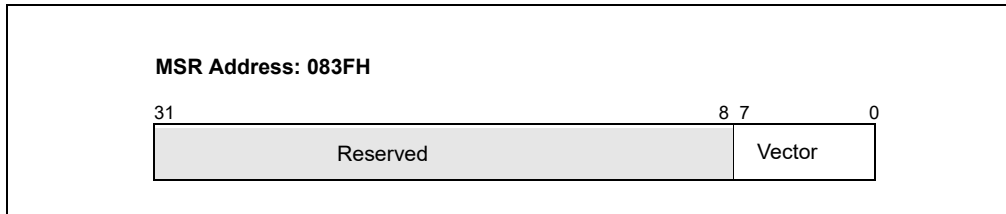
The use of the lowest 4 bits in the x2APIC ID implies that at least 16 APIC IDs are reserved for logical processors within a socket in multi-socket configurations. If more than 16 APIC IDs are reserved for logical processors in a socket/package then multiple cluster IDs can exist within the package.

The LDR initialization occurs whenever the x2APIC mode is enabled (see Section 10.12.5).

### 10.12.11 SELF IPI Register

SELF IPIs are used extensively by some system software. The x2APIC architecture introduces a new register interface. This new register is dedicated to the purpose of sending self-IPIs with the intent of enabling a highly optimized path for sending self-IPIs.

Figure 10-30 provides the layout of the SELF IPI register. System software only specifies the vector associated with the interrupt to be sent. The semantics of sending a self-IPI via the SELF IPI register are identical to sending a self targeted edge triggered fixed interrupt with the specified vector. Specifically the semantics are identical to the following settings for an inter-processor interrupt sent via the ICR - Destination Shorthand (ICR[19:18] = 01 (Self)), Trigger Mode (ICR[15] = 0 (Edge)), Delivery Mode (ICR[10:8] = 000 (Fixed)), Vector (ICR[7:0] = Vector).



**Figure 10-30. SELF IPI register**

The SELF IPI register is a write-only register. A RDMSR instruction with address of the SELF IPI register causes a general-protection exception.

The handling and prioritization of a self-IPI sent via the SELF IPI register is architecturally identical to that for an IPI sent via the ICR from a legacy xAPIC unit. Specifically the state of the interrupt would be tracked via the Interrupt Request Register (IRR) and In Service Register (ISR) and Trigger Mode Register (TMR) as if it were received from the system bus. Also sending the IPI via the Self Interrupt Register ensures that interrupt is delivered to the processor core. Specifically completion of the WRMSR instruction to the SELF IPI register implies that the interrupt has been logged into the IRR. As expected for edge triggered interrupts, depending on the processor priority and readiness to accept interrupts, it is possible that interrupts sent via the SELF IPI register or via the ICR with identical vectors can be combined.

## 10.13 APIC BUS MESSAGE FORMATS

This section describes the message formats used when transmitting messages on the serial APIC bus. The information described here pertains only to the Pentium and P6 family processors.

### 10.13.1 Bus Message Formats

The local and I/O APICs transmit three types of messages on the serial APIC bus: EOI message, short message, and non-focused lowest priority message. The purpose of each type of message and its format are described below.

### 10.13.2 EOI Message

Local APICs send 14-cycle EOI messages to the I/O APIC to indicate that a level triggered interrupt has been accepted by the processor. This interrupt, in turn, is a result of software writing into the EOI register of the local APIC. Table 10-1 shows the cycles in an EOI message.

**Table 10-1. EOI Message (14 Cycles)**

Cycle	Bit1	Bit0	
1	1	1	11 = EOI
2	ArbID3	0	Arbitration ID bits 3 through 0

**Table 10-1. EOI Message (14 Cycles) (Contd.)**

Cycle	Bit1	Bit0	
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	V7	V6	Interrupt vector V7 - V0
7	V5	V4	
8	V3	V2	
9	V1	V0	
10	C	C	Checksum for cycles 6 - 9
11	0	0	
12	A	A	Status Cycle 0
13	A1	A1	Status Cycle 1
14	0	0	Idle

The checksum is computed for cycles 6 through 9. It is a cumulative sum of the 2-bit (Bit1:Bit0) logical data values. The carry out of all but the last addition is added to the sum. If any APIC computes a different checksum than the one appearing on the bus in cycle 10, it signals an error, driving 11 on the APIC bus during cycle 12. In this case, the APICs disregard the message. The sending APIC will receive an appropriate error indication (see Section 10.5.3, "Error Handling") and resend the message. The status cycles are defined in Table 10-4.

### 10.13.2.1 Short Message

Short messages (21-cycles) are used for sending fixed, NMI, SMI, INIT, start-up, ExtINT and lowest-priority-with-focus interrupts. Table 10-2 shows the cycles in a short message.

**Table 10-2. Short Message (21 Cycles)**

Cycle	Bit1	Bit0	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination Mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	

**Table 10-2. Short Message (21 Cycles) (Contd.)**

Cycle	Bit1	Bit0	
17	C	C	Checksum for cycles 6-16
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	0	0	Idle

If the physical delivery mode is being used, then cycles 15 and 16 represent the APIC ID and cycles 13 and 14 are considered don't care by the receiver. If the logical delivery mode is being used, then cycles 13 through 16 are the 8-bit logical destination field.

For shorthands of "all-incl-self" and "all-excl-self," the physical delivery mode and an arbitration priority of 15 (D0:D3 = 1111) are used. The agent sending the message is the only one required to distinguish between the two cases. It does so using internal information.

When using lowest priority delivery with an existing focus processor, the focus processor identifies itself by driving 10 during cycle 19 and accepts the interrupt. This is an indication to other APICs to terminate arbitration. If the focus processor has not been found, the short message is extended on-the-fly to the non-focused lowest-priority message. Note that except for the EOI message, messages generating a checksum or an acceptance error (see Section 10.5.3, "Error Handling") terminate after cycle 21.

### 10.13.2.2 Non-focused Lowest Priority Message

These 34-cycle messages (see Table 10-3) are used in the lowest priority delivery mode when a focus processor is not present. Cycles 1 through 20 are same as for the short message. If during the status cycle (cycle 19) the state of the (A:A) flags is 10B, a focus processor has been identified, and the short message format is used (see Table 10-2). If the (A:A) flags are set to 00B, lowest priority arbitration is started and the 34-cycles of the non-focused lowest priority message are competed. For other combinations of status flags, refer to Section 10.13.2.3, "APIC Bus Status Cycles."

**Table 10-3. Non-Focused Lowest Priority Message (34 Cycles)**

Cycle	Bit0	Bit1	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	

**Table 10-3. Non-Focused Lowest Priority Message (34 Cycles) (Contd.)**

Cycle	Bit0	Bit1	
17	C	C	Checksum for cycles 6-16
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	P7	0	P7 - P0 = Inverted Processor Priority
22	P6	0	
23	P5	0	
24	P4	0	
25	P3	0	
26	P2	0	
27	P1	0	
28	P0	0	
29	ArbID3	0	Arbitration ID 3 -0
30	ArbID2	0	
31	ArbID1	0	
32	ArbID0	0	
33	A2	A2	Status Cycle
34	0	0	Idle

Cycles 21 through 28 are used to arbitrate for the lowest priority processor. The processors participating in the arbitration drive their inverted processor priority on the bus. Only the local APICs having free interrupt slots participate in the lowest priority arbitration. If no such APIC exists, the message will be rejected, requiring it to be tried at a later time.

Cycles 29 through 32 are also used for arbitration in case two or more processors have the same lowest priority. In the lowest priority delivery mode, all combinations of errors in cycle 33 (A2 A2) will set the “accept error” bit in the error status register (see Figure 10-9). Arbitration priority update is performed in cycle 20, and is not affected by errors detected in cycle 33. Only the local APIC that wins in the lowest priority arbitration, drives cycle 33. An error in cycle 33 will force the sender to resend the message.

### 10.13.2.3 APIC Bus Status Cycles

Certain cycles within an APIC bus message are status cycles. During these cycles the status flags (A:A) and (A1:A1) are examined. Table 10-4 shows how these status flags are interpreted, depending on the current delivery mode and existence of a focus processor.

**Table 10-4. APIC Bus Status Cycles Interpretation**

Delivery Mode	A Status	A1 Status	A2 Status	Update ArbID and Cycle#	Message Length	Retry
EOI	00: CS_OK	10: Accept	XX:	Yes, 13	14 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 13	14 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	14 Cycle	Yes
	11: CS_Error	XX:	XX:	No	14 Cycle	Yes
	10: Error	XX:	XX:	No	14 Cycle	Yes
	01: Error	XX:	XX:	No	14 Cycle	Yes
Fixed	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
NMI, SMI, INIT, ExtINT, Start-Up	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
Lowest	00: CS_OK, NoFocus	11: Do Lowest	10: Accept	Yes, 20	34 Cycle	No
	00: CS_OK, NoFocus	11: Do Lowest	11: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	11: Do Lowest	0X: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	10: End and Retry	XX:	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	0X: Error	XX:	No	34 Cycle	Yes
	10: CS_OK, Focus	XX:	XX:	Yes, 20	34 Cycle	No
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes



## 12. Updates to Chapter 15, Volume 3B

Change bars show changes to Chapter 15 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

-----  
Changes to this chapter: Addition of IOMCA information.



This chapter describes the machine-check architecture and machine-check exception mechanism found in the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors. See Chapter 6, “Interrupt 18—Machine-Check Exception (#MC),” for more information on machine-check exceptions. A brief description of the Pentium processor’s machine check capability is also given.

Additionally, a signaling mechanism for software to respond to hardware corrected machine check error is covered.

### 15.1 MACHINE-CHECK ARCHITECTURE

The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors implement a machine-check architecture that provides a mechanism for detecting and reporting hardware (machine) errors, such as: system bus errors, ECC errors, parity errors, cache errors, and TLB errors. It consists of a set of model-specific registers (MSRs) that are used to set up machine checking and additional banks of MSRs used for recording errors that are detected.

The processor signals the detection of an uncorrected machine-check error by generating a machine-check exception (#MC), which is an abort class exception. The implementation of the machine-check architecture does not ordinarily permit the processor to be restarted reliably after generating a machine-check exception. However, the machine-check-exception handler can collect information about the machine-check error from the machine-check MSRs.

Starting with 45 nm Intel 64 processor on which CPUID reports DisplayFamily\_DisplayModel as 06H\_1AH (see CPUID instruction in Chapter 3, “Instruction Set Reference, A-L” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*), the processor can report information on corrected machine-check errors and deliver a programmable interrupt for software to respond to MC errors, referred to as corrected machine-check error interrupt (CMCI). See Section 15.5 for detail.

Intel 64 processors supporting machine-check architecture and CMCI may also support an additional enhancement, namely, support for software recovery from certain uncorrected recoverable machine check errors. See Section 15.6 for detail.

### 15.2 COMPATIBILITY WITH PENTIUM PROCESSOR

The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors support and extend the machine-check exception mechanism introduced in the Pentium processor. The Pentium processor reports the following machine-check errors:

- data parity errors during read cycles
- unsuccessful completion of a bus cycle

The above errors are reported using the P5\_MC\_TYPE and P5\_MC\_ADDR MSRs (implementation specific for the Pentium processor). Use the RDMSR instruction to read these MSRs. See Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* for the addresses.

The machine-check error reporting mechanism that Pentium processors use is similar to that used in Pentium 4, Intel Xeon, Intel Atom, and P6 family processors. When an error is detected, it is recorded in P5\_MC\_TYPE and P5\_MC\_ADDR; the processor then generates a machine-check exception (#MC).

See Section 15.3.3, “Mapping of the Pentium Processor Machine-Check Errors to the Machine-Check Architecture,” and Section 15.10.2, “Pentium Processor Machine-Check Exception Handling,” for information on compatibility between machine-check code written to run on the Pentium processors and code written to run on P6 family processors.

## 15.3 MACHINE-CHECK MSRS

Machine check MSRs in the Pentium 4, Intel Atom, Intel Xeon, and P6 family processors consist of a set of global control and status registers and several error-reporting register banks. See Figure 15-1.

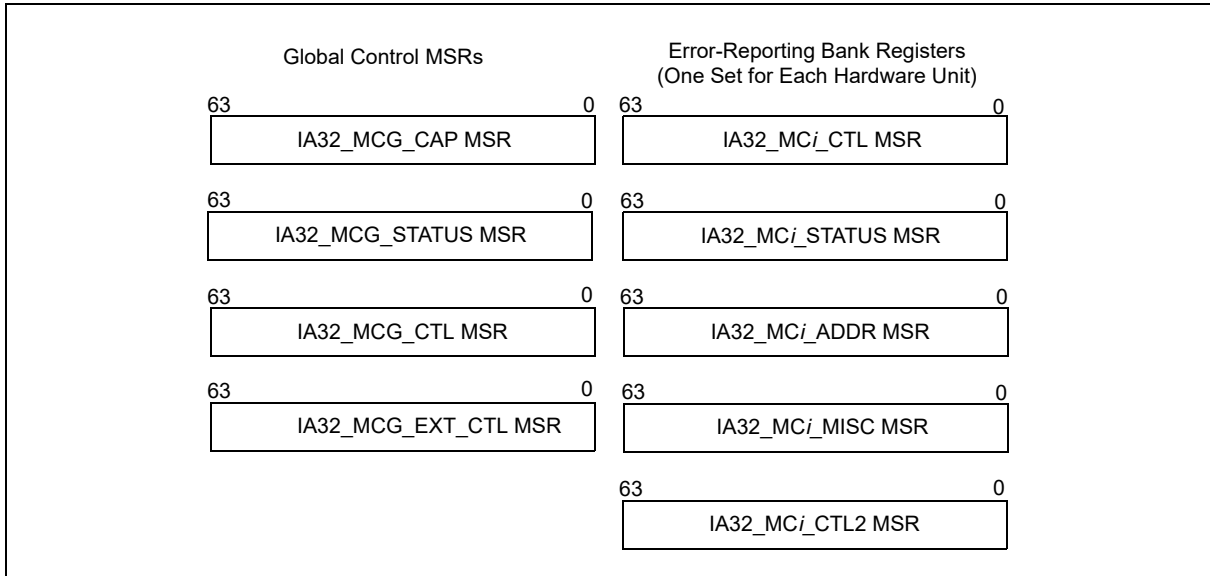


Figure 15-1. Machine-Check MSRs

Each error-reporting bank is associated with a specific hardware unit (or group of hardware units) in the processor. Use RDMSR and WRMSR to read and to write these registers.

### 15.3.1 Machine-Check Global Control MSRs

The machine-check global control MSRs include the IA32\_MCG\_CAP, IA32\_MCG\_STATUS, and optionally IA32\_MCG\_CTL and IA32\_MCG\_EXT\_CTL. See Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* for the addresses of these registers.

#### 15.3.1.1 IA32\_MCG\_CAP MSR

The IA32\_MCG\_CAP MSR is a read-only register that provides information about the machine-check architecture of the processor. Figure 15-2 shows the layout of the register.

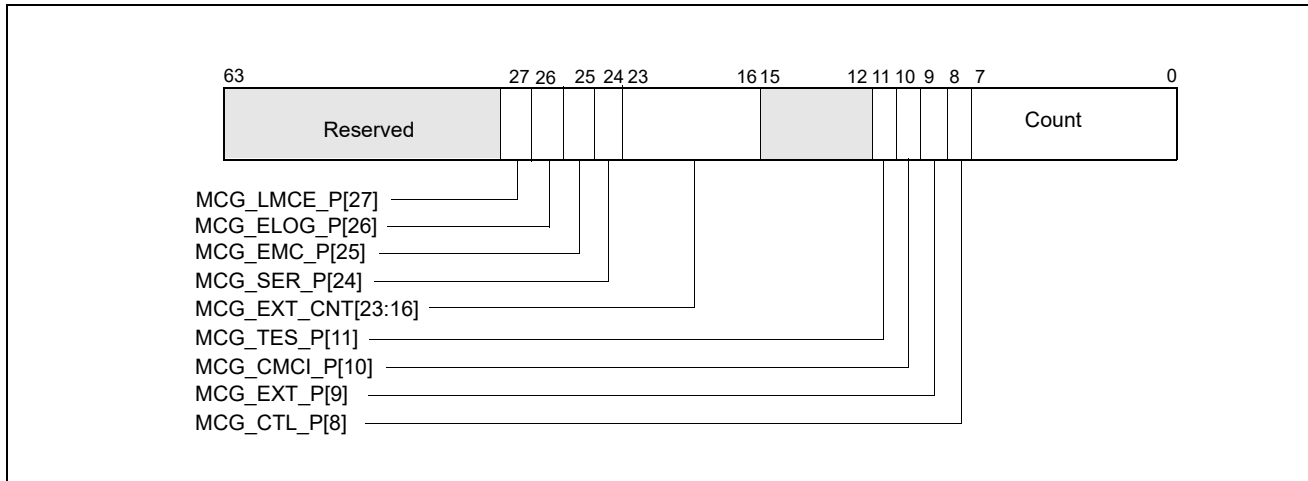


Figure 15-2. IA32\_MCG\_CAP Register

Where:

- **Count field, bits 7:0** — Indicates the number of hardware unit error-reporting banks available in a particular processor implementation.
- **MCG\_CTL\_P (control MSR present) flag, bit 8** — Indicates that the processor implements the IA32\_MCG\_CTL MSR when set; this register is absent when clear.
- **MCG\_EXT\_P (extended MSRs present) flag, bit 9** — Indicates that the processor implements the extended machine-check state registers found starting at MSR address 180H; these registers are absent when clear.
- **MCG\_CMCI\_P (Corrected MC error counting/signaling extension present) flag, bit 10** — Indicates (when set) that extended state and associated MSRs necessary to support the reporting of an interrupt on a corrected MC error event and/or count threshold of corrected MC errors, is present. When this bit is set, it does not imply this feature is supported across all banks. Software should check the availability of the necessary logic on a bank by bank basis when using this signaling capability (i.e. bit 30 settable in individual IA32\_MCi\_CTL2 register).
- **MCG\_TES\_P (threshold-based error status present) flag, bit 11** — Indicates (when set) that bits 56:53 of the IA32\_MCi\_STATUS MSR are part of the architectural space. Bits 56:55 are reserved, and bits 54:53 are used to report threshold-based error status. Note that when MCG\_TES\_P is not set, bits 56:53 of the IA32\_MCi\_STATUS MSR are model-specific.
- **MCG\_EXT\_CNT, bits 23:16** — Indicates the number of extended machine-check state registers present. This field is meaningful only when the MCG\_EXT\_P flag is set.
- **MCG\_SER\_P (software error recovery support present) flag, bit 24** — Indicates (when set) that the processor supports software error recovery (see Section 15.6), and IA32\_MCi\_STATUS MSR bits 56:55 are used to report the signaling of uncorrected recoverable errors and whether software must take recovery actions for uncorrected errors. Note that when MCG\_TES\_P is not set, bits 56:53 of the IA32\_MCi\_STATUS MSR are model-specific. If MCG\_TES\_P is set but MCG\_SER\_P is not set, bits 56:55 are reserved.
- **MCG EMC\_P (Enhanced Machine Check Capability) flag, bit 25** — Indicates (when set) that the processor supports enhanced machine check capabilities for firmware first signaling.
- **MCG\_ELOG\_P (extended error logging) flag, bit 26** — Indicates (when set) that the processor allows platform firmware to be invoked when an error is detected so that it may provide additional platform specific information in an ACPI format “Generic Error Data Entry” that augments the data included in machine check bank registers.

For additional information about extended error logging interface, see

<https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/enhanced-mca-logging-xeon-paper.pdf>.

- **MCG\_LMCE\_P (local machine check exception) flag, bit 27** — Indicates (when set) that the following interfaces are present:
  - an extended state LMCE\_S (located in bit 3 of IA32\_MCG\_STATUS), and
  - the IA32\_MCG\_EXT\_CTL MSR, necessary to support Local Machine Check Exception (LMCE).

A non-zero MCG\_LMCE\_P indicates that, when LMCE is enabled as described in Section 15.3.1.5, some machine check errors may be delivered to only a single logical processor.

The effect of writing to the IA32\_MCG\_CAP MSR is undefined.

### 15.3.1.2 IA32\_MCG\_STATUS MSR

The IA32\_MCG\_STATUS MSR describes the current state of the processor after a machine-check exception has occurred (see Figure 15-3).

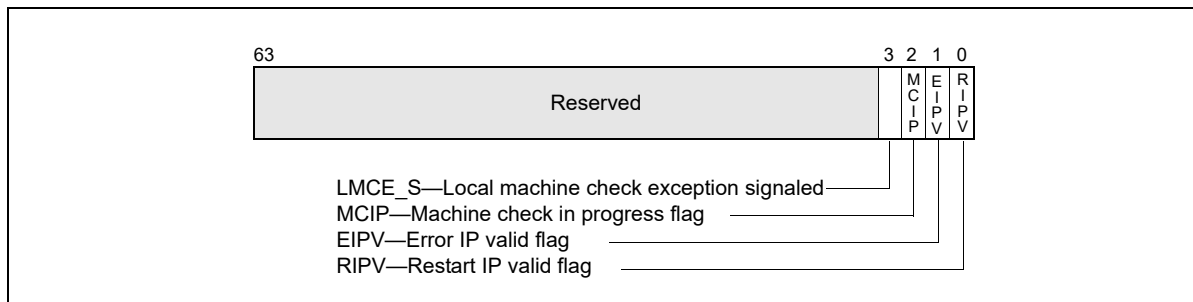


Figure 15-3. IA32\_MCG\_STATUS Register

Where:

- **RIPV (restart IP valid) flag, bit 0** — Indicates (when set) that program execution can be restarted reliably at the instruction pointed to by the instruction pointer pushed on the stack when the machine-check exception is generated. When clear, the program cannot be reliably restarted at the pushed instruction pointer.
- **EIPV (error IP valid) flag, bit 1** — Indicates (when set) that the instruction pointed to by the instruction pointer pushed onto the stack when the machine-check exception is generated is directly associated with the error. When this flag is cleared, the instruction pointed to may not be associated with the error.
- **MCIP (machine check in progress) flag, bit 2** — Indicates (when set) that a machine-check exception was generated. Software can set or clear this flag. The occurrence of a second Machine-Check Event while MCIP is set will cause the processor to enter a shutdown state. For information on processor behavior in the shutdown state, please refer to the description in Chapter 6, "Interrupt and Exception Handling": "Interrupt 8—Double Fault Exception (#DF)".
- **LMCE\_S (local machine check exception signaled), bit 3** — Indicates (when set) that a local machine-check exception was generated. This indicates that the current machine-check event was delivered to only this logical processor.

Bits 63:04 in IA32\_MCG\_STATUS are reserved. An attempt to write to IA32\_MCG\_STATUS with any value other than 0 would result in #GP.

### 15.3.1.3 IA32\_MCG\_CTL MSR

The IA32\_MCG\_CTL MSR is present if the capability flag MCG\_CTL\_P is set in the IA32\_MCG\_CAP MSR.

IA32\_MCG\_CTL controls the reporting of machine-check exceptions. If present, writing 1s to this register enables machine-check features and writing all 0s disables machine-check features. All other values are undefined and/or implementation specific.

### 15.3.1.4 IA32\_MCG\_EXT\_CTL MSR

The IA32\_MCG\_EXT\_CTL MSR is present if the capability flag MCG\_LMCE\_P is set in the IA32\_MCG\_CAP MSR. IA32\_MCG\_EXT\_CTL.LMCE\_EN (bit 0) allows the processor to signal some MCEs to only a single logical processor in the system.

If MCG\_LMCE\_P is not set in IA32\_MCG\_CAP, or platform software has not enabled LMCE by setting IA32\_FEATURE\_CONTROL.LMCE\_ON (bit 20), any attempt to write or read IA32\_MCG\_EXT\_CTL will result in #GP. The IA32\_MCG\_EXT\_CTL MSR is cleared on RESET.

Figure 15-4 shows the layout of the IA32\_MCG\_EXT\_CTL register

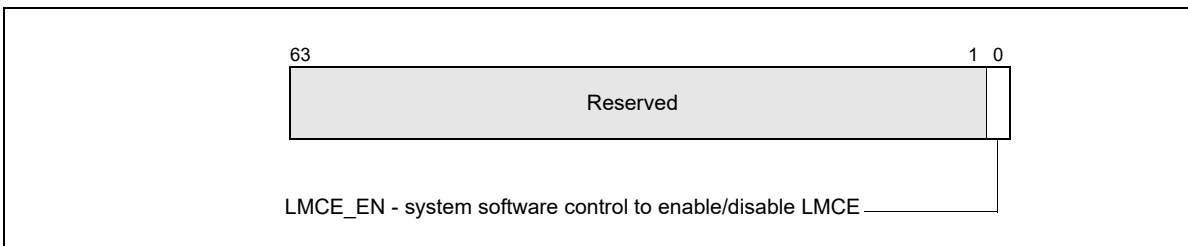


Figure 15-4. IA32\_MCG\_EXT\_CTL Register

where

- **LMCE\_EN (local machine check exception enable) flag, bit 0** - System software sets this to allow hardware to signal some MCEs to only a single logical processor. System software can set LMCE\_EN only if the platform software has configured IA32\_FEATURE\_CONTROL as described in Section 15.3.1.5.

### 15.3.1.5 Enabling Local Machine Check

The intended usage of LMCE requires proper configuration by both platform software and system software. Platform software can turn LMCE on by setting bit 20 (LMCE\_ON) in IA32\_FEATURE\_CONTROL MSR (MSR address 3AH).

System software must ensure that both IA32\_FEATURE\_CONTROL.Lock (bit 0) and IA32\_FEATURE\_CONTROL.LMCE\_ON (bit 20) are set before attempting to set IA32\_MCG\_EXT\_CTL.LMCE\_EN (bit 0). When system software has enabled LMCE, then hardware will determine if a particular error can be delivered only to a single logical processor. Software should make no assumptions about the type of error that hardware can choose to deliver as LMCE. The severity and override rules stay the same as described in Table 15-8 to determine the recovery actions.

## 15.3.2 Error-Reporting Register Banks

Each error-reporting register bank can contain the IA32\_MCi\_CTL, IA32\_MCi\_STATUS, IA32\_MCi\_ADDR, and IA32\_MCi\_MISC MSRs. The number of reporting banks is indicated by bits [7:0] of IA32\_MCG\_CAP MSR (address 0179H). The first error-reporting register (IA32\_MC0\_CTL) always starts at address 400H.

See Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* for addresses of the error-reporting registers in the Pentium 4, Intel Atom, and Intel Xeon processors; and for addresses of the error-reporting registers P6 family processors.

### 15.3.2.1 IA32\_MCi\_CTL MSRs

The IA32\_MCi\_CTL MSR controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units). Each of the 64 flags (EE<sub>j</sub>) represents a potential error. Setting an EE<sub>j</sub> flag enables signaling #MC of the associated error and clearing it disables signaling of the error. Error logging happens regardless of the setting of these bits. The processor drops writes to bits that are not implemented. Figure 15-5 shows the bit fields of IA32\_MCi\_CTL.

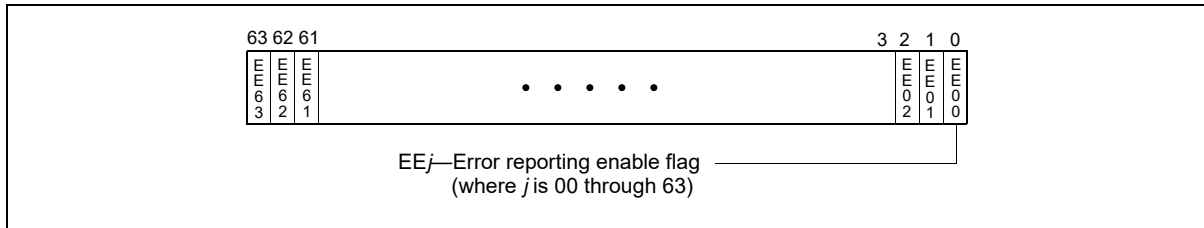


Figure 15-5. IA32\_MCi\_CTL Register

**NOTE**

For P6 family processors, processors based on Intel Core microarchitecture (excluding those on which on which CPUID reports DisplayFamily\_DisplayModel as 06H\_1AH and onward): the operating system or executive software must not modify the contents of the IA32\_MC0\_CTL MSR. This MSR is internally aliased to the EBL\_CR\_POWERON MSR and controls platform-specific error handling features. System specific firmware (the BIOS) is responsible for the appropriate initialization of the IA32\_MC0\_CTL MSR. P6 family processors only allow the writing of all 1s or all 0s to the IA32\_MCi\_CTL MSR.

**15.3.2.2 IA32\_MCi\_STATUS MSRS**

Each IA32\_MCi\_STATUS MSR contains information related to a machine-check error if its VAL (valid) flag is set (see Figure 15-6). Software is responsible for clearing IA32\_MCi\_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.

**NOTE**

Figure 15-6 depicts the IA32\_MCi\_STATUS MSR when IA32\_MCG\_CAP[24] = 1, IA32\_MCG\_CAP[11] = 1 and IA32\_MCG\_CAP[10] = 1. When IA32\_MCG\_CAP[24] = 0 and IA32\_MCG\_CAP[11] = 1, bits 56:55 is reserved and bits 54:53 for threshold-based error reporting. When IA32\_MCG\_CAP[11] = 0, bits 56:53 are part of the “Other Information” field. The use of bits 54:53 for threshold-based error reporting began with Intel Core Duo processors, and is currently used for cache memory. See Section 15.4, “Enhanced Cache Error reporting,” for more information. When IA32\_MCG\_CAP[10] = 0, bits 52:38 are part of the “Other Information” field. The use of bits 52:38 for corrected MC error count is introduced with Intel 64 processor on which CPUID reports DisplayFamily\_DisplayModel as 06H\_1AH.

Where:

- **MCA (machine-check architecture) error code field, bits 15:0** — Specifies the machine-check architecture-defined error code for the machine-check error condition detected. The machine-check architecture-defined error codes are guaranteed to be the same for all IA-32 processors that implement the machine-check architecture. See Section 15.9, “Interpreting the MCA Error Codes,” and Chapter 16, “Interpreting Machine-Check Error Codes”, for information on machine-check error codes.
- **Model-specific error code field, bits 31:16** — Specifies the model-specific error code that uniquely identifies the machine-check error condition detected. The model-specific error codes may differ among IA-32 processors for the same machine-check error condition. See Chapter 16, “Interpreting Machine-Check Error Codes” for information on model-specific error codes.
- **Reserved, Error Status, and Other Information fields, bits 56:32** —
  - If IA32\_MCG\_CAP.MCG\_EMC\_P[bit 25] is 0, bits 37:32 contain “Other Information” that is implementation-specific and is not part of the machine-check architecture.
  - If IA32\_MCG\_CAP.MCG\_EMC\_P is 1, “Other Information” is in bits 36:32. If bit 37 is 0, system firmware has not changed the contents of IA32\_MCi\_STATUS. If bit 37 is 1, system firmware may have edited the contents of IA32\_MCi\_STATUS.
  - If IA32\_MCG\_CAP.MCG\_CMCI\_P[bit 10] is 0, bits 52:38 also contain “Other Information” (in the same sense as bits 37:32).





- **PCC (processor context corrupt) flag, bit 57** — Indicates (when set) that the state of the processor might have been corrupted by the error condition detected and that reliable restarting of the processor may not be possible. When clear, this flag indicates that the error did not affect the processor’s state, and software may be able to restart. When system software supports recovery, consult Section 15.10.4, “Machine-Check Software Handler Guidelines for Error Recovery” for additional rules that apply.
- **ADDRV (IA32\_MCi\_ADDR register valid) flag, bit 58** — Indicates (when set) that the IA32\_MCi\_ADDR register contains the address where the error occurred (see Section 15.3.2.3, “IA32\_MCi\_ADDR MSRs”). When clear, this flag indicates that the IA32\_MCi\_ADDR register is either not implemented or does not contain the address where the error occurred. Do not read these registers if they are not implemented in the processor.
- **MISCV (IA32\_MCi\_MISC register valid) flag, bit 59** — Indicates (when set) that the IA32\_MCi\_MISC register contains additional information regarding the error. When clear, this flag indicates that the IA32\_MCi\_MISC register is either not implemented or does not contain additional information regarding the error. Do not read these registers if they are not implemented in the processor.
- **EN (error enabled) flag, bit 60** — Indicates (when set) that the error was enabled by the associated EEj bit of the IA32\_MCi\_CTL register.
- **UC (error uncorrected) flag, bit 61** — Indicates (when set) that the processor did not or was not able to correct the error condition. When clear, this flag indicates that the processor was able to correct the error condition.
- **OVER (machine check overflow) flag, bit 62** — Indicates (when set) that a machine-check error occurred while the results of a previous error were still in the error-reporting register bank (that is, the VAL bit was already set in the IA32\_MCi\_STATUS register). The processor sets the OVER flag and software is responsible for clearing it. In general, enabled errors are written over disabled errors, and uncorrected errors are written over corrected errors. Uncorrected errors are not written over previous valid uncorrected errors. When MCG\_CMCI\_P is set, corrected errors may not set the OVER flag. Software can rely on corrected error count in IA32\_MCi\_Status[52:38] to determine if any additional corrected errors may have occurred. For more information, see Section 15.3.2.2.1, “Overwrite Rules for Machine Check Overflow”.
- **VAL (IA32\_MCi\_STATUS register valid) flag, bit 63** — Indicates (when set) that the information within the IA32\_MCi\_STATUS register is valid. When this flag is set, the processor follows the rules given for the OVER flag in the IA32\_MCi\_STATUS register when overwriting previously valid entries. The processor sets the VAL flag and software is responsible for clearing it.

### 15.3.2.2.1 Overwrite Rules for Machine Check Overflow

Table 15-2 shows the overwrite rules for how to treat a second event if the cache has already posted an event to the MC bank – that is, what to do if the valid bit for an MC bank already is set to 1. When more than one structure posts events in a given bank, these rules specify whether a new event will overwrite a previous posting or not. These rules define a priority for uncorrected (highest priority), yellow, and green/unmonitored (lowest priority) status.

In Table 15-2, the values in the two left-most columns are IA32\_MCi\_STATUS[54:53].

**Table 15-2. Overwrite Rules for Enabled Errors**

First Event	Second Event	UC bit	Color	MCA Info
00/green	00/green	0	00/green	either
00/green	yellow	0	yellow	second error
yellow	00/green	0	yellow	first error
yellow	yellow	0	yellow	either
00/green/yellow	UC	1	undefined	second
UC	00/green/yellow	1	undefined	first

If a second event overwrites a previously posted event, the information (as guarded by individual valid bits) in the MCI bank is entirely from the second event. Similarly, if a first event is retained, all of the information previously posted for that event is retained. In general, when the logged error or the recent error is a corrected error, the OVER bit (MCI\_Status[62]) may be set to indicate an overflow. When MCG\_CMCI\_P is set in IA32\_MCG\_CAP, system software should consult IA32\_MCi\_STATUS[52:38] to determine if additional corrected errors may have



occurred. Software may re-read IA32\_MCi\_STATUS, IA32\_MCi\_ADDR and IA32\_MCi\_MISC appropriately to ensure data collected represent the last error logged.

After software polls a posting and clears the register, the valid bit is no longer set and therefore the meaning of the rest of the bits, including the yellow/green/00 status field in bits 54:53, is undefined. The yellow/green indication will only be posted for events associated with monitored structures – otherwise the unmonitored (00) code will be posted in IA32\_MCi\_STATUS[54:53].

### 15.3.2.3 IA32\_MCi\_ADDR MSRs

The IA32\_MCi\_ADDR MSR contains the address of the code or data memory location that produced the machine-check error if the ADDR\_V flag in the IA32\_MCi\_STATUS register is set (see Section 15-7, “IA32\_MCi\_ADDR MSR”). The IA32\_MCi\_ADDR register is either not implemented or contains no address if the ADDR\_V flag in the IA32\_MCi\_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general protection exception.

The address returned is an offset into a segment, linear address, or physical address. This depends on the error encountered. When these registers are implemented, these registers can be cleared by explicitly writing 0s to these registers. Writing 1s to these registers will cause a general-protection exception. See Figure 15-7.

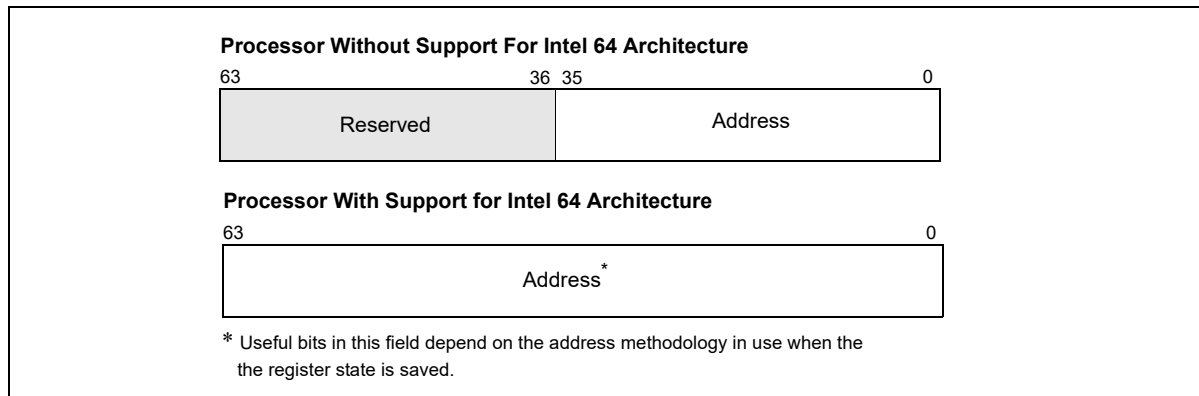


Figure 15-7. IA32\_MCi\_ADDR MSR

### 15.3.2.4 IA32\_MCi\_MISC MSRs

The IA32\_MCi\_MISC MSR contains additional information describing the machine-check error if the MISC\_V flag in the IA32\_MCi\_STATUS register is set. The IA32\_MCi\_MISC\_MSR is either not implemented or does not contain additional information if the MISC\_V flag in the IA32\_MCi\_STATUS register is clear.

When not implemented in the processor, all reads and writes to this MSR will cause a general protection exception. When implemented in a processor, these registers can be cleared by explicitly writing all 0s to them; writing 1s to them causes a general-protection exception to be generated. This register is not implemented in any of the error-reporting register banks for the P6 or Intel Atom family processors.

If both MISC\_V and IA32\_MCG\_CAP[24] are set, the IA32\_MCi\_MISC\_MSR is defined according to Figure 15-8 to support software recovery of uncorrected errors (see Section 15.6).

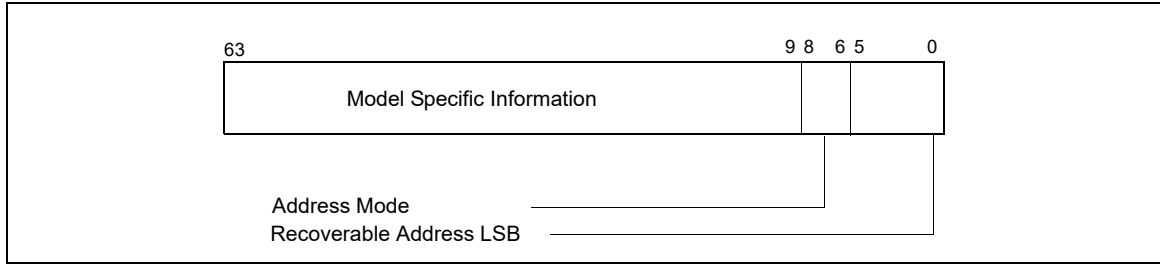


Figure 15-8. UCR Support in IA32\_MCI\_MISC Register

- Recoverable Address LSB (bits 5:0): The lowest valid recoverable address bit. Indicates the position of the least significant bit (LSB) of the recoverable error address. For example, if the processor logs bits [43:9] of the address, the LSB sub-field in IA32\_MCI\_MISC is 01001b (9 decimal). For this example, bits [8:0] of the recoverable error address in IA32\_MCI\_ADDR should be ignored.
- Address Mode (bits 8:6): Address mode for the address logged in IA32\_MCI\_ADDR. The supported address modes are given in Table 15-3.

Table 15-3. Address Mode in IA32\_MCI\_MISC[8:6]

IA32_MCI_MISC[8:6] Encoding	Definition
000	Segment Offset
001	Linear Address
010	Physical Address
011	Memory Address
100 to 110	Reserved
111	Generic

- Model Specific Information (bits 63:9): Not architecturally defined.

### 15.3.2.4.2 IOMCA

Logging and Signaling of errors from PCI Express domain is governed by PCI Express Advanced Error Reporting (AER) architecture. PCI Express architecture divides errors in two categories: Uncorrectable errors and Correctable errors. Uncorrectable errors can further be classified as Fatal or Non-Fatal. Uncorrected IO errors are signaled to the system software either as AER Message Signaled Interrupt (MSI) or via platform specific mechanisms such as NMI. Generally, the signaling mechanism is controlled by BIOS and/or platform firmware. Certain processors support an error handling mode, called IOMCA mode, where Uncorrected PCI Express errors are signaled in the form of machine check exception and logged in machine check banks.

When a processor is in this mode, Uncorrected PCI Express errors are logged in the MCACOD field of the IA32\_MCI\_STATUS register as Generic I/O error. The corresponding MCA error code is defined in Table 15-8. IA32\_MCI\_Status [15:0] Simple Error Code Encoding. Machine check logging complements and does not replace AER logging that occurs inside the PCI Express hierarchy. The PCI Express Root Complex and Endpoints continue to log the error in accordance with PCI Express AER mechanism. In IOMCA mode, MCI\_MISC register in the bank that logged IOMCA can optionally contain information that link the Machine Check logs with the AER logs or proprietary logs. In such a scenario, the machine check handler can utilize the contents of MCI\_MISC to locate the next level of error logs corresponding to the same error. Specifically, if MCI\_Status.MISCV is 1 and MCACOD is 0x0E0B, MCI\_MISC contains the PCI Express address of the Root Complex device containing the AER Logs. Software can consult the header type and class code registers in the Root Complex device's PCIe Configuration space to determine what type of device it is. This Root Complex device can either be a PCI Express Root Port, PCI Express Root Complex Event Collector or a proprietary device.

Errors that originate from PCI Express or Legacy Endpoints are logged in the corresponding Root Port in addition to the generating device. If `MISCV=1` and `MCi_MISC` contains the address of the Root Port or a Root Complex Event collector, software can parse the AER logs to learn more about the error.

If `MISCV=1` and `MCi_MISC` points to a device that is neither a Root Complex Event Collector nor a Root Port, software must consult the Vendor ID/Device ID and use device specific knowledge to locate and interpret the error log registers. In some cases, the Root Complex device configuration space may not be accessible to the software and both the Vendor and Device ID read as `0xFFFF`.

- The format of `MCi_MISC` for IOMCA errors is shown in Table 15-4.

**Table 15-4. Address Mode in IA32\_MCi\_MISC[8:6]**

63:40	39:32	31:16	15:9	8:6	5:0
RSVD	PCI Express Segment number	PCI Express Requestor ID	RSVD	ADDR MODE <sup>1</sup>	RECOV ADDR LSB <sup>1</sup>

**NOTES:**

1. Not Applicable if `ADDRV=0`.

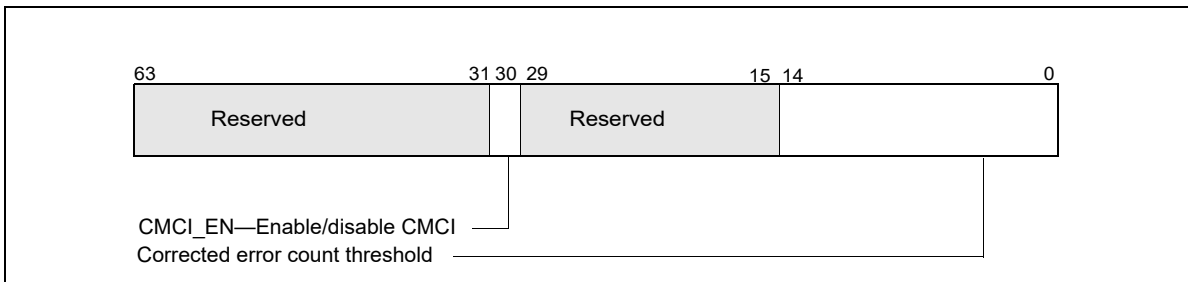
Refer to PCI Express Specification 3.0 for definition of PCI Express Requestor ID and AER architecture. Refer to PCI Firmware Specification 3.0 for an explanation of PCI Express Segment number and how software can access configuration space of a PCI Express device given the segment number and Requestor ID.

### 15.3.2.5 IA32\_MCi\_CTL2 MSRs

The `IA32_MCi_CTL2` MSR provides the programming interface to use corrected MC error signaling capability that is indicated by `IA32_MCG_CAP[10] = 1`. Software must check for the presence of `IA32_MCi_CTL2` on a per-bank basis.

When `IA32_MCG_CAP[10] = 1`, the `IA32_MCi_CTL2` MSR for each bank exists, i.e. reads and writes to these MSR are supported. However, signaling interface for corrected MC errors may not be supported in all banks.

The layout of `IA32_MCi_CTL2` is shown in Figure 15-9:



**Figure 15-9. IA32\_MCi\_CTL2 Register**

- **Corrected error count threshold, bits 14:0** — Software must initialize this field. The value is compared with the corrected error count field in `IA32_MCi_STATUS`, bits 38 through 52. An overflow event is signaled to the CMCI LVT entry (see Table 10-1) in the APIC when the count value equals the threshold value. The new LVT entry in the APIC is at `02F0H` offset from the `APIC_BASE`. If CMCI interface is not supported for a particular bank (but `IA32_MCG_CAP[10] = 1`), this field will always read 0.
- **CMCI\_EN (Corrected error interrupt enable/disable/indicator), bits 30** — Software sets this bit to enable the generation of corrected machine-check error interrupt (CMCI). If CMCI interface is not supported for a particular bank (but `IA32_MCG_CAP[10] = 1`), this bit is writeable but will always return 0 for that bank. This bit also indicates CMCI is supported or not supported in the corresponding bank. See Section 15.5 for details of software detection of CMCI facility.

Some microarchitectural sub-systems that are the source of corrected MC errors may be shared by more than one logical processors. Consequently, the facilities for reporting MC errors and controlling mechanisms may be shared by more than one logical processors. For example, the IA32\_MC<sub>i</sub>\_CTL2 MSR is shared between logical processors sharing a processor core. Software is responsible to program IA32\_MC<sub>i</sub>\_CTL2 MSR in a consistent manner with CMCi delivery and usage.

After processor reset, IA32\_MC<sub>i</sub>\_CTL2 MSRs are zero'ed.

### 15.3.2.6 IA32\_MCG Extended Machine Check State MSRs

The Pentium 4 and Intel Xeon processors implement a variable number of extended machine-check state MSRs. The MCG\_EXT\_P flag in the IA32\_MCG\_CAP MSR indicates the presence of these extended registers, and the MCG\_EXT\_CNT field indicates the number of these registers actually implemented. See Section 15.3.1.1, "IA32\_MCG\_CAP MSR." Also see Table 15-5.

**Table 15-5. Extended Machine Check State MSRs in Processors Without Support for Intel 64 Architecture**

MSR	Address	Description
IA32_MCG_EAX	180H	Contains state of the EAX register at the time of the machine-check error.
IA32_MCG_EBX	181H	Contains state of the EBX register at the time of the machine-check error.
IA32_MCG_ECX	182H	Contains state of the ECX register at the time of the machine-check error.
IA32_MCG_EDX	183H	Contains state of the EDX register at the time of the machine-check error.
IA32_MCG_ESI	184H	Contains state of the ESI register at the time of the machine-check error.
IA32_MCG_EDI	185H	Contains state of the EDI register at the time of the machine-check error.
IA32_MCG_EBP	186H	Contains state of the EBP register at the time of the machine-check error.
IA32_MCG_ESP	187H	Contains state of the ESP register at the time of the machine-check error.
IA32_MCG_EFLAGS	188H	Contains state of the EFLAGS register at the time of the machine-check error.
IA32_MCG_EIP	189H	Contains state of the EIP register at the time of the machine-check error.
IA32_MCG_MISC	18AH	When set, indicates that a page assist or page fault occurred during DS normal operation.

In processors with support for Intel 64 architecture, 64-bit machine check state MSRs are aliased to the legacy MSRs. In addition, there may be registers beyond IA32\_MCG\_MISC. These may include up to five reserved MSRs (IA32\_MCG\_RESERVED[1:5]) and save-state MSRs for registers introduced in 64-bit mode. See Table 15-6.

**Table 15-6. Extended Machine Check State MSRs In Processors With Support For Intel 64 Architecture**

MSR	Address	Description
IA32_MCG_RAX	180H	Contains state of the RAX register at the time of the machine-check error.
IA32_MCG_RBX	181H	Contains state of the RBX register at the time of the machine-check error.
IA32_MCG_RCX	182H	Contains state of the RCX register at the time of the machine-check error.
IA32_MCG_RDX	183H	Contains state of the RDX register at the time of the machine-check error.
IA32_MCG_RSI	184H	Contains state of the RSI register at the time of the machine-check error.
IA32_MCG_RDI	185H	Contains state of the RDI register at the time of the machine-check error.
IA32_MCG_RBP	186H	Contains state of the RBP register at the time of the machine-check error.
IA32_MCG_RSP	187H	Contains state of the RSP register at the time of the machine-check error.
IA32_MCG_RFLAGS	188H	Contains state of the RFLAGS register at the time of the machine-check error.
IA32_MCG_RIP	189H	Contains state of the RIP register at the time of the machine-check error.

**Table 15-6. Extended Machine Check State MSRs  
In Processors With Support For Intel 64 Architecture (Contd.)**

MSR	Address	Description
IA32_MCG_MISC	18AH	When set, indicates that a page assist or page fault occurred during DS normal operation.
IA32_MCG_RSERVED[1:5]	18BH-18FH	These registers, if present, are reserved.
IA32_MCG_R8	190H	Contains state of the R8 register at the time of the machine-check error.
IA32_MCG_R9	191H	Contains state of the R9 register at the time of the machine-check error.
IA32_MCG_R10	192H	Contains state of the R10 register at the time of the machine-check error.
IA32_MCG_R11	193H	Contains state of the R11 register at the time of the machine-check error.
IA32_MCG_R12	194H	Contains state of the R12 register at the time of the machine-check error.
IA32_MCG_R13	195H	Contains state of the R13 register at the time of the machine-check error.
IA32_MCG_R14	196H	Contains state of the R14 register at the time of the machine-check error.
IA32_MCG_R15	197H	Contains state of the R15 register at the time of the machine-check error.

When a machine-check error is detected on a Pentium 4 or Intel Xeon processor, the processor saves the state of the general-purpose registers, the R/EFLAGS register, and the R/EIP in these extended machine-check state MSRs. This information can be used by a debugger to analyze the error.

These registers are read/write to zero registers. This means software can read them; but if software writes to them, only all zeros is allowed. If software attempts to write a non-zero value into one of these registers, a general-protection (#GP) exception is generated. These registers are cleared on a hardware reset (power-up or RESET), but maintain their contents following a soft reset (INIT reset).

### 15.3.3 Mapping of the Pentium Processor Machine-Check Errors to the Machine-Check Architecture

The Pentium processor reports machine-check errors using two registers: P5\_MC\_TYPE and P5\_MC\_ADDR. The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors map these registers to the IA32\_MCI\_STATUS and IA32\_MCI\_ADDR in the error-reporting register bank. This bank reports on the same type of external bus errors reported in P5\_MC\_TYPE and P5\_MC\_ADDR.

The information in these registers can then be accessed in two ways:

- By reading the IA32\_MCI\_STATUS and IA32\_MCI\_ADDR registers as part of a general machine-check exception handler written for Pentium 4, Intel Atom and P6 family processors.
- By reading the P5\_MC\_TYPE and P5\_MC\_ADDR registers using the RDMSR instruction.

The second capability permits a machine-check exception handler written to run on a Pentium processor to be run on a Pentium 4, Intel Xeon, Intel Atom, or P6 family processor. There is a limitation in that information returned by the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors is encoded differently than information returned by the Pentium processor. To run a Pentium processor machine-check exception handler on a Pentium 4, Intel Xeon, Intel Atom, or P6 family processor; the handler must be written to interpret P5\_MC\_TYPE encodings correctly.

## 15.4 ENHANCED CACHE ERROR REPORTING

Starting with Intel Core Duo processors, cache error reporting was enhanced. In earlier Intel processors, cache status was based on the number of correction events that occurred in a cache. In the new paradigm, called "threshold-based error status", cache status is based on the number of lines (ECC blocks) in a cache that incur repeated corrections. The threshold is chosen by Intel, based on various factors. If a processor supports threshold-based error status, it sets IA32\_MCG\_CAP[11] (MCG\_TES\_P) to 1; if not, to 0.

A processor that supports enhanced cache error reporting contains hardware that tracks the operating status of certain caches and provides an indicator of their “health”. The hardware reports a “green” status when the number of lines that incur repeated corrections is at or below a pre-defined threshold, and a “yellow” status when the number of affected lines exceeds the threshold. Yellow status means that the cache reporting the event is operating correctly, but you should schedule the system for servicing within a few weeks.

Intel recommends that you rely on this mechanism for structures supported by threshold-base error reporting.

The CPU/system/platform response to a yellow event should be less severe than its response to an uncorrected error. An uncorrected error means that a serious error has actually occurred, whereas the yellow condition is a warning that the number of affected lines has exceeded the threshold but is not, in itself, a serious event: the error was corrected and system state was not compromised.

The green/yellow status indicator is not a foolproof early warning for an uncorrected error resulting from the failure of two bits in the same ECC block. Such a failure can occur and cause an uncorrected error before the yellow threshold is reached. However, the chance of an uncorrected error increases as the number of affected lines increases.

## 15.5 CORRECTED MACHINE CHECK ERROR INTERRUPT

Corrected machine-check error interrupt (CMCI) is an architectural enhancement to the machine-check architecture. It provides capabilities beyond those of threshold-based error reporting (Section 15.4). With threshold-based error reporting, software is limited to use periodic polling to query the status of hardware corrected MC errors. CMCI provides a signaling mechanism to deliver a local interrupt based on threshold values that software can program using the IA32\_MCi\_CTL2 MSR.

CMCI is disabled by default. System software is required to enable CMCI for each IA32\_MCi bank that support the reporting of hardware corrected errors if IA32\_MCG\_CAP[10] = 1.

System software use IA32\_MCi\_CTL2 MSR to enable/disable the CMCI capability for each bank and program threshold values into IA32\_MCi\_CTL2 MSR. CMCI is not affected by the CR4.MCE bit, and it is not affected by the IA32\_MCi\_CTL MSR.

To detect the existence of thresholding for a given bank, software writes only bits 14:0 with the threshold value. If the bits persist, then thresholding is available (and CMCI is available). If the bits are all 0's, then no thresholding exists. To detect that CMCI signaling exists, software writes a 1 to bit 30 of the MCI\_CTL2 register. Upon subsequent read, if bit 30 = 0, no CMCI is available for this bank and no corrected or UCNA errors will be reported on this bank. If bit 30 = 1, then CMCI is available and enabled.

### 15.5.1 CMCI Local APIC Interface

The operation of CMCI is depicted in Figure 15-10.

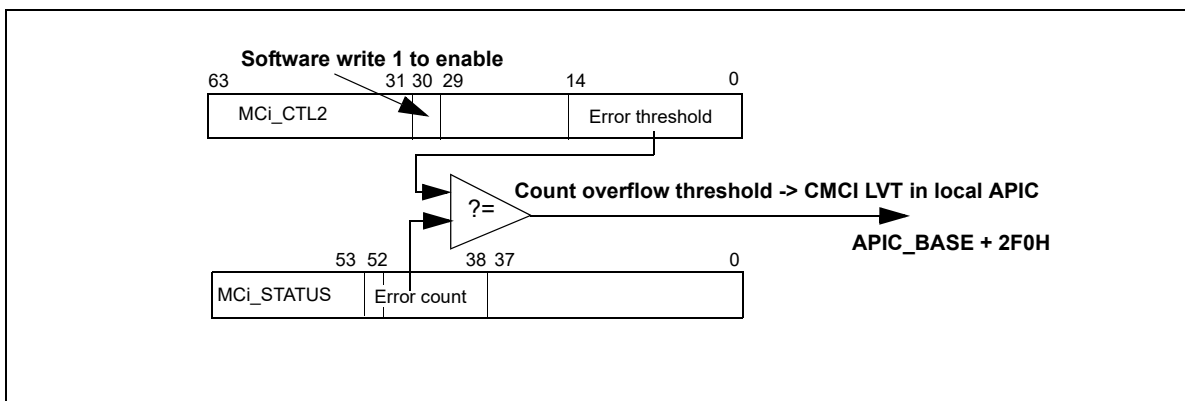


Figure 15-10. CMCI Behavior

CMCI interrupt delivery is configured by writing to the LVT CMCI register entry in the local APIC register space at default address of `APIC_BASE + 2F0H`. A CMCI interrupt can be delivered to more than one logical processors if multiple logical processors are affected by the associated MC errors. For example, if a corrected bit error in a cache shared by two logical processors caused a CMCI, the interrupt will be delivered to both logical processors sharing that microarchitectural sub-system. Similarly, package level errors may cause CMCI to be delivered to all logical processors within the package. However, system level errors will not be handled by CMCI.

See Section 10.5.1, “Local Vector Table” for details regarding the LVT CMCI register.

## 15.5.2 System Software Recommendation for Managing CMCI and Machine Check Resources

System software must enable and manage CMCI, set up interrupt handlers to service CMCI interrupts delivered to affected logical processors, program CMCI LVT entry, and query machine check banks that are shared by more than one logical processors.

This section describes techniques system software can implement to manage CMCI initialization, service CMCI interrupts in an efficient manner to minimize contentions to access shared MSR resources.

### 15.5.2.1 CMCI Initialization

Although a CMCI interrupt may be delivered to more than one logical processors depending on the nature of the corrected MC error, only one instance of the interrupt service routine needs to perform the necessary service and make queries to the machine-check banks. The following steps describes a technique that limits the amount of work the system has to do in response to a CMCI.

- To provide maximum flexibility, system software should define per-thread data structure for each logical processor to allow equal-opportunity and efficient response to interrupt delivery. Specifically, the per-thread data structure should include a set of per-bank fields to track which machine check bank it needs to access in response to a delivered CMCI interrupt. The number of banks that needs to be tracked is determined by `IA32_MCG_CAP[7:0]`.
- Initialization of per-thread data structure. The initialization of per-thread data structure must be done serially on each logical processor in the system. The sequencing order to start the per-thread initialization between different logical processor is arbitrary. But it must observe the following specific detail to satisfy the shared nature of specific MSR resources:
  - a. Each thread initializes its data structure to indicate that it does not own any MC bank registers.
  - b. Each thread examines `IA32_MCi_CTL2[30]` indicator for each bank to determine if another thread has already claimed ownership of that bank.
    - If `IA32_MCi_CTL2[30]` had been set by another thread. This thread can not own bank *i* and should proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
    - If `IA32_MCi_CTL2[30] = 0`, proceed to step c.
  - c. Check whether writing a 1 into `IA32_MCi_CTL2[30]` can return with 1 on a subsequent read to determine this bank can support CMCI.
    - If `IA32_MCi_CTL2[30] = 0`, this bank does not support CMCI. This thread can not own bank *i* and should proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
    - If `IA32_MCi_CTL2[30] = 1`, modify the per-thread data structure to indicate this thread claims ownership to the MC bank; proceed to initialize the error threshold count (bits 15:0) of that bank as described in Chapter 15, “CMCI Threshold Management”. Then proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
- After the thread has examined all of the machine check banks, it sees if it owns any MC banks to service CMCI. If any bank has been claimed by this thread:
  - Ensure that the CMCI interrupt handler has been set up as described in Chapter 15, “CMCI Interrupt Handler”.
  - Initialize the CMCI LVT entry, as described in Section 15.5.1, “CMCI Local APIC Interface”.



- Log and clear all of IA32\_MCi\_Status registers for the banks that this thread owns. This will allow new errors to be logged.

### 15.5.2.2 CMCI Threshold Management

The Corrected MC error threshold field, IA32\_MCi\_CTL2[15:0], is architecturally defined. Specifically, all these bits are writable by software, but different processor implementations may choose to implement less than 15 bits as threshold for the overflow comparison with IA32\_MCi\_STATUS[52:38]. The following describes techniques that software can manage CMCI threshold to be compatible with changes in implementation characteristics:

- Software can set the initial threshold value to 1 by writing 1 to IA32\_MCi\_CTL2[15:0]. This will cause overflow condition on every corrected MC error and generates a CMCI interrupt.
- To increase the threshold and reduce the frequency of CMCI servicing:
  - a. Find the maximum threshold value a given processor implementation supports. The steps are:
    - Write 7FFFH to IA32\_MCi\_CTL2[15:0],
    - Read back IA32\_MCi\_CTL2[15:0], the lower 15 bits (14:0) is the maximum threshold supported by the processor.
  - b. Increase the threshold to a value below the maximum value discovered using step a.

### 15.5.2.3 CMCI Interrupt Handler

The following describes techniques system software may consider to implement a CMCI service routine:

- The service routine examines its private per-thread data structure to check which set of MC banks it has ownership. If the thread does not have ownership of a given MC bank, proceed to the next MC bank. Ownership is determined at initialization time which is described in Section [Cross Reference to 14.5.2.1].

If the thread had claimed ownership to an MC bank, this technique will allow each logical processors to handle corrected MC errors independently and requires no synchronization to access shared MSR resources. Consult Example 15-5 for guidelines on logging when processing CMCI.

## 15.6 RECOVERY OF UNCORRECTED RECOVERABLE (UCR) ERRORS

Recovery of uncorrected recoverable machine check errors is an enhancement in machine-check architecture. The first processor that supports this feature is 45 nm Intel 64 processor on which CPUID reports DisplayFamily\_DisplayModel as 06H\_2EH (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). This allow system software to perform recovery action on certain class of uncorrected errors and continue execution.

### 15.6.1 Detection of Software Error Recovery Support

Software must use bit 24 of IA32\_MCG\_CAP (MCG\_SER\_P) to detect the presence of software error recovery support (see Figure 15-2). When IA32\_MCG\_CAP[24] is set, this indicates that the processor supports software error recovery. When this bit is clear, this indicates that there is no support for error recovery from the processor and the primary responsibility of the machine check handler is logging the machine check error information and shutting down the system.

The new class of architectural MCA errors from which system software can attempt recovery is called Uncorrected Recoverable (UCR) Errors. UCR errors are uncorrected errors that have been detected and signaled but have not corrupted the processor context. For certain UCR errors, this means that once system software has performed a certain recovery action, it is possible to continue execution on this processor. UCR error reporting provides an error containment mechanism for data poisoning. The machine check handler will use the error log information from the error reporting registers to analyze and implement specific error recovery actions for UCR errors.



## 15.6.2 UCR Error Reporting and Logging

IA32\_MCi\_STATUS MSR is used for reporting UCR errors and existing corrected or uncorrected errors. The definitions of IA32\_MCi\_STATUS, including bit fields to identify UCR errors, is shown in Figure 15-6. UCR errors can be signaled through either the corrected machine check interrupt (CMCI) or machine check exception (MCE) path depending on the type of the UCR error.

When IA32\_MCG\_CAP[24] is set, a UCR error is indicated by the following bit settings in the IA32\_MCi\_STATUS register:

- Valid (bit 63) = 1
- UC (bit 61) = 1
- PCC (bit 57) = 0

Additional information from the IA32\_MCi\_MISC and the IA32\_MCi\_ADDR registers for the UCR error are available when the ADDR\_V and the MISC\_V flags in the IA32\_MCi\_STATUS register are set (see Section 15.3.2.4). The MCA error code field of the IA32\_MCi\_STATUS register indicates the type of UCR error. System software can interpret the MCA error code field to analyze and identify the necessary recovery action for the given UCR error.

In addition, the IA32\_MCi\_STATUS register bit fields, bits 56:55, are defined (see Figure 15-6) to provide additional information to help system software to properly identify the necessary recovery action for the UCR error:

- S (Signaling) flag, bit 56 - Indicates (when set) that a machine check exception was generated for the UCR error reported in this MC bank and system software needs to check the AR flag and the MCA error code fields in the IA32\_MCi\_STATUS register to identify the necessary recovery action for this error. When the S flag in the IA32\_MCi\_STATUS register is clear, this UCR error was not signaled via a machine check exception and instead was reported as a corrected machine check (CMC). System software is not required to take any recovery action when the S flag in the IA32\_MCi\_STATUS register is clear.
- AR (Action Required) flag, bit 55 - Indicates (when set) that MCA error code specific recovery action must be performed by system software at the time this error was signaled. This recovery action must be completed successfully before any additional work is scheduled for this processor. When the RIP\_V flag in the IA32\_MCG\_STATUS is clear, an alternative execution stream needs to be provided; when the MCA error code specific recovery specific recovery action cannot be successfully completed, system software must shut down the system. When the AR flag in the IA32\_MCi\_STATUS register is clear, system software may still take MCA error code specific recovery action but this is optional; system software can safely resume program execution at the instruction pointer saved on the stack from the machine check exception when the RIP\_V flag in the IA32\_MCG\_STATUS register is set.

Both the S and the AR flags in the IA32\_MCi\_STATUS register are defined to be sticky bits, which mean that once set, the processor does not clear them. Only software and good power-on reset can clear the S and the AR-flags. Both the S and the AR flags are only set when the processor reports the UCR errors (MCG\_CAP[24] is set).

## 15.6.3 UCR Error Classification

With the S and AR flag encoding in the IA32\_MCi\_STATUS register, UCR errors can be classified as:

- Uncorrected no action required (UCNA) - is a UCR error that is not signaled via a machine check exception and, instead, is reported to system software as a corrected machine check error. UCNA errors indicate that some data in the system is corrupted, but the data has not been consumed and the processor state is valid and you may continue execution on this processor. UCNA errors require no action from system software to continue execution. A UCNA error is indicated with UC=1, PCC=0, S=0 and AR=0 in the IA32\_MCi\_STATUS register.
- Software recoverable action optional (SRAO) - a UCR error is signaled either via a machine check exception or CMCI. System software recovery action is optional and not required to continue execution from this machine check exception. SRAO errors indicate that some data in the system is corrupt, but the data has not been consumed and the processor state is valid. SRAO errors provide the additional error information for system software to perform a recovery action. An SRAO error when signaled as a machine check is indicated with UC=1, PCC=0, S=1, EN=1 and AR=0 in the IA32\_MCi\_STATUS register. In cases when SRAO is signaled via CMCI the error signature is indicated via UC=1, PCC=0, S=0. Recovery actions for SRAO errors are MCA error code specific. The MISC\_V and the ADDR\_V flags in the IA32\_MCi\_STATUS register are set when the additional error information is available from the IA32\_MCi\_MISC and the IA32\_MCi\_ADDR registers. System software needs to inspect the MCA error code fields in the IA32\_MCi\_STATUS register to identify the specific recovery

action for a given SRAO error. If MISCV and ADDRIV are not set, it is recommended that no system software error recovery be performed however, system software can resume execution.

- Software recoverable action required (SRAR) - a UCR error that requires system software to take a recovery action on this processor before scheduling another stream of execution on this processor. SRAR errors indicate that the error was detected and raised at the point of the consumption in the execution flow. An SRAR error is indicated with UC=1, PCC=0, S=1, EN=1 and AR=1 in the IA32\_MCi\_STATUS register. Recovery actions are MCA error code specific. The MISCV and the ADDRIV flags in the IA32\_MCi\_STATUS register are set when the additional error information is available from the IA32\_MCi\_MISC and the IA32\_MCi\_ADDR registers. System software needs to inspect the MCA error code fields in the IA32\_MCi\_STATUS register to identify the specific recovery action for a given SRAR error. If MISCV and ADDRIV are not set, it is recommended that system software shutdown the system.

Table 15-7 summarizes UCR, corrected, and uncorrected errors.

**Table 15-7. MC Error Classifications**

Type of Error <sup>1</sup>	UC	EN	PCC	S	AR	Signaling	Software Action	Example
Uncorrected Error (UC)	1	1	1	x	x	MCE	If EN=1, reset the system, else log and OK to keep the system running.	
SRAR	1	1	0	1	1	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, must bugcheck. If OVER=1, reset system, else take specific recovery action.	Cache to processor load error.
SRAO	1	x <sup>2</sup>	0	x <sup>2</sup>	0	MCE/CMC	For known MCACOD, take specific recovery action; For unknown MCACOD, OK to keep the system running.	Patrol scrub and explicit writeback poison errors.
UCNA	1	x	0	0	0	CMC	Log the error and Ok to keep the system running.	Poison detection error.
Corrected Error (CE)	0	x	x	x	x	CMC	Log the error and no corrective action required.	ECC in caches and memory.

**NOTES:**

1. SRAR, SRAO and UCNA errors are supported by the processor only when IA32\_MCG\_CAP[24] (MCG\_SER\_P) is set.
2. EN=1, S=1 when signaled via MCE. EN=x, S=0 when signaled via CMC.

### 15.6.4 UCR Error Overwrite Rules

In general, the overwrite rules are as follows:

- UCR errors will overwrite corrected errors.
- Uncorrected (PCC=1) errors overwrite UCR (PCC=0) errors.
- UCR errors are not written over previous UCR errors.
- Corrected errors do not write over previous UCR errors.

Regardless of whether the 1st error is retained or the 2nd error is overwritten over the 1st error, the OVER flag in the IA32\_MCi\_STATUS register will be set to indicate an overflow condition. As the S flag and AR flag in the IA32\_MCi\_STATUS register are defined to be sticky flags, a second event cannot clear these 2 flags once set, however the MC bank information may be filled in for the 2nd error. The table below shows the overwrite rules and how to treat a second error if the first event is already logged in a MC bank along with the resulting bit setting of the UC, PCC, and AR flags in the IA32\_MCi\_STATUS register. As UCNA and SRAO errors do not require recovery action from system software to continue program execution, a system reset by system software is not required unless the AR flag or PCC flag is set for the UCR overflow case (OVER=1, VAL=1, UC=1, PCC=0).

Table 15-8 lists overwrite rules for uncorrected errors, corrected errors, and uncorrected recoverable errors.

**Table 15-8. Overwrite Rules for UC, CE, and UCR Errors**

First Event	Second Event	UC	PCC	S	AR	MCA Bank	Reset System
CE	UCR	1	0	0 if UCNA, else 1	1 if SRAR, else 0	second	yes, if AR=1
UCR	CE	1	0	0 if UCNA, else 1	1 if SRAR, else 0	first	yes, if AR=1
UCNA	UCNA	1	0	0	0	first	no
UCNA	SRAO	1	0	1	0	first	no
UCNA	SRAR	1	0	1	1	first	yes
SRAO	UCNA	1	0	1	0	first	no
SRAO	SRAO	1	0	1	0	first	no
SRAO	SRAR	1	0	1	1	first	yes
SRAR	UCNA	1	0	1	1	first	yes
SRAR	SRAO	1	0	1	1	first	yes
SRAR	SRAR	1	0	1	1	first	yes
UCR	UC	1	1	undefined	undefined	second	yes
UC	UCR	1	1	undefined	undefined	first	yes

## 15.7 MACHINE-CHECK AVAILABILITY

The machine-check architecture and machine-check exception (#MC) are model-specific features. Software can execute the CPUID instruction to determine whether a processor implements these features. Following the execution of the CPUID instruction, the settings of the MCA flag (bit 14) and MCE flag (bit 7) in EDX indicate whether the processor implements the machine-check architecture and machine-check exception.

## 15.8 MACHINE-CHECK INITIALIZATION

To use the processors machine-check architecture, software must initialize the processor to activate the machine-check exception and the error-reporting mechanism.

Example 15-1 gives pseudocode for performing this initialization. This pseudocode checks for the existence of the machine-check architecture and exception; it then enables machine-check exception and the error-reporting register banks. The pseudocode shown is compatible with the Pentium 4, Intel Xeon, Intel Atom, P6 family, and Pentium processors.

Following power up or power cycling, IA32\_MCi\_STATUS registers are not guaranteed to have valid data until after they are initially cleared to zero by software (as shown in the initialization pseudocode in Example 15-1). In addition, when using P6 family processors, software must set MCI\_STATUS registers to zero when doing a soft-reset.

### Example 15-1. Machine-Check Initialization Pseudocode

Check CPUID Feature Flags for MCE and MCA support

IF CPU supports MCE

THEN

IF CPU supports MCA

THEN

IF (IA32\_MCG\_CAP.MCG\_CTL\_P = 1)

(\* IA32\_MCG\_CTL register is present \*)

THEN

IA32\_MCG\_CTL ← FFFFFFFFFFFFFFFFH;

(\* enables all MCA features \*)

FI

IF (IA32\_MCG\_CAP.MCG\_LMCE\_P = 1 and IA32\_FEATURE\_CONTROL.LOCK = 1 and IA32\_FEATURE\_CONTROL.LMCE\_ON= 1)

```
(* IA32_MCG_EXT_CTL register is present and platform has enabled LMCE to permit system software to use LMCE *)
THEN
  IA32_MCG_EXT_CTL ← IA32_MCG_EXT_CTL | 01H;
  (* System software enables LMCE capability for hardware to signal MCE to a single logical processor*)
FI
```

```
(* Determine number of error-reporting banks supported *)
COUNT ← IA32_MCG_CAP.Count;
MAX_BANK_NUMBER ← COUNT - 1;
```

```
IF (Processor Family is 6H and Processor EXTMODEL:MODEL is less than 1AH)
THEN
```

```
  (* Enable logging of all errors except for MCO_CTL register *)
  FOR error-reporting banks (1 through MAX_BANK_NUMBER)
  DO
    IA32_MCi_CTL ← 0FFFFFFFFFFFFFFFH;
  OD
```

```
ELSE
  (* Enable logging of all errors including MCO_CTL register *)
  FOR error-reporting banks (0 through MAX_BANK_NUMBER)
  DO
    IA32_MCi_CTL ← 0FFFFFFFFFFFFFFFH;
  OD
FI
```

```
(* BIOS clears all errors only on power-on reset *)
```

```
IF (BIOS detects Power-on reset)
THEN
  FOR error-reporting banks (0 through MAX_BANK_NUMBER)
  DO
    IA32_MCi_STATUS ← 0;
  OD
ELSE
  FOR error-reporting banks (0 through MAX_BANK_NUMBER)
  DO
    (Optional for BIOS and OS) Log valid errors
    (OS only) IA32_MCi_STATUS ← 0;
  OD
```

```
FI
```

```
FI
```

Setup the Machine Check Exception (#MC) handler for vector 18 in IDT

Set the MCE bit (bit 6) in CR4 register to enable Machine-Check Exceptions

```
FI
```

## 15.9 INTERPRETING THE MCA ERROR CODES

When the processor detects a machine-check error condition, it writes a 16-bit error code to the MCA error code field of one of the IA32\_MCi\_STATUS registers and sets the VAL (valid) flag in that register. The processor may also write a 16-bit model-specific error code in the IA32\_MCi\_STATUS register depending on the implementation of the machine-check architecture of the processor.

The MCA error codes are architecturally defined for Intel 64 and IA-32 processors. To determine the cause of a machine-check exception, the machine-check exception handler must read the VAL flag for each IA32\_MCi\_STATUS register. If the flag is set, the machine check-exception handler must then read the MCA error code field of the register. It is the encoding of the MCA error code field [15:0] that determines the type of error being reported and not the register bank reporting it.

There are two types of MCA error codes: simple error codes and compound error codes.

## 15.9.1 Simple Error Codes

Table 15-9 shows the simple error codes. These unique codes indicate global error information.

**Table 15-9. IA32\_MCi\_Status [15:0] Simple Error Code Encoding**

Error Code	Binary Encoding	Meaning
No Error	0000 0000 0000 0000	No error has been reported to this bank of error-reporting registers.
Unclassified	0000 0000 0000 0001	This error has not been classified into the MCA error classes.
Microcode ROM Parity Error	0000 0000 0000 0010	Parity error in internal microcode ROM
External Error	0000 0000 0000 0011	The BINIT# from another processor caused this processor to enter machine check. <sup>1</sup>
FRC Error	0000 0000 0000 0100	FRC (functional redundancy check) master/slave error
Internal Parity Error	0000 0000 0000 0101	Internal parity error.
SMM Handler Code Access Violation	0000 0000 0000 0110	An attempt was made by the SMM Handler to execute outside the ranges specified by SMRR.
Internal Timer Error	0000 0100 0000 0000	Internal timer error.
I/O Error	0000 1110 0000 1011	generic I/O error.
Internal Unclassified	0000 01xx xxxx xxxx	Internal unclassified errors. <sup>2</sup>

### NOTES:

1. BINIT# assertion will cause a machine check exception if the processor (or any processor on the same external bus) has BINIT# observation enabled during power-on configuration (hardware strapping) and if machine check exceptions are enabled (by setting CR4.MCE = 1).
2. At least one X must equal one. Internal unclassified errors have not been classified.

## 15.9.2 Compound Error Codes

Compound error codes describe errors related to the TLBs, memory, caches, bus and interconnect logic, and internal timer. A set of sub-fields is common to all of compound errors. These sub-fields describe the type of access, level in the cache hierarchy, and type of request. Table 15-10 shows the general form of the compound error codes.

**Table 15-10. IA32\_MCi\_Status [15:0] Compound Error Code Encoding**

Type	Form	Interpretation
Generic Cache Hierarchy	000F 0000 0000 11LL	Generic cache hierarchy error
TLB Errors	000F 0000 0001 TTLL	{TT}TLB{LL}_ERR
Memory Controller Errors	000F 0000 1MMM CCCC	{MMM}_CHANNEL{CCCC}_ERR
Cache Hierarchy Errors	000F 0001 RRRR TTLL	{TT}CACHE{LL}_{RRRR}_ERR
Bus and Interconnect Errors	000F 1PPT RRRR IILL	BUS{LL}_{PP}_{RRRR}_{II}_{T}_ERR

The “Interpretation” column in the table indicates the name of a compound error. The name is constructed by substituting mnemonics for the sub-field names given within curly braces. For example, the error code ICACHEL1\_RD\_ERR is constructed from the form:

```
{TT}CACHE{LL}_{RRRR}_ERR,
where {TT} is replaced by I, {LL} is replaced by L1, and {RRRR} is replaced by RD.
```

For more information on the “Form” and “Interpretation” columns, see Sections Section 15.9.2.1, “Correction Report Filtering (F) Bit” through Section 15.9.2.5, “Bus and Interconnect Errors”.

### 15.9.2.1 Correction Report Filtering (F) Bit

Starting with Intel Core Duo processors, bit 12 in the “Form” column in Table 15-10 is used to indicate that a particular posting to a log may be the last posting for corrections in that line/entry, at least for some time:

- 0 in bit 12 indicates “normal” filtering (original P6/Pentium4/Atom/Xeon processor meaning).
- 1 in bit 12 indicates “corrected” filtering (filtering is activated for the line/entry in the posting). Filtering means that some or all of the subsequent corrections to this entry (in this structure) will not be posted. The enhanced error reporting introduced with the Intel Core Duo processors is based on tracking the lines affected by repeated corrections (see Section 15.4, “Enhanced Cache Error reporting”). This capability is indicated by IA32\_MCG\_CAP[11]. Only the first few correction events for a line are posted; subsequent redundant correction events to the same line are not posted. Uncorrected events are always posted.

The behavior of error filtering after crossing the yellow threshold is model-specific. Filtering has meaning only for corrected errors (UC=0 in IA32\_MCi\_STATUS MSR). System software must ignore filtering bit (12) for uncorrected errors.

### 15.9.2.2 Transaction Type (TT) Sub-Field

The 2-bit TT sub-field (Table 15-11) indicates the type of transaction (data, instruction, or generic). The sub-field applies to the TLB, cache, and interconnect error conditions. Note that interconnect error conditions are primarily associated with P6 family and Pentium processors, which utilize an external APIC bus separate from the system bus. The generic type is reported when the processor cannot determine the transaction type.

**Table 15-11. Encoding for TT (Transaction Type) Sub-Field**

Transaction Type	Mnemonic	Binary Encoding
Instruction	I	00
Data	D	01
Generic	G	10

### 15.9.2.3 Level (LL) Sub-Field

The 2-bit LL sub-field (see Table 15-12) indicates the level in the memory hierarchy where the error occurred (level 0, level 1, level 2, or generic). The LL sub-field also applies to the TLB, cache, and interconnect error conditions. The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors support two levels in the cache hierarchy and one level in the TLBs. Again, the generic type is reported when the processor cannot determine the hierarchy level.

**Table 15-12. Level Encoding for LL (Memory Hierarchy Level) Sub-Field**

Hierarchy Level	Mnemonic	Binary Encoding
Level 0	L0	00
Level 1	L1	01
Level 2	L2	10
Generic	LG	11

### 15.9.2.4 Request (RRRR) Sub-Field

The 4-bit RRRR sub-field (see Table 15-13) indicates the type of action associated with the error. Actions include read and write operations, prefetches, cache evictions, and snoops. Generic error is returned when the type of error cannot be determined. Generic read and generic write are returned when the processor cannot determine the type of instruction or data request that caused the error. Eviction and snoop requests apply only to the caches. All of the other requests apply to TLBs, caches and interconnects.

**Table 15-13. Encoding of Request (RRRR) Sub-Field**

Request Type	Mnemonic	Binary Encoding
Generic Error	ERR	0000

**Table 15-13. Encoding of Request (RRRR) Sub-Field (Contd.)**

Generic Read	RD	0001
Generic Write	WR	0010
Data Read	DRD	0011
Data Write	DWR	0100
Instruction Fetch	IRD	0101
Prefetch	PREFETCH	0110
Eviction	EVICT	0111
Snoop	SNOOP	1000

### 15.9.2.5 Bus and Interconnect Errors

The bus and interconnect errors are defined with the 2-bit PP (participation), 1-bit T (time-out), and 2-bit II (memory or I/O) sub-fields, in addition to the LL and RRRR sub-fields (see Table 15-14). The bus error conditions are implementation dependent and related to the type of bus implemented by the processor. Likewise, the interconnect error conditions are predicated on a specific implementation-dependent interconnect model that describes the connections between the different levels of the storage hierarchy. The type of bus is implementation dependent, and as such is not specified in this document. A bus or interconnect transaction consists of a request involving an address and a response.

**Table 15-14. Encodings of PP, T, and II Sub-Fields**

Sub-Field	Transaction	Mnemonic	Binary Encoding
PP (Participation)	Local processor* originated request	SRC	00
	Local processor* responded to request	RES	01
	Local processor* observed error as third party	OBS	10
	Generic		11
T (Time-out)	Request timed out	TIMEOUT	1
	Request did not time out	NOTIMEOUT	0
II (Memory or I/O)	Memory Access	M	00
	Reserved		01
	I/O	IO	10
	Other transaction		11

**NOTE:**

\* Local processor differentiates the processor reporting the error from other system components (including the APIC, other processors, etc.).

### 15.9.2.6 Memory Controller Errors

The memory controller errors are defined with the 3-bit MMM (memory transaction type), and 4-bit CCCC (channel) sub-fields. The encodings for MMM and CCCC are defined in Table 15-15.

**Table 15-15. Encodings of MMM and CCCC Sub-Fields**

Sub-Field	Transaction	Mnemonic	Binary Encoding
MMM	Generic undefined request	GEN	000
	Memory read error	RD	001
	Memory write error	WR	010
	Address/Command Error	AC	011
	Memory Scrubbing Error	MS	100
	Reserved		101-111

**Table 15-15. Encodings of MMM and CCCC Sub-Fields (Contd.)**

CCCC	Channel number	CHN	0000-1110
	Channel not specified		1111

### 15.9.3 Architecturally Defined UCR Errors

Software recoverable compound error code are defined in this section.

#### 15.9.3.1 Architecturally Defined SRAO Errors

The following two SRAO errors are architecturally defined.

- UCR Errors detected by memory controller scrubbing; and
- UCR Errors detected during L3 cache (L3) explicit writebacks.

The MCA error code encodings for these two architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 15-10). Their values and compound encoding format are given in Table 15-16.

**Table 15-16. MCA Compound Error Code Encoding for SRAO Errors**

Type	MCACOD Value	MCA Error Code Encoding <sup>1</sup>
Memory Scrubbing	COH - CFH	0000_0000_1100_CCCC 000F 0000 1MMM CCCC (Memory Controller Error), where Memory subfield MMM = 100B (memory scrubbing) Channel subfield CCCC = channel # or generic
L3 Explicit Writeback	17AH	0000_0001_0111_1010 000F 0001 RRRR TTLL (Cache Hierarchy Error) where Request subfields RRRR = 0111B (Eviction) Transaction Type subfields TT = 10B (Generic) Level subfields LL = 10B

**NOTES:**

1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error must be ignored.

Table 15-17 lists values of relevant bit fields of IA32\_MCi\_STATUS for architecturally defined SRAO errors.

**Table 15-17. IA32\_MCi\_STATUS Values for SRAO Errors**

SRAO Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR	MCACOD
Memory Scrubbing	1	0	1	x <sup>1</sup>	1	1	0	x <sup>1</sup>	0	COH-CFH
L3 Explicit Writeback	1	0	1	x <sup>1</sup>	1	1	0	x <sup>1</sup>	0	17AH

**NOTES:**

1. When signaled as MCE, EN=1 and S=1. If error was signaled via CMC, then EN=x, and S=0.

For both the memory scrubbing and L3 explicit writeback errors, the ADDRv and MISCV flags in the IA32\_MCi\_STATUS register are set to indicate that the offending physical address information is available from the IA32\_MCi\_MISC and the IA32\_MCi\_ADDR registers. For the memory scrubbing and L3 explicit writeback errors, the address mode in the IA32\_MCi\_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32\_MCi\_MISC register should indicate the lowest valid address bit in the address information provided from the IA32\_MCi\_ADDR register.

MCE signal is broadcast to all logical processors as outlined in Section 15.10.4.1. If LMCE is supported and enabled, some errors (not limited to UCR errors) may be delivered to only a single logical processor. System software should consult IA32\_MCG\_STATUS.LMCE\_S to determine if the MCE signaled is only to this logical processor.



IA32\_MCi\_STATUS banks can be shared by logical processors within a core or within the same package. So several logical processors may find an SRAO error in the shared IA32\_MCi\_STATUS bank but other processors do not find it in any of the IA32\_MCi\_STATUS banks. Table 15-18 shows the RIPV and EIPV flag indication in the IA32\_MCG\_STATUS register for the memory scrubbing and L3 explicit writeback errors on both the reporting and non-reporting logical processors.

**Table 15-18. IA32\_MCG\_STATUS Flag Indication for SRAO Errors**

SRAO Type	Reporting Logical Processors		Non-reporting Logical Processors	
	RIPV	EIPV	RIPV	EIPV
Memory Scrubbing	1	0	1	0
L3 Explicit Writeback	1	0	1	0

### 15.9.3.2 Architecturally Defined SRAR Errors

The following two SRAR errors are architecturally defined.

- UCR Errors detected on data load; and
- UCR Errors detected on instruction fetch.

The MCA error code encodings for these two architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 15-10). Their values and compound encoding format are given in Table 15-19.

**Table 15-19. MCA Compound Error Code Encoding for SRAR Errors**

Type	MCACOD Value	MCA Error Code Encoding <sup>1</sup>
Data Load	134H	0000_0001_0011_0100 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0011B (Data Load) Transaction Type subfield TT= 01B (Data) Level subfield LL = 00B (Level 0)
Instruction Fetch	150H	0000_0001_0101_0000 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0101B (Instruction Fetch) Transaction Type subfield TT= 00B (Instruction) Level subfield LL = 00B (Level 0)

#### NOTES:

1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error must be ignored.

Table 15-20 lists values of relevant bit fields of IA32\_MCi\_STATUS for architecturally defined SRAR errors.

**Table 15-20. IA32\_MCi\_STATUS Values for SRAR Errors**

SRAR Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR	MCACOD
Data Load	1	0	1	1	1	1	0	1	1	134H
Instruction Fetch	1	0	1	1	1	1	0	1	1	150H

For both the data load and instruction fetch errors, the ADDRv and MISCV flags in the IA32\_MCi\_STATUS register are set to indicate that the offending physical address information is available from the IA32\_MCi\_MISC and the IA32\_MCi\_ADDR registers. For the memory scrubbing and L3 explicit writeback errors, the address mode in the IA32\_MCi\_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32\_MCi\_MISC register should indicate the lowest valid address bit in the address information provided from the IA32\_MCi\_ADDR register.

MCE signal is broadcast to all logical processors on the system on which the UCR errors are supported, except when the processor supports LMCE and LMCE is enabled by system software (see Section 15.3.1.5). The

IA32\_MCG\_STATUS MSR allows system software to distinguish the affected logical processor of an SRAR error amongst logical processors that observed SRAR via MCI\_STATUS bank.

Table 15-21 shows the RIPV and EIPV flag indication in the IA32\_MCG\_STATUS register for the data load and instruction fetch errors on both the reporting and non-reporting logical processors. The recoverable SRAR error reported by a processor may be continuable, where the system software can interpret the context of continuable as follows: the error was isolated, contained. If software can rectify the error condition in the current instruction stream, the execution context on that logical processor can be continued without loss of information.

**Table 15-21. IA32\_MCG\_STATUS Flag Indication for SRAR Errors**

SRAR Type	Affected Logical Processor			Non-Affected Logical Processors		
	RIPV	EIPV	Continuable	RIPV	EIPV	Continuable
Recoverable-continuable	1	1	Yes <sup>1</sup>	1	0	Yes
Recoverable-not-continuable	0	x	No			

**NOTES:**

1. see the definition of the context of “continuable” above and additional detail below.

**SRAR Error And Affected Logical Processors**

The affected logical processor is the one that has detected and raised an SRAR error at the point of the consumption in the execution flow. The affected logical processor should find the Data Load or the Instruction Fetch error information in the IA32\_MCI\_STATUS register that is reporting the SRAR error.

Table 15-21 list the actionable scenarios that system software can respond to an SRAR error on an affected logical processor according to RIPV and EIPV values:

- Recoverable-Continuable SRAR Error (RIPV=1, EIPV=1):  
 For Recoverable-Continuable SRAR errors, the affected logical processor should find that both the IA32\_MCG\_STATUS.RIPV and the IA32\_MCG\_STATUS.EIPV flags are set, indicating that system software may be able to restart execution from the interrupted context if it is able to rectify the error condition. If system software cannot rectify the error condition then it must treat the error as a recoverable error where restarting execution with the interrupted context is not possible. Restarting without rectifying the error condition will result in most cases with another SRAR error on the same instruction.
- Recoverable-not-continuable SRAR Error (RIPV=0, EIPV=x):  
 For Recoverable-not-continuable errors, the affected logical processor should find that either
  - IA32\_MCG\_STATUS.RIPV= 0, IA32\_MCG\_STATUS.EIPV=1, or
  - IA32\_MCG\_STATUS.RIPV= 0, IA32\_MCG\_STATUS.EIPV=0.
 In either case, this indicates that the error is detected at the instruction pointer saved on the stack for this machine check exception and restarting execution with the interrupted context is not possible. System software may take the following recovery actions for the affected logical processor:
  - The current executing thread cannot be continued. System software must terminate the interrupted stream of execution and provide a new stream of execution on return from the machine check handler for the affected logical processor.

**SRAR Error And Non-Affected Logical Processors**

The logical processors that observed but not affected by an SRAR error should find that the RIPV flag in the IA32\_MCG\_STATUS register is set and the EIPV flag in the IA32\_MCG\_STATUS register is cleared, indicating that it is safe to restart the execution at the instruction saved on the stack for the machine check exception on these processors after the recovery action is successfully taken by system software.

## 15.9.4 Multiple MCA Errors

When multiple MCA errors are detected within a certain detection window, the processor may aggregate the reporting of these errors together as a single event, i.e. a single machine exception condition. If this occurs, system software may find multiple MCA errors logged in different MC banks on one logical processor or find multiple MCA errors logged across different processors for a single machine check broadcast event. In order to handle multiple UCR errors reported from a single machine check event and possibly recover from multiple errors, system software may consider the following:

- Whether it can recover from multiple errors is determined by the most severe error reported on the system. If the most severe error is found to be an unrecoverable error (VAL=1, UC=1, PCC=1 and EN=1) after system software examines the MC banks of all processors to which the MCA signal is broadcast, recovery from the multiple errors is not possible and system software needs to reset the system.
- When multiple recoverable errors are reported and no other fatal condition (e.g. overflowed condition for SRAR error) is found for the reported recoverable errors, it is possible for system software to recover from the multiple recoverable errors by taking necessary recovery action for each individual recoverable error. However, system software can no longer expect one to one relationship with the error information recorded in the IA32\_MCi\_STATUS register and the states of the RIPV and EIPV flags in the IA32\_MCG\_STATUS register as the states of the RIPV and the EIPV flags in the IA32\_MCG\_STATUS register may indicate the information for the most severe error recorded on the processor. System software is required to use the RIPV flag indication in the IA32\_MCG\_STATUS register to make a final decision of recoverability of the errors and find the restart-ability requirement after examining each IA32\_MCi\_STATUS register error information in the MC banks.

In certain cases where system software observes more than one SRAR error logged for a single logical processor, it can no longer rely on affected threads as specified in Table 15-20 above. System software is recommended to reset the system if this condition is observed.

## 15.9.5 Machine-Check Error Codes Interpretation

Chapter 16, "Interpreting Machine-Check Error Codes," provides information on interpreting the MCA error code, model-specific error code, and other information error code fields. For P6 family processors, information has been included on decoding external bus errors. For Pentium 4 and Intel Xeon processors; information is included on external bus, internal timer and cache hierarchy errors.

## 15.10 GUIDELINES FOR WRITING MACHINE-CHECK SOFTWARE

The machine-check architecture and error logging can be used in three different ways:

- To detect machine errors during normal instruction execution, using the machine-check exception (#MC).
- To periodically check and log machine errors.
- To examine recoverable UCR errors, determine software recoverability and perform recovery actions via a machine-check exception handler or a corrected machine-check interrupt handler.

To use the machine-check exception, the operating system or executive software must provide a machine-check exception handler. This handler may need to be designed specifically for each family of processors.

A special program or utility is required to log machine errors.

Guidelines for writing a machine-check exception handler or a machine-error logging utility are given in the following sections.

### 15.10.1 Machine-Check Exception Handler

The machine-check exception (#MC) corresponds to vector 18. To service machine-check exceptions, a trap gate must be added to the IDT. The pointer in the trap gate must point to a machine-check exception handler. Two approaches can be taken to designing the exception handler:

1. The handler can merely log all the machine status and error information, then call a debugger or shut down the system.

2. The handler can analyze the reported error information and, in some cases, attempt to correct the error and restart the processor.

For Pentium 4, Intel Xeon, Intel Atom, P6 family, and Pentium processors; virtually all machine-check conditions cannot be corrected (they result in abort-type exceptions). The logging of status and error information is therefore a baseline implementation requirement.

When IA32\_MCG\_CAP[24] is clear, consider the following when writing a machine-check exception handler:

- To determine the nature of the error, the handler must read each of the error-reporting register banks. The count field in the IA32\_MCG\_CAP register gives number of register banks. The first register of register bank 0 is at address 400H.
- The VAL (valid) flag in each IA32\_MCi\_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and do not need to be checked.
- To write a portable exception handler, only the MCA error code field in the IA32\_MCi\_STATUS register should be checked. See Section 15.9, "Interpreting the MCA Error Codes," for information that can be used to write an algorithm to interpret this field.
- Correctable errors are corrected automatically by the processor. The UC flag in each IA32\_MCi\_STATUS register indicates whether the processor automatically corrected an error.
- The RIPV, PCC, and OVER flags in each IA32\_MCi\_STATUS register indicate whether recovery from the error is possible. If PCC or OVER are set, recovery is not possible. If RIPV is not set, program execution can not be restarted reliably. When recovery is not possible, the handler typically records the error information and signals an abort to the operating system.
- The RIPV flag in the IA32\_MCG\_STATUS register indicates whether the program can be restarted at the instruction indicated by the instruction pointer (the address of the instruction pushed on the stack when the exception was generated). If this flag is clear, the processor may still be able to be restarted (for debugging purposes) but not without loss of program continuity.
- For unrecoverable errors, the EIPV flag in the IA32\_MCG\_STATUS register indicates whether the instruction indicated by the instruction pointer pushed on the stack (when the exception was generated) is related to the error. If the flag is clear, the pushed instruction may not be related to the error.
- The MCIP flag in the IA32\_MCG\_STATUS register indicates whether a machine-check exception was generated. Before returning from the machine-check exception handler, software should clear this flag so that it can be used reliably by an error logging utility. The MCIP flag also detects recursion. The machine-check architecture does not support recursion. When the processor detects machine-check recursion, it enters the shutdown state.

Example 15-2 gives typical steps carried out by a machine-check exception handler.

**Example 15-2. Machine-Check Exception Handler Pseudocode**

```

IF CPU supports MCE
  THEN
    IF CPU supports MCA
      THEN
        call errorlogging routine; (* returns restartability *)
      FI;
    ELSE (* Pentium(R) processor compatible *)
      READ P5_MC_ADDR
      READ P5_MC_TYPE;
      report RESTARTABILITY to console;
    FI;
  IF error is not restartable
    THEN
      report RESTARTABILITY to console;
      abort system;
    FI;
  CLEAR MCIP flag in IA32_MCG_STATUS;

```

**15.10.2 Pentium Processor Machine-Check Exception Handling**

Machine-check exception handler on P6 family, Intel Atom and later processor families, should follow the guidelines described in Section 15.10.1 and Example 15-2 that check the processor's support of MCA.

**NOTE**

On processors that support MCA (CPUID.1.EDX.MCA = 1) reading the P5\_MC\_TYPE and P5\_MC\_ADDR registers may produce invalid data.

When machine-check exceptions are enabled for the Pentium processor (MCE flag is set in control register CR4), the machine-check exception handler uses the RDMSR instruction to read the error type from the P5\_MC\_TYPE register and the machine check address from the P5\_MC\_ADDR register. The handler then normally reports these register values to the system console before aborting execution (see Example 15-2).

**15.10.3 Logging Correctable Machine-Check Errors**

The error handling routine for servicing the machine-check exceptions is responsible for logging uncorrected errors.

If a machine-check error is correctable, the processor does not generate a machine-check exception for it. To detect correctable machine-check errors, a utility program must be written that reads each of the machine-check error-reporting register banks and logs the results in an accounting file or data structure. This utility can be implemented in either of the following ways.

- A system daemon that polls the register banks on an infrequent basis, such as hourly or daily.
- A user-initiated application that polls the register banks and records the exceptions. Here, the actual polling service is provided by an operating-system driver or through the system call interface.
- An interrupt service routine servicing CMCI can read the MC banks and log the error. Please refer to Section 15.10.4.2 for guidelines on logging correctable machine checks.

Example 15-3 gives pseudocode for an error logging utility.

**Example 15-3. Machine-Check Error Logging Pseudocode**

```

Assume that execution is restartable;
IF the processor supports MCA
  THEN
    FOR each bank of machine-check registers
      DO
        READ IA32_MCi_STATUS;
        IF VAL flag in IA32_MCi_STATUS = 1
          THEN
            IF ADDRV flag in IA32_MCi_STATUS = 1
              THEN READ IA32_MCi_ADDR;
            FI;
            IF MISCV flag in IA32_MCi_STATUS = 1
              THEN READ IA32_MCi_MISC;
            FI;
            IF MCIP flag in IA32_MCG_STATUS = 1
              (* Machine-check exception is in progress *)
              AND PCC flag in IA32_MCi_STATUS = 1
              OR RIPV flag in IA32_MCG_STATUS = 0
              (* execution is not restartable *)
              THEN
                RESTARTABILITY = FALSE;
                return RESTARTABILITY to calling procedure;
            FI;
            Save time-stamp counter and processor ID;
            Set IA32_MCi_STATUS to all 0s;
            Execute serializing instruction (i.e., CPUID);
          FI;
        FI;
      OD;
    FI;

```

If the processor supports the machine-check architecture, the utility reads through the banks of error-reporting registers looking for valid register entries. It then saves the values of the IA32\_MC*i*\_STATUS, IA32\_MC*i*\_ADDR, IA32\_MC*i*\_MISC and IA32\_MCG\_STATUS registers for each bank that is valid. The routine minimizes processing time by recording the raw data into a system data structure or file, reducing the overhead associated with polling. User utilities analyze the collected data in an off-line environment.

When the MCIP flag is set in the IA32\_MCG\_STATUS register, a machine-check exception is in progress and the machine-check exception handler has called the exception logging routine.

Once the logging process has been completed the exception-handling routine must determine whether execution can be restarted, which is usually possible when damage has not occurred (The PCC flag is clear, in the IA32\_MC*i*\_STATUS register) and when the processor can guarantee that execution is restartable (the RIPV flag is set in the IA32\_MCG\_STATUS register). If execution cannot be restarted, the system is not recoverable and the exception-handling routine should signal the console appropriately before returning the error status to the Operating System kernel for subsequent shutdown.

The machine-check architecture allows buffering of exceptions from a given error-reporting bank although the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors do not implement this feature. The error logging routine should provide compatibility with future processors by reading each hardware error-reporting bank's IA32\_MC*i*\_STATUS register and then writing 0s to clear the OVER and VAL flags in this register. The error logging utility should re-read the IA32\_MC*i*\_STATUS register for the bank ensuring that the valid bit is clear. The processor will write the next error into the register bank and set the VAL flags.

Additional information that should be stored by the exception-logging routine includes the processor's time-stamp counter value, which provides a mechanism to indicate the frequency of exceptions. A multiprocessing operating system stores the identity of the processor node incurring the exception using a unique identifier, such as the processor's APIC ID (see Section 10.8, "Handling Interrupts").

The basic algorithm given in Example 15-3 can be modified to provide more robust recovery techniques. For example, software has the flexibility to attempt recovery using information unavailable to the hardware. Specifically, the machine-check exception handler can, after logging carefully analyze the error-reporting registers when the error-logging routine reports an error that does not allow execution to be restarted. These recovery techniques

can use external bus related model-specific information provided with the error report to localize the source of the error within the system and determine the appropriate recovery strategy.

## 15.10.4 Machine-Check Software Handler Guidelines for Error Recovery

### 15.10.4.1 Machine-Check Exception Handler for Error Recovery

When writing a machine-check exception (MCE) handler to support software recovery from Uncorrected Recoverable (UCR) errors, consider the following:

- When IA32\_MCG\_CAP [24] is zero, there are no recoverable errors supported and all machine-check are fatal exceptions. The logging of status and error information is therefore a baseline implementation requirement.
- When IA32\_MCG\_CAP [24] is 1, certain uncorrected errors called uncorrected recoverable (UCR) errors may be software recoverable. The handler can analyze the reported error information, and in some cases attempt to recover from the uncorrected error and continue execution.
- For processors on which CPUID reports DisplayFamily\_DisplayModel as 06H\_0EH and onward, an MCA signal is broadcast to all logical processors in the system (see CPUID instruction in Chapter 3, “Instruction Set Reference, A-L” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*). Due to the potentially shared machine check MSR resources among the logical processors on the same package/core, the MCE handler may be required to synchronize with the other processors that received a machine check error and serialize access to the machine check registers when analyzing, logging and clearing the information in the machine check registers.
  - On processors that indicate ability for local machine-check exception (MCG\_LMCE\_P), hardware can choose to report the error to only a single logical processor if system software has enabled LMCE by setting IA32\_MCG\_EXT\_CTL[LMCE\_EN] = 1 as outlined in Section 15.3.1.5.
- The VAL (valid) flag in each IA32\_MCi\_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and should not be checked.
- The MCE handler is primarily responsible for processing uncorrected errors. The UC flag in each IA32\_MCi\_Status register indicates whether the reported error was corrected (UC=0) or uncorrected (UC=1). The MCE handler can optionally log and clear the corrected errors in the MC banks if it can implement software algorithm to avoid the undesired race conditions with the CMCI or CMC polling handler.
- For uncorrectable errors, the EIPV flag in the IA32\_MCG\_STATUS register indicates (when set) that the instruction pointed to by the instruction pointer pushed onto the stack when the machine-check exception is generated is directly associated with the error. When this flag is cleared, the instruction pointed to may not be associated with the error.
- The MCIP flag in the IA32\_MCG\_STATUS register indicates whether a machine-check exception was generated. When a machine check exception is generated, it is expected that the MCIP flag in the IA32\_MCG\_STATUS register is set to 1. If it is not set, this machine check was generated by either an INT 18 instruction or some piece of hardware signaling an interrupt with vector 18.

When IA32\_MCG\_CAP [24] is 1, the following rules can apply when writing a machine check exception (MCE) handler to support software recovery:

- The PCC flag in each IA32\_MCi\_STATUS register indicates whether recovery from the error is possible for uncorrected errors (UC=1). If the PCC flag is set for enabled uncorrected errors (UC=1 and EN=1), recovery is not possible. When recovery is not possible, the MCE handler typically records the error information and signals the operating system to reset the system.
- The RIPV flag in the IA32\_MCG\_STATUS register indicates whether restarting the program execution from the instruction pointer saved on the stack for the machine check exception is possible. When the RIPV is set, program execution can be restarted reliably when recovery is possible. If the RIPV flag is not set, program execution cannot be restarted reliably. In this case the recovery algorithm may involve terminating the current program execution and resuming an alternate thread of execution upon return from the machine check handler when recovery is possible. When recovery is not possible, the MCE handler signals the operating system to reset the system.



- When the EN flag is zero but the VAL and UC flags are one in the IA32\_MCi\_STATUS register, the reported uncorrected error in this bank is not enabled. As uncorrected errors with the EN flag = 0 are not the source of machine check exceptions, the MCE handler should log and clear non-enabled errors when the S bit is set and should continue searching for enabled errors from the other IA32\_MCi\_STATUS registers. Note that when IA32\_MCG\_CAP [24] is 0, any uncorrected error condition (VAL =1 and UC=1) including the one with the EN flag cleared are fatal and the handler must signal the operating system to reset the system. For the errors that do not generate machine check exceptions, the EN flag has no meaning.
- When the VAL flag is one, the UC flag is one, the EN flag is one and the PCC flag is zero in the IA32\_MCi\_STATUS register, the error in this bank is an uncorrected recoverable (UCR) error. The MCE handler needs to examine the S flag and the AR flag to find the type of the UCR error for software recovery and determine if software error recovery is possible.
- When both the S and the AR flags are clear in the IA32\_MCi\_STATUS register for the UCR error (VAL=1, UC=1, EN=x and PCC=0), the error in this bank is an uncorrected no-action required error (UCNA). UCNA errors are uncorrected but do not require any OS recovery action to continue execution. These errors indicate that some data in the system is corrupt, but that data has not been consumed and may not be consumed. If that data is consumed a non-UNCA machine check exception will be generated. UCNA errors are signaled in the same way as corrected machine check errors and the CMCI and CMC polling handler is primarily responsible for handling UCNA errors. Like corrected errors, the MCA handler can optionally log and clear UCNA errors as long as it can avoid the undesired race condition with the CMCI or CMC polling handler. As UCNA errors are not the source of machine check exceptions, the MCA handler should continue searching for uncorrected or software recoverable errors in all other MC banks.
- When the S flag in the IA32\_MCi\_STATUS register is set for the UCR error ((VAL=1, UC=1, EN=1 and PCC=0), the error in this bank is software recoverable and it was signaled through a machine-check exception. The AR flag in the IA32\_MCi\_STATUS register further clarifies the type of the software recoverable errors.
- When the AR flag in the IA32\_MCi\_STATUS register is clear for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action optional (SRAO) error. The MCE handler and the operating system can analyze the IA32\_MCi\_STATUS [15:0] to implement MCA error code specific optional recovery action, but this recovery action is optional. System software can resume the program execution from the instruction pointer saved on the stack for the machine check exception when the RIPV flag in the IA32\_MCG\_STATUS register is set.
- Even if the OVER flag in the IA32\_MCi\_STATUS register is set for the SRAO error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=0), the MCE handler can take recovery action for the SRAO error logged in the IA32\_MCi\_STATUS register. Since the recovery action for SRAO errors is optional, restarting the program execution from the instruction pointer saved on the stack for the machine check exception is still possible for the overflowed SRAO error if the RIPV flag in the IA32\_MCG\_STATUS is set.
- When the AR flag in the IA32\_MCi\_STATUS register is set for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action required (SRAR) error. The MCE handler and the operating system must take recovery action in order to continue execution after the machine-check exception. The MCA handler and the operating system need to analyze the IA32\_MCi\_STATUS [15:0] to determine the MCA error code specific recovery action. If no recovery action can be performed, the operating system must reset the system.
- When the OVER flag in the IA32\_MCi\_STATUS register is set for the SRAR error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=1), the MCE handler cannot take recovery action as the information of the SRAR error in the IA32\_MCi\_STATUS register was potentially lost due to the overflow condition. Since the recovery action for SRAR errors must be taken, the MCE handler must signal the operating system to reset the system.
- When the MCE handler cannot find any uncorrected (VAL=1, UC=1 and EN=1) or any software recoverable errors (VAL=1, UC=1, EN=1, PCC=0 and S=1) in any of the IA32\_MCi banks of the processors, this is an unexpected condition for the MCE handler and the handler should signal the operating system to reset the system.
- Before returning from the machine-check exception handler, software must clear the MCIP flag in the IA32\_MCG\_STATUS register. The MCIP flag is used to detect recursion. The machine-check architecture does not support recursion. When the processor receives a machine check when MCIP is set, it automatically enters the shutdown state.

Example 15-4 gives pseudocode for an MC exception handler that supports recovery of UCR.



**Example 15-4. Machine-Check Error Handler Pseudocode Supporting UCR**

```

MACHINE CHECK HANDLER: (* Called from INT 18 handler *)
NOERROR = TRUE;
ProcessorCount = 0;
IF CPU supports MCA
  THEN
    RESTARTABILITY = TRUE;
    IF (Processor Family = 6 AND DisplayModel ≥ 0EH) OR (Processor Family > 6)
      THEN
        IF ( MCG_LMCE = 1)
          MCA_BROADCAST = FALSE;
        ELSE
          MCA_BROADCAST = TRUE;
        FI;
        Acquire SpinLock;
        ProcessorCount++; (* Allowing one logical processor at a time to examine machine check registers *)
        CALL MCA ERROR PROCESSING; (* returns RESTARTABILITY and NOERROR *)
      ELSE
        MCA_BROADCAST = FALSE;
        (* Implement a rendezvous mechanism with the other processors if necessary *)
        CALL MCA ERROR PROCESSING;
      FI;
    ELSE (* Pentium(R) processor compatible *)
      READ P5_MC_ADDR
      READ P5_MC_TYPE;
      RESTARTABILITY = FALSE;
    FI;
  FI;

IF NOERROR = TRUE
  THEN
    IF NOT (MCG_RIPV = 1 AND MCG_EIPV = 0)
      THEN
        RESTARTABILITY = FALSE;
      FI
    FI;

IF RESTARTABILITY = FALSE
  THEN
    Report RESTARTABILITY to console;
    Reset system;
  FI;

IF MCA_BROADCAST = TRUE
  THEN
    IF ProcessorCount = MAX_PROCESSORS
      AND NOERROR = TRUE
        THEN
          Report RESTARTABILITY to console;
          Reset system;
        FI;
    Release SpinLock;
    Wait till ProcessorCount = MAX_PROCESSORS on system;
    (* implement a timeout and abort function if necessary *)
  FI;
CLEAR IA32_MCG_STATUS;
RESUME Execution;
(* End of MACHINE CHECK HANDLER*)

```

```

MCA ERROR PROCESSING: (* MCA Error Processing Routine called from MCA Handler *)
IF MCIP flag in IA32_MCG_STATUS = 0
  THEN (* MCIP=0 upon MCA is unexpected *)
    RESTARTABILITY = FALSE;
  FI;

```

## MACHINE-CHECK ARCHITECTURE

FOR each bank of machine-check registers

DO

CLEAR\_MC\_BANK = FALSE;

READ IA32\_MCI\_STATUS;

IF VAL Flag in IA32\_MCI\_STATUS = 1

THEN

IF UC Flag in IA32\_MCI\_STATUS = 1

THEN

IF Bit 24 in IA32\_MCG\_CAP = 0

THEN (\* the processor does not support software error recovery \*)

RESTARTABILITY = FALSE;

NOERROR = FALSE;

GOTO LOG MCA REGISTER;

FI;

(\* the processor supports software error recovery \*)

IF EN Flag in IA32\_MCI\_STATUS = 0 AND OVER Flag in IA32\_MCI\_STATUS=0

THEN (\* It is a spurious MCA Log. Log and clear the register \*)

CLEAR\_MC\_BANK = TRUE;

GOTO LOG MCA REGISTER;

FI;

IF PCC = 1 and EN = 1 in IA32\_MCI\_STATUS

THEN (\* processor context might have been corrupted \*)

RESTARTABILITY = FALSE;

ELSE (\* It is an uncorrected recoverable (UCR) error \*)

IF S Flag in IA32\_MCI\_STATUS = 0

THEN

IF AR Flag in IA32\_MCI\_STATUS = 0

THEN (\* It is an uncorrected no action required (UCNA) error \*)

GOTO CONTINUE; (\* let CMCI and CMC polling handler to process \*)

ELSE

RESTARTABILITY = FALSE; (\* S=0, AR=1 is illegal \*)

FI

FI;

IF RESTARTABILITY = FALSE

THEN (\* no need to take recovery action if RESTARTABILITY is already false \*)

NOERROR = FALSE;

GOTO LOG MCA REGISTER;

FI;

(\* S in IA32\_MCI\_STATUS = 1 \*)

IF AR Flag in IA32\_MCI\_STATUS = 1

THEN (\* It is a software recoverable and action required (SRAR) error \*)

IF OVER Flag in IA32\_MCI\_STATUS = 1

THEN

RESTARTABILITY = FALSE;

NOERROR = FALSE;

GOTO LOG MCA REGISTER;

FI

IF MCACOD Value in IA32\_MCI\_STATUS is recognized

AND Current Processor is an Affected Processor

THEN

Implement MCACOD specific recovery action;

CLEAR\_MC\_BANK = TRUE;

ELSE

RESTARTABILITY = FALSE;

FI;

ELSE (\* It is a software recoverable and action optional (SRAO) error \*)

IF OVER Flag in IA32\_MCI\_STATUS = 0 AND

MCACOD in IA32\_MCI\_STATUS is recognized

THEN

Implement MCACOD specific recovery action;

FI;

CLEAR\_MC\_BANK = TRUE;

FI; AR

FI; PCC

NOERROR = FALSE;

```

        GOTO LOG MCA REGISTER;
    ELSE (* It is a corrected error; continue to the next IA32_MCi_STATUS *)
        GOTO CONTINUE;
    FI; UC
    FI; VAL
LOG MCA REGISTER:
    SAVE IA32_MCi_STATUS;
    If MISCV in IA32_MCi_STATUS
        THEN
            SAVE IA32_MCi_MISC;
        FI;
    IF ADDRv in IA32_MCi_STATUS
        THEN
            SAVE IA32_MCi_ADDR;
        FI;
    IF CLEAR_MC_BANK = TRUE
        THEN
            SET all 0 to IA32_MCi_STATUS;
            If MISCV in IA32_MCi_STATUS
                THEN
                    SET all 0 to IA32_MCi_MISC;
                FI;
            IF ADDRv in IA32_MCi_STATUS
                THEN
                    SET all 0 to IA32_MCi_ADDR;
                FI;
        FI;
    CONTINUE:
    OD;
(*END FOR *)
RETURN;
(* End of MCA ERROR PROCESSING*)

```

#### 15.10.4.2 Corrected Machine-Check Handler for Error Recovery

When writing a corrected machine check handler, which is invoked as a result of CMCI or called from an OS CMC Polling dispatcher, consider the following:

- The VAL (valid) flag in each IA32\_MCi\_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank does not contain valid error information and does not need to be checked.
- The CMCI or CMC polling handler is responsible for logging and clearing corrected errors. The UC flag in each IA32\_MCi\_Status register indicates whether the reported error was corrected (UC=0) or not (UC=1).
- When IA32\_MCG\_CAP [24] is one, the CMC handler is also responsible for logging and clearing uncorrected no-action required (UCNA) errors. When the UC flag is one but the PCC, S, and AR flags are zero in the IA32\_MCi\_STATUS register, the reported error in this bank is an uncorrected no-action required (UCNA) error. In cases when SRAO error are signaled as UCNA error via CMCI, software can perform recovery for those errors identified in Table 15-16.
- In addition to corrected errors and UCNA errors, the CMC handler optionally logs uncorrected (UC=1 and PCC=1), software recoverable machine check errors (UC=1, PCC=0 and S=1), but should avoid clearing those errors from the MC banks. Clearing these errors may result in accidentally removing these errors before these errors are actually handled and processed by the MCE handler for attempted software error recovery.

Example 15-5 gives pseudocode for a CMCI handler with UCR support.

**Example 15-5. Corrected Error Handler Pseudocode with UCR Support**

Corrected Error HANDLER: (\* Called from CMCI handler or OS CMC Polling Dispatcher\*)  
 IF CPU supports MCA

```

  THEN
    FOR each bank of machine-check registers
      DO
        READ IA32_MCI_STATUS;
        IF VAL flag in IA32_MCI_STATUS = 1
          THEN
            IF UC Flag in IA32_MCI_STATUS = 0 (* It is a corrected error *)
              THEN
                GOTO LOG CMC ERROR;
              ELSE
                IF Bit 24 in IA32_MCG_CAP = 0
                  THEN
                    GOTO CONTINUE;
                FI;
                IF S Flag in IA32_MCI_STATUS = 0 AND AR Flag in IA32_MCI_STATUS = 0
                  THEN (* It is a uncorrected no action required error *)
                    GOTO LOG CMC ERROR
                FI
                IF EN Flag in IA32_MCI_STATUS = 0
                  THEN (* It is a spurious MCA error *)
                    GOTO LOG CMC ERROR
                FI;
              FI;
            FI;
            GOTO CONTINUE;
          LOG CMC ERROR:
            SAVE IA32_MCI_STATUS;
            If MISCV Flag in IA32_MCI_STATUS
              THEN
                SAVE IA32_MCI_MISC;
                SET all 0 to IA32_MCI_MISC;
            FI;
            IF ADDR_V Flag in IA32_MCI_STATUS
              THEN
                SAVE IA32_MCI_ADDR;
                SET all 0 to IA32_MCI_ADDR
            FI;
            SET all 0 to IA32_MCI_STATUS;
            CONTINUE:
          OD;
        (*END FOR *)
      FI;
  
```

### 13. 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*.

-----  
Changes to this chapter: Added Section 19.13 "Performance Monitoring Events for Processors Based on the Goldmont Plus Microarchitecture". Updates to Section 19.14 "Performance Monitoring Events for Processors Based on the Goldmont Microarchitecture".

# CHAPTER 19

## PERFORMANCE MONITORING EVENTS

---

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.

Model-specific performance events are listed for each generation of microarchitecture:

- Section 19.2 - Processors based on Skylake microarchitecture
- Section 19.3 - Processors based on Skylake and Kaby Lake microarchitectures
- Section 19.4 - Processors based on Knights Landing microarchitecture
- Section 19.5 - Processors based on Broadwell microarchitecture
- Section 19.6 - Processors based on Haswell microarchitecture
- Section 19.6.1 - Processors based on Haswell-E microarchitecture
- Section 19.7 - Processors based on Ivy Bridge microarchitecture
- Section 19.7.1 - Processors based on Ivy Bridge-E microarchitecture
- Section 19.8 - Processors based on Sandy Bridge microarchitecture
- Section 19.9 - Processors based on Intel® microarchitecture code name Nehalem
- Section 19.10 - Processors based on Intel® microarchitecture code name Westmere
- Section 19.11 - Processors based on Enhanced Intel® Core™ microarchitecture
- Section 19.12 - Processors based on Intel® Core™ microarchitecture
- Section 19.13 - Processors based on the Goldmont microarchitecture
- Section 19.15 - Processors based on the Silvermont microarchitecture
- Section 19.15.1 - Processors based on the Airmont microarchitecture
- Section 19.16 - 45 nm and 32 nm Intel® Atom™ Processors
- Section 19.17 - Intel® Core™ Solo and Intel® Core™ Duo processors
- Section 19.18 - Processors based on Intel NetBurst® microarchitecture
- Section 19.19 - Pentium® M family processors
- Section 19.20 - P6 family processors
- Section 19.21 - 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.

All performance event encodings not documented in the appropriate tables for the given processor are considered reserved, and their use will result in undefined counter updates with associated overflow actions.

The event tables listed in this chapter provide information for tool developers to support architectural and model-specific performance monitoring events. The tables are up to date at processor launch, but are subject to changes. The most up to date event tables and additional details of performance event implementation for end-user (including additional details beyond event code/umask) can be found at the “perfmon” repository provided by The Intel Open Source Technology Center (<https://download.01.org/perfmon/>).

## 19.1 ARCHITECTURAL PERFORMANCE MONITORING EVENTS

Architectural performance events are introduced in Intel Core Solo and Intel Core Duo processors. They are also supported on processors based on Intel Core microarchitecture. Table 19-1 lists pre-defined architectural performance events that can be configured using general-purpose performance counters and associated event-select registers.

**Table 19-1. Architectural Performance Events**

Event Num.	Event Mask Name	Umask Value	Description
3CH	UnHalted Core Cycles	00H	Counts core clock cycles whenever the logical processor is in C0 state (not halted). The frequency of this event varies with state transitions in the core.
3CH	UnHalted Reference Cycles <sup>1</sup>	01H	Counts at a fixed frequency whenever the logical processor is in C0 state (not halted).
C0H	Instructions Retired	00H	Counts when the last uop of an instruction retires.
2EH	LLC Reference	4FH	Counts requests originating from the core that reference a cache line in the last level on-die cache.
2EH	LLC Misses	41H	Counts each cache miss condition for references to the last level on-die cache.
C4H	Branch Instruction Retired	00H	Counts when the last uop of a branch instruction retires.
C5H	Branch Misses Retired	00H	Counts when the last uop of a branch instruction retires which corrected misprediction of the branch prediction hardware at execution time.

### NOTES:

1. Current implementations count at core crystal clock, TSC, or bus clock frequency.

Fixed-function performance counters count only events defined in Table 19-2.

**Table 19-2. Fixed-Function Performance Counter and Pre-defined Performance Events**

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
IA32_PERF_FIXED_CTR0	309H	Inst_Retired.Any	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers.
IA32_PERF_FIXED_CTR1	30AH	CPU_CLK_UNHALTED.THREAD/CPU_CLK_UNHALTED.CORE/CPU_CLK_UNHALTED.THREAD_ANY	<p>The CPU_CLK_UNHALTED.THREAD event counts the number of core cycles while the logical processor is not in a halt state.</p> <p>If there is only one logical processor in a processor core, CPU_CLK_UNHALTED.CORE counts the unhalting cycles of the processor core.</p> <p>If there are more than one logical processor in a processor core, CPU_CLK_UNHALTED.THREAD_ANY is supported by programming IA32_FIXED_CTR_CTRL[bit 6]AnyThread = 1.</p> <p>The core frequency may change from time to time due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason this event may have a changing ratio with regards to time.</p>

**Table 19-2. Fixed-Function Performance Counter and Pre-defined Performance Events (Contd.)**

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
IA32_PERF_FIXED_CTR2	30BH	CPU_CLK_UNHALTED.REF_TSC	This event counts the number of reference cycles at the TSC rate when the core is not in a halt state and not in a TM stop-clock state. The core enters the halt state when it is running the HLT instruction or the MWAIT instruction. This event is not affected by core frequency changes (e.g., P states) but counts at the same frequency as the time stamp counter. This event can approximate elapsed time while the core was not in a halt state and not in a TM stopclock state.

## 19.2 PERFORMANCE MONITORING EVENTS FOR INTEL® XEON® PROCESSOR SCALABLE FAMILY

The Intel® Xeon® Processor Scalable Family is based on the Skylake microarchitecture. These processors support the architectural performance monitoring events listed in Table 19-1. Fixed counters in the core PMU support the architecture events defined in Table 19-2. Model-specific performance monitoring events in the processor core are listed in Table 19-4. The events in Table 19-4 apply to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following value: 06\_55H .

The comment column in Table 19-4 uses abbreviated letters to indicate additional conditions applicable to the Event Mask Mnemonic. For event umasks listed in Table 19-4 that do not show "AnyT", users should refrain from programming "AnyThread =1" in IA32\_PERF\_EVTSELx.

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
00H	01H	INST_RETIRED.ANY	Counts the number of instructions retired from execution. For instructions that consist of multiple micro-ops, Counts the retirement of the last micro-op of the instruction. Counting continues during hardware interrupts, traps, and inside interrupt handlers. Notes: INST_RETIRED.ANY is counted by a designated fixed counter, leaving the four (eight when Hyperthreading is disabled) programmable counters available for other events. INST_RETIRED.ANY_P is counted by a programmable counter and it is an architectural performance event. Counting: Faulting executions of GETSEC/VM entry/VM Exit/MWait will not count as retired instructions.	Fixed Counter
00H	02H	CPU_CLK_UNHALTED.THREAD	Counts the number of core cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. This event is a component in many key event ratios. The core frequency may change from time to time due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason this event may have a changing ratio with regards to time. When the core frequency is constant, this event can approximate elapsed time while the core was not in the halt state. It is counted on a dedicated fixed counter, leaving the four (eight when Hyperthreading is disabled) programmable counters available for other events.	Fixed Counter



**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
00H	02H	CPU_CLK_UNHALTED.THREAD_ANY	Core cycles when at least one thread on the physical core is not in halt state.	AnyThread=1
00H	03H	CPU_CLK_UNHALTED.REF_TSC	Counts the number of reference cycles when the core is not in a halt state. The core enters the halt state when it is running the HLT instruction or the MWAIT instruction. This event is not affected by core frequency changes (for example, P states, TM2 transitions) but has the same incrementing frequency as the time stamp counter. This event can approximate elapsed time while the core was not in a halt state. This event has a constant ratio with the CPU_CLK_UNHALTED.REF_XCLK event. It is counted on a dedicated fixed counter, leaving the four (eight when Hyperthreading is disabled) programmable counters available for other events. Note: On all current platforms this event stops counting during 'throttling (TM)' states duty off periods the processor is 'halted'. The counter update is done at a lower clock rate than the core clock the overflow status bit for this counter may appear 'sticky'. After the counter has overflowed and software clears the overflow status bit and resets the counter to less than MAX. The reset value to the counter is not clocked immediately so the overflow status bit will flip "high (1)" and generate another PMI (if enabled) after which the reset value gets clocked into the counter. Therefore, software will get the interrupt, read the overflow status bit '1 for bit 34 while the counter value is less than MAX. Software should ignore this case.	Fixed Counter
03H	02H	LD_BLOCKS.STORE_FORWARD	Counts how many times the load operation got the true Block-on-Store blocking code preventing store forwarding. This includes cases when: a. preceding store conflicts with the load (incomplete overlap), b. store forwarding is impossible due to u-arch limitations, c. preceding lock RMW operations are not forwarded, d. store has the no-forward bit set (uncacheable/page-split/masked stores), e. all-blocking stores are used (mostly, fences and port I/O), and others. The most common case is a load blocked due to its address range overlapping with a preceding smaller uncompleted store. Note: This event does not take into account cases of out-of-SW-control (for example, SbTailHit), unknown physical STA, and cases of blocking loads on store due to being non-WB memory type or a lock. These cases are covered by other events. See the table of not supported store forwards in the Optimization Guide.	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are temporarily blocked because all resources for handling the split accesses are in use.	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
07H	01H	LD_BLOCKS_PARTIAL.ADDRESS_ALIAS	Counts false dependencies in MOB when the partial comparison upon loose net check and dependency was resolved by the Enhanced Loose net mechanism. This may not result in high performance penalties. Loose net checks can fail when loads and stores are 4k aliased.	
08H	01H	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Counts demand data loads that caused a page walk of any page size (4K/2M/4M/1G). This implies it missed in all TLB levels, but the walk need not have completed.	
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED_4K	Counts demand data loads that caused a completed page walk (4K page size). This implies it missed in all TLB levels. The page walk can end with or without a fault.	
08H	04H	DTLB_LOAD_MISSES.WALK_COMPLETED_2M_4M	Counts demand data loads that caused a completed page walk (2M and 4M page sizes). This implies it missed in all TLB levels. The page walk can end with or without a fault.	
08H	08H	DTLB_LOAD_MISSES.WALK_COMPLETED_1G	Counts load misses in all DTLB levels that cause a completed page walk (1G page size). The page walk can end with or without a fault.	
08H	0EH	DTLB_LOAD_MISSES.WALK_COMPLETED	Counts demand data loads that caused a completed page walk of any page size (4K/2M/4M/1G). This implies it missed in all TLB levels. The page walk can end with or without a fault.	
08H	10H	DTLB_LOAD_MISSES.WALK_PENALTY	Counts 1 per cycle for each PMH that is busy with a page walk for a load. EPT page walk duration are excluded in Skylake microarchitecture.	
08H	10H	DTLB_LOAD_MISSES.WALK_ACTIVE	Counts cycles when at least one PMH (Page Miss Handler) is busy with a page walk for a load.	CounterMask=1 CMSK1
08H	20H	DTLB_LOAD_MISSES.STLB_HIT	Counts loads that miss the DTLB (Data TLB) and hit the STLB (Second level TLB).	
0DH	01H	INT_MISC.RECOVERY_CYCLES	Core cycles the Resource allocator was stalled due to recovery from an earlier branch misprediction or machine clear event.	
0DH	01H	INT_MISC.RECOVERY_CYCLES_ANY	Core cycles the allocator was stalled due to recovery from earlier clear event for any thread running on the physical core (e.g. misprediction or memory nuke).	AnyThread=1 AnyT
0DH	80H	INT_MISC.CLEAR_RESTEER_CYCLES	Cycles the issue-stage is waiting for front-end to fetch from resteeered path following branch misprediction or machine clear events.	
0EH	01H	UOPS_ISSUED.ANY	Counts the number of uops that the Resource Allocation Table (RAT) issues to the Reservation Station (RS).	
0EH	01H	UOPS_ISSUED.STALL_CYCLES	Counts cycles during which the Resource Allocation Table (RAT) does not issue any uops to the reservation station (RS) for the current thread.	CounterMask=1 Invert=1 CMSK1, INV

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
0EH	02H	UOPS_ISSUED.VECTOR_WIDTH_MISMATCH	Counts the number of Blend Uops issued by the Resource Allocation Table (RAT) to the reservation station (RS) in order to preserve upper bits of vector registers. Starting with the Skylake microarchitecture, these Blend uops are needed since every Intel SSE instruction executed in Dirty Upper State needs to preserve bits 128-255 of the destination register. For more information, refer to Mixing Intel AVX and Intel SSE Code section of the Optimization Guide.	
0EH	20H	UOPS_ISSUED.SLOW_LEA	Number of slow LEA uops being allocated. A uop is generally considered SlowLea if it has 3 sources (e.g. 2 sources + immediate) regardless if as a result of LEA instruction or not.	
14H	01H	ARITH.DIVIDER_ACTIVE	Cycles when divide unit is busy executing divide or square root operations. Accounts for integer and floating-point operations.	CounterMask=1
24H	21H	L2_RQSTS.DEMAND_DATA_RD_MISS	Counts the number of demand Data Read requests that miss L2 cache. Only not rejected loads are counted.	
24H	22H	L2_RQSTS.RFO_MISS	Counts the RFO (Read-for-Ownership) requests that miss L2 cache.	
24H	24H	L2_RQSTS.CODE_RD_MISS	Counts L2 cache misses when fetching instructions.	
24H	27H	L2_RQSTS.ALL_DEMAND_MISS	Demand requests that miss L2 cache.	
24H	38H	L2_RQSTS.PF_MISS	Counts requests from the L1/L2/L3 hardware prefetchers or Load software prefetches that miss L2 cache.	
24H	3FH	L2_RQSTS.MISS	All requests that miss L2 cache.	
24H	41H	L2_RQSTS.DEMAND_DATA_RD_HIT	Counts the number of demand Data Read requests that hit L2 cache. Only non rejected loads are counted.	
24H	42H	L2_RQSTS.RFO_HIT	Counts the RFO (Read-for-Ownership) requests that hit L2 cache.	
24H	44H	L2_RQSTS.CODE_RD_HIT	Counts L2 cache hits when fetching instructions, code reads.	
24H	D8H	L2_RQSTS.PF_HIT	Counts requests from the L1/L2/L3 hardware prefetchers or Load software prefetches that hit L2 cache.	
24H	E1H	L2_RQSTS.ALL_DEMAND_DATA_RD	Counts the number of demand Data Read requests (including requests from L1D hardware prefetchers). These loads may hit or miss L2 cache. Only non rejected loads are counted.	
24H	E2H	L2_RQSTS.ALL_RFO	Counts the total number of RFO (read for ownership) requests to L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches.	
24H	E4H	L2_RQSTS.ALL_CODE_RD	Counts the total number of L2 code requests.	
24H	E7H	L2_RQSTS.ALL_DEMAND_REFERENCES	Demand requests to L2 cache.	
24H	F8H	L2_RQSTS.ALL_PF	Counts the total number of requests from the L2 hardware prefetchers.	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	FFH	L2_RQSTS.REFERENCES	All L2 requests.	
28H	07H	CORE_POWER.LVL0_TURBO_LICENSE	Core cycles where the core was running with power-delivery for baseline license level 0. This includes non-AVX codes, SSE, AVX 128-bit, and low-current AVX 256-bit codes.	
28H	18H	CORE_POWER.LVL1_TURBO_LICENSE	Core cycles where the core was running with power-delivery for license level 1. This includes high current AVX 256-bit instructions as well as low current AVX 512-bit instructions.	
28H	20H	CORE_POWER.LVL2_TURBO_LICENSE	Core cycles where the core was running with power-delivery for license level 2 (introduced in Skylake Server microarchitecture). This includes high current AVX 512-bit instructions.	
28H	40H	CORE_POWER.THROTTLE	Core cycles the out-of-order engine was throttled due to a pending power level request.	
2EH	41H	LONGEST_LAT_CACHE.MISS	Counts core-originated cacheable requests that miss the L3 cache (Longest Latency cache). Requests include data and code reads, Reads-for-Ownership (RFOs), speculative accesses and hardware prefetches from L1 and L2. It does not include all misses to the L3.	See Table 19-1.
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	Counts core-originated cacheable requests to the L3 cache (Longest Latency cache). Requests include data and code reads, Reads-for-Ownership (RFOs), speculative accesses and hardware prefetches from L1 and L2. It does not include all accesses to the L3.	See Table 19-1.
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	This is an architectural event that 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. For this reason, this event may have a changing ratio with regards to wall clock time.	See Table 19-1.
3CH	00H	CPU_CLK_UNHALTED.THREAD_P_ANY	Core cycles when at least one thread on the physical core is not in halt state.	AnyThread=1 AnyT
3CH	00H	CPU_CLK_UNHALTED.RINGO_TRANS	Counts when the Current Privilege Level (CPL) transitions from ring 1, 2 or 3 to ring 0 (Kernel).	EdgeDetect=1 CounterMask=1
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK	Core crystal clock cycles when the thread is unhalting.	See Table 19-1.
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK_ANY	Core crystal clock cycles when at least one thread on the physical core is unhalting.	AnyThread=1 AnyT
3CH	01H	CPU_CLK_UNHALTED.REF_XCLK	Core crystal clock cycles when the thread is unhalting.	See Table 19-1.
3CH	01H	CPU_CLK_UNHALTED.REF_XCLK_ANY	Core crystal clock cycles when at least one thread on the physical core is unhalting.	AnyThread=1 AnyT
3CH	02H	CPU_CLK_THREAD_UNHALTED.ONE_THREAD_ACTIVE	Core crystal clock cycles when this thread is unhalting and the other thread is halted.	
3CH	02H	CPU_CLK_UNHALTED.ONE_THREAD_ACTIVE	Core crystal clock cycles when this thread is unhalting and the other thread is halted.	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
48H	01H	L1D_PEND_MISS.PENDING	Counts duration of L1D miss outstanding, that is each cycle number of Fill Buffers (FB) outstanding required by Demand Reads. FB either is held by demand loads, or it is held by non-demand loads and gets hit at least once by demand. The valid outstanding interval is defined until the FB deallocation by one of the following ways: from FB allocation, if FB is allocated by demand from the demand Hit FB, if it is allocated by hardware or software prefetch. Note: In the L1D, a Demand Read contains cacheable or noncacheable demand loads, including ones causing cache-line splits and reads due to page walks resulted from any request type.	
48H	01H	L1D_PEND_MISS.PENDING_CYCLES	Counts duration of L1D miss outstanding in cycles.	CounterMask=1 CMSK1
48H	01H	L1D_PEND_MISS.PENDING_CYCLES_ANY	Cycles with L1D load Misses outstanding from any thread on physical core.	CounterMask=1 AnyThread=1 CMSK1, AnyT
48H	02H	L1D_PEND_MISS.FB_FULL	Number of times a request needed a FB (Fill Buffer) entry but there was no entry available for it. A request includes cacheable/uncacheable demands that are load, store or SW prefetch instructions.	
49H	01H	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Counts demand data stores that caused a page walk of any page size (4K/2M/4M/1G). This implies it missed in all TLB levels, but the walk need not have completed.	
49H	02H	DTLB_STORE_MISSES.WALK_COMPLETED_4K	Counts demand data stores that caused a completed page walk (4K page size). This implies it missed in all TLB levels. The page walk can end with or without a fault.	
49H	04H	DTLB_STORE_MISSES.WALK_COMPLETED_2M_4M	Counts demand data stores that caused a completed page walk (2M and 4M page sizes). This implies it missed in all TLB levels. The page walk can end with or without a fault.	
49H	08H	DTLB_STORE_MISSES.WALK_COMPLETED_1G	Counts store misses in all DTLB levels that cause a completed page walk (1G page size). The page walk can end with or without a fault.	
49H	0EH	DTLB_STORE_MISSES.WALK_COMPLETED	Counts demand data stores that caused a completed page walk of any page size (4K/2M/4M/1G). This implies it missed in all TLB levels. The page walk can end with or without a fault.	
49H	10H	DTLB_STORE_MISSES.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a page walk for a store. EPT page walk duration are excluded in Skylake microarchitecture.	
49H	10H	DTLB_STORE_MISSES.WALK_ACTIVE	Counts cycles when at least one PMH (Page Miss Handler) is busy with a page walk for a store.	CounterMask=1 CMSK1
49H	20H	DTLB_STORE_MISSES.STLB_HIT	Stores that miss the DTLB (Data TLB) and hit the STLB (2nd Level TLB).	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
4CH	01H	LOAD_HIT_PRE.SW_PF	Counts all not software-prefetch load dispatches that hit the fill buffer (FB) allocated for the software prefetch. It can also be incremented by some lock instructions. So it should only be used with profiling so that the locks can be excluded by ASM (Assembly File) inspection of the nearby instructions.	
4FH	10H	EPT.WALK_PENDING	Counts cycles for each PMH (Page Miss Handler) that is busy with an EPT (Extended Page Table) walk for any request type.	
51H	01H	L1D.REPLACEMENT	Counts L1D data line replacements including opportunistic replacements, and replacements that require stall-for-replace or block-for-replace.	
54H	01H	TX_MEM.ABORT_CONFLICT	Number of times a TSX line had a cache conflict.	
54H	02H	TX_MEM.ABORT_CAPACITY	Number of times a transactional abort was signaled due to a data capacity limitation for transactional reads or writes.	
54H	04H	TX_MEM.ABORT_HLE_STORE_T O_ELIDED_LOCK	Number of times a TSX Abort was triggered due to a non-release/commit store to lock.	
54H	08H	TX_MEM.ABORT_HLE_ELISION_ BUFFER_NOT_EMPTY	Number of times a TSX Abort was triggered due to commit but Lock Buffer not empty.	
54H	10H	TX_MEM.ABORT_HLE_ELISION_ BUFFER_MISMATCH	Number of times a TSX Abort was triggered due to release/commit but data and address mismatch.	
54H	20H	TX_MEM.ABORT_HLE_ELISION_ BUFFER_UNSUPPORTED_ALIGN MENT	Number of times a TSX Abort was triggered due to attempting an unsupported alignment from Lock Buffer.	
54H	40H	TX_MEM.HLE_ELISION_BUFFER_ FULL	Number of times we could not allocate Lock Buffer.	
5DH	01H	TX_EXEC.MISC1	Unfriendly TSX abort triggered by a flowmarker.	
5DH	02H	TX_EXEC.MISC2	Unfriendly TSX abort triggered by a vzeroupper instruction.	
5DH	04H	TX_EXEC.MISC3	Unfriendly TSX abort triggered by a nest count that is too deep.	
5DH	08H	TX_EXEC.MISC4	RTM region detected inside HLE.	
5DH	10H	TX_EXEC.MISC5	Counts the number of times an HLE XACQUIRE instruction was executed inside an RTM transactional region.	
5EH	01H	RS_EVENTS.EMPTY_CYCLES	Counts cycles during which the reservation station (RS) is empty for the thread; Note: In ST-mode, not active thread should drive 0. This is usually caused by severely costly branch mispredictions, or allocator/FE issues.	
5EH	01H	RS_EVENTS.EMPTY_END	Counts end of periods where the Reservation Station (RS) was empty. Could be useful to precisely locate front-end Latency Bound issues.	EdgeDetect=1 CounterMask=1 Invert=1 CMSK1, INV

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Counts the number of offcore outstanding Demand Data Read transactions in the super queue (SQ) every cycle. A transaction is considered to be in the Offcore outstanding state between L2 miss and transaction completion sent to requestor. See the corresponding Umask under OFFCORE_REQUESTS. Note: A prefetch promoted to Demand is counted from the promotion point.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_DATA_RD	Counts cycles when offcore outstanding Demand Data Read transactions are present in the super queue (SQ). A transaction is considered to be in the Offcore outstanding state between L2 miss and transaction completion sent to requestor (SQ de-allocation).	CounterMask=1 CMSK1
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD_GE_6	Cycles with at least 6 offcore outstanding Demand Data Read transactions in uncore queue.	CounterMask=6 CMSK6
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD	Counts the number of offcore outstanding Code Reads transactions in the super queue every cycle. The 'Offcore outstanding' state of the transaction lasts from the L2 miss until the sending transaction completion to requestor (SQ deallocation). See the corresponding Umask under OFFCORE_REQUESTS.	CounterMask=1 CMSK1
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_CODE_RD	Counts the number of offcore outstanding Code Reads transactions in the super queue every cycle. The 'Offcore outstanding' state of the transaction lasts from the L2 miss until the sending transaction completion to requestor (SQ deallocation). See the corresponding Umask under OFFCORE_REQUESTS.	CMSK1
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Counts the number of offcore outstanding RFO (store) transactions in the super queue (SQ) every cycle. A transaction is considered to be in the Offcore outstanding state between L2 miss and transaction completion sent to requestor (SQ de-allocation). See corresponding Umask under OFFCORE_REQUESTS.	CounterMask=1 CMSK1
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_RFO	Counts the number of offcore outstanding demand rfo Reads transactions in the super queue every cycle. The 'Offcore outstanding' state of the transaction lasts from the L2 miss until the sending transaction completion to requestor (SQ deallocation). See the corresponding Umask under OFFCORE_REQUESTS.	CMSK1
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Counts the number of offcore outstanding cacheable Core Data Read transactions in the super queue every cycle. A transaction is considered to be in the Offcore outstanding state between L2 miss and transaction completion sent to requestor (SQ de-allocation). See corresponding Umask under OFFCORE_REQUESTS.	
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DATA_RD	Counts cycles when offcore outstanding cacheable Core Data Read transactions are present in the super queue. A transaction is considered to be in the Offcore outstanding state between L2 miss and transaction completion sent to requestor (SQ de-allocation). See corresponding Umask under OFFCORE_REQUESTS.	CounterMask=1 CMSK1



**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.L3_MISS_DEMAND_DATA_RD	Counts number of Offcore outstanding Demand Data Read requests that miss L3 cache in the superQ every cycle.	
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_L3_MISS_DEMAND_DATA_RD	Cycles with at least 1 Demand Data Read requests who miss L3 cache in the superQ.	CounterMask=1 CMSK1
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.L3_MISS_DEMAND_DATA_RD_GE_6	Cycles with at least 6 Demand Data Read requests that miss L3 cache in the superQ.	CounterMask=6 CMSK6
79H	04H	IDQ.MITE_UOPS	Counts the number of uops delivered to Instruction Decode Queue (IDQ) from the MITE path. Counting includes uops that may 'bypass' the IDQ. This also means that uops are not being delivered from the Decode Stream Buffer (DSB).	
79H	04H	IDQ.MITE_CYCLES	Counts cycles during which uops are being delivered to Instruction Decode Queue (IDQ) from the MITE path. Counting includes uops that may 'bypass' the IDQ.	CounterMask=1 CMSK1
79H	08H	IDQ.DSB_UOPS	Counts the number of uops delivered to Instruction Decode Queue (IDQ) from the Decode Stream Buffer (DSB) path. Counting includes uops that may 'bypass' the IDQ.	
79H	08H	IDQ.DSB_CYCLES	Counts cycles during which uops are being delivered to Instruction Decode Queue (IDQ) from the Decode Stream Buffer (DSB) path. Counting includes uops that may 'bypass' the IDQ.	CounterMask=1 CMSK1
79H	10H	IDQ.MS_DSB_CYCLES	Counts cycles during which uops initiated by Decode Stream Buffer (DSB) are being delivered to Instruction Decode Queue (IDQ) while the Microcode Sequencer (MS) is busy. Counting includes uops that may 'bypass' the IDQ.	CounterMask=1
79H	18H	IDQ.ALL_DSB_CYCLES_4_UOPS	Counts the number of cycles 4 uops were delivered to Instruction Decode Queue (IDQ) from the Decode Stream Buffer (DSB) path. Count includes uops that may 'bypass' the IDQ.	CounterMask=4 CMSK4
79H	18H	IDQ.ALL_DSB_CYCLES_ANY_UOPS	Counts the number of cycles uops were delivered to Instruction Decode Queue (IDQ) from the Decode Stream Buffer (DSB) path. Count includes uops that may 'bypass' the IDQ.	CounterMask=1 CMSK1
79H	20H	IDQ.MS_MITE_UOPS	Counts the number of uops initiated by MITE and delivered to Instruction Decode Queue (IDQ) while the Microcode Sequencer (MS) is busy. Counting includes uops that may 'bypass' the IDQ.	
79H	24H	IDQ.ALL_MITE_CYCLES_4_UOPS	Counts the number of cycles 4 uops were delivered to the Instruction Decode Queue (IDQ) from the MITE (legacy decode pipeline) path. Counting includes uops that may 'bypass' the IDQ. During these cycles uops are not being delivered from the Decode Stream Buffer (DSB).	CounterMask=4 CMSK4



**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
79H	24H	IDQ.ALL_MITE_CYCLES_ANY_UOPS	Counts the number of cycles uops were delivered to the Instruction Decode Queue (IDQ) from the MITE (legacy decode pipeline) path. Counting includes uops that may 'bypass' the IDQ. During these cycles uops are not being delivered from the Decode Stream Buffer (DSB).	CounterMask=1 CMSK1
79H	30H	IDQ.MS_CYCLES	Counts cycles during which uops are being delivered to Instruction Decode Queue (IDQ) while the Microcode Sequencer (MS) is busy. Counting includes uops that may 'bypass' the IDQ. Uops maybe initiated by Decode Stream Buffer (DSB) or MITE.	CounterMask=1 CMSK1
79H	30H	IDQ.MS_SWITCHES	Number of switches from DSB (Decode Stream Buffer) or MITE (legacy decode pipeline) to the Microcode Sequencer.	EdgeDetect=1 CounterMask=1 EDGE
79H	30H	IDQ.MS_UOPS	Counts the total number of uops delivered by the Microcode Sequencer (MS). Any instruction over 4 uops will be delivered by the MS. Some instructions such as transcendentals may additionally generate uops from the MS.	
80H	04H	ICACHE_16B.IFDATA_STALL	Cycles where a code line fetch is stalled due to an L1 instruction cache miss. The legacy decode pipeline works at a 16 Byte granularity.	
83H	01H	ICACHE_64B.IFTAG_HIT	Instruction fetch tag lookups that hit in the instruction cache (L1). Counts at 64-byte cache-line granularity.	
83H	02H	ICACHE_64B.IFTAG_MISS	Instruction fetch tag lookups that miss in the instruction cache (L1). Counts at 64-byte cache-line granularity.	
83H	04H	ICACHE_64B.IFTAG_STALL	Cycles where a code fetch is stalled due to L1 instruction cache tag miss.	
85H	01H	ITLB_MISSES.MISS_CAUSES_A_WALK	Counts page walks of any page size (4K/2M/4M/1G) caused by a code fetch. This implies it missed in the ITLB and further levels of TLB, but the walk need not have completed.	
85H	02H	ITLB_MISSES.WALK_COMPLETE_D_4K	Counts completed page walks (4K page size) caused by a code fetch. This implies it missed in the ITLB and further levels of TLB. The page walk can end with or without a fault.	
85H	04H	ITLB_MISSES.WALK_COMPLETE_D_2M_4M	Counts completed page walks of any page size (4K/2M/4M/1G) caused by a code fetch. This implies it missed in the ITLB and further levels of TLB. The page walk can end with or without a fault.	
85H	08H	ITLB_MISSES.WALK_COMPLETE_D_1G	Counts store misses in all DTLB levels that cause a completed page walk (1G page size). The page walk can end with or without a fault.	
85H	0EH	ITLB_MISSES.WALK_COMPLETE_D	Counts completed page walks (2M and 4M page sizes) caused by a code fetch. This implies it missed in the ITLB and further levels of TLB. The page walk can end with or without a fault.	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
85H	10H	ITLB_MISSES.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a page walk for an instruction fetch request. EPT page walk duration are excluded in Skylake microarchitecture.	
85H	10H	ITLB_MISSES.WALK_ACTIVE	Cycles when at least one PMH is busy with a page walk for code (instruction fetch) request. EPT page walk duration are excluded in Skylake microarchitecture.	CounterMask=1
85H	20H	ITLB_MISSES.STLB_HIT	Instruction fetch requests that miss the ITLB and hit the STLB.	
87H	01H	ILD_STALL.LCP	Counts cycles that the Instruction Length decoder (ILD) stalls occurred due to dynamically changing prefix length of the decoded instruction (by operand size prefix instruction 0x66, address size prefix instruction 0x67 or REX.W for Intel64). Count is proportional to the number of prefixes in a 16B-line. This may result in a three-cycle penalty for each LCP (Length changing prefix) in a 16-byte chunk.	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CORE	Counts the number of uops not delivered to Resource Allocation Table (RAT) per thread adding "4 - x" when Resource Allocation Table (RAT) is not stalled and Instruction Decode Queue (IDQ) delivers x uops to Resource Allocation Table (RAT) (where x belongs to {0,1,2,3}). Counting does not cover cases when: a. IDQ-Resource Allocation Table (RAT) pipe serves the other thread. b. Resource Allocation Table (RAT) is stalled for the thread (including uop drops and clear BE conditions). c. Instruction Decode Queue (IDQ) delivers four uops.	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_0_UOPS_DELIV.CORE	Counts, on the per-thread basis, cycles when no uops are delivered to Resource Allocation Table (RAT). IDQ_Uops_Not_Delivered.core =4.	CounterMask=4 CMSK4
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_LE_1_UOP_DELIV.CORE	Counts, on the per-thread basis, cycles when less than 1 uop is delivered to Resource Allocation Table (RAT). IDQ_Uops_Not_Delivered.core >= 3.	CounterMask=3 CMSK3
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_LE_2_UOP_DELIV.CORE	Cycles with less than 2 uops delivered by the front end.	CounterMask=2 CMSK2
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_LE_3_UOP_DELIV.CORE	Cycles with less than 3 uops delivered by the front end.	CounterMask=1 CMSK1
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_FE_WAS_OK	Counts cycles FE delivered 4 uops or Resource Allocation Table (RAT) was stalling FE.	CounterMask=1 Invert=1 CMSK, INV
A1H	01H	UOPS_DISPATCHED_PORT.PORT_0	Counts, on the per-thread basis, cycles during which at least one uop is dispatched from the Reservation Station (RS) to port 0.	
A1H	02H	UOPS_DISPATCHED_PORT.PORT_1	Counts, on the per-thread basis, cycles during which at least one uop is dispatched from the Reservation Station (RS) to port 1.	
A1H	04H	UOPS_DISPATCHED_PORT.PORT_2	Counts, on the per-thread basis, cycles during which at least one uop is dispatched from the Reservation Station (RS) to port 2.	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A1H	08H	UOPS_DISPATCHED_PORT.PORT_3	Counts, on the per-thread basis, cycles during which at least one uop is dispatched from the Reservation Station (RS) to port 3.	
A1H	10H	UOPS_DISPATCHED_PORT.PORT_4	Counts, on the per-thread basis, cycles during which at least one uop is dispatched from the Reservation Station (RS) to port 4.	
A1H	20H	UOPS_DISPATCHED_PORT.PORT_5	Counts, on the per-thread basis, cycles during which at least one uop is dispatched from the Reservation Station (RS) to port 5.	
A1H	40H	UOPS_DISPATCHED_PORT.PORT_6	Counts, on the per-thread basis, cycles during which at least one uop is dispatched from the Reservation Station (RS) to port 6.	
A1H	80H	UOPS_DISPATCHED_PORT.PORT_7	Counts, on the per-thread basis, cycles during which at least one uop is dispatched from the Reservation Station (RS) to port 7.	
A2H	01H	RESOURCE_STALLS.ANY	Counts resource-related stall cycles. Reasons for stalls can be as follows: a. *any* u-arch structure got full (LB, SB, RS, ROB, BOB, LM, Physical Register Reclaim Table (PRRT), or Physical History Table (PHT) slots). b. *any* u-arch structure got empty (like INT/SIMD FreeLists). c. FPU control word (FPCW), MXCSR and others. This counts cycles that the pipeline back end blocked uop delivery from the front end.	
A2H	08H	RESOURCE_STALLS.SB	Counts allocation stall cycles caused by the store buffer (SB) being full. This counts cycles that the pipeline back end blocked uop delivery from the front end.	
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_MISS	Cycles while L2 cache miss demand load is outstanding.	CounterMask=1 CMSK1
A3H	02H	CYCLE_ACTIVITY.CYCLES_L3_MISS	Cycles while L3 cache miss demand load is outstanding.	CounterMask=2 CMSK2
A3H	04H	CYCLE_ACTIVITY.STALLS_TOTAL	Total execution stalls.	CounterMask=4 CMSK4
A3H	05H	CYCLE_ACTIVITY.STALLS_L2_MISS	Execution stalls while L2 cache miss demand load is outstanding.	CounterMask=5 CMSK5
A3H	06H	CYCLE_ACTIVITY.STALLS_L3_MISS	Execution stalls while L3 cache miss demand load is outstanding.	CounterMask=6 CMSK6
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_MISS	Cycles while L1 cache miss demand load is outstanding.	CounterMask=8 CMSK8
A3H	0CH	CYCLE_ACTIVITY.STALLS_L1D_MISS	Execution stalls while L1 cache miss demand load is outstanding.	CounterMask=12 CMSK12
A3H	10H	CYCLE_ACTIVITY.CYCLES_MEM_ANY	Cycles while memory subsystem has an outstanding load.	CounterMask=16 CMSK16
A3H	14H	CYCLE_ACTIVITY.STALLS_MEM_ANY	Execution stalls while memory subsystem has an outstanding load.	CounterMask=20 CMSK20
A6H	01H	EXE_ACTIVITY.EXE_BOUND_PORTS	Counts cycles during which no uops were executed on all ports and Reservation Station (RS) was not empty.	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A6H	02H	EXE_ACTIVITY.1_PORTS_UTIL	Counts cycles during which a total of 1 uop was executed on all ports and Reservation Station (RS) was not empty.	
A6H	04H	EXE_ACTIVITY.2_PORTS_UTIL	Counts cycles during which a total of 2 uops were executed on all ports and Reservation Station (RS) was not empty.	
A6H	08H	EXE_ACTIVITY.3_PORTS_UTIL	Cycles total of 3 uops are executed on all ports and Reservation Station (RS) was not empty.	
A6H	10H	EXE_ACTIVITY.4_PORTS_UTIL	Cycles total of 4 uops are executed on all ports and Reservation Station (RS) was not empty.	
A6H	40H	EXE_ACTIVITY.BOUND_ON_STORES	Cycles where the Store Buffer was full and no outstanding load.	
A8H	01H	LSD.UOPS	Number of uops delivered to the back-end by the LSD (Loop Stream Detector).	
A8H	01H	LSD.CYCLES_ACTIVE	Counts the cycles when at least one uop is delivered by the LSD (Loop-stream detector).	CounterMask=1 CMSK1
A8H	01H	LSD.CYCLES_4_UOPS	Counts the cycles when 4 uops are delivered by the LSD (Loop-stream detector).	CounterMask=4 CMSK4
ABH	02H	DSB2MITE_SWITCHES.PENALTY_CYCLES	Counts Decode Stream Buffer (DSB)-to-MITE switch true penalty cycles. These cycles do not include uops routed through because of the switch itself, for example, when Instruction Decode Queue (IDQ) pre-allocation is unavailable, or Instruction Decode Queue (IDQ) is full. SBD-to-MITE switch true penalty cycles happen after the merge mux (MM) receives Decode Stream Buffer (DSB) Sync-indication until receiving the first MITE uop. MM is placed before Instruction Decode Queue (IDQ) to merge uops being fed from the MITE and Decode Stream Buffer (DSB) paths. Decode Stream Buffer (DSB) inserts the Sync-indication whenever a Decode Stream Buffer (DSB)-to-MITE switch occurs. Penalty: A Decode Stream Buffer (DSB) hit followed by a Decode Stream Buffer (DSB) miss can cost up to six cycles in which no uops are delivered to the IDQ. Most often, such switches from the Decode Stream Buffer (DSB) to the legacy pipeline cost 0 to 2 cycles.	
AEH	01H	ITLB.ITLB_FLUSH	Counts the number of flushes of the big or small ITLB pages. Counting include both TLB Flush (covering all sets) and TLB Set Clear (set-specific).	
BOH	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Counts the Demand Data Read requests sent to uncore. Use it in conjunction with OFFCORE_REQUESTS_OUTSTANDING to determine average latency in the uncore.	
BOH	02H	OFFCORE_REQUESTS.DEMAND_CODE_RD	Counts both cacheable and non-cacheable code read requests.	
BOH	04H	OFFCORE_REQUESTS.DEMAND_RFO	Counts the demand RFO (read for ownership) requests including regular RFOs, locks, ItoM.	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B0H	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Counts the demand and prefetch data reads. All Core Data Reads include cacheable 'Demands' and L2 prefetchers (not L3 prefetchers). Counting also covers reads due to page walks resulted from any request type.	
B0H	10H	OFFCORE_REQUESTS.L3_MISS_DEMAND_DATA_RD	Demand Data Read requests who miss L3 cache.	
B0H	80H	OFFCORE_REQUESTS.ALL_REQUESTS	Counts memory transactions reached the super queue including requests initiated by the core, all L3 prefetches, page walks, etc.	
B1H	01H	UOPS_EXECUTED.THREAD	Number of uops to be executed per-thread each cycle.	
B1H	01H	UOPS_EXECUTED.STALL_CYCLES	Counts cycles during which no uops were dispatched from the Reservation Station (RS) per thread.	CounterMask=1 Invert=1 CMSK, INV
B1H	01H	UOPS_EXECUTED.CYCLES_GE_1_UOP_EXEC	Cycles where at least 1 uop was executed per-thread.	CounterMask=1 CMSK1
B1H	01H	UOPS_EXECUTED.CYCLES_GE_2_UOPS_EXEC	Cycles where at least 2 uops were executed per-thread.	CounterMask=2 CMSK2
B1H	01H	UOPS_EXECUTED.CYCLES_GE_3_UOPS_EXEC	Cycles where at least 3 uops were executed per-thread.	CounterMask=3 CMSK3
B1H	01H	UOPS_EXECUTED.CYCLES_GE_4_UOPS_EXEC	Cycles where at least 4 uops were executed per-thread.	CounterMask=4 CMSK4
B1H	02H	UOPS_EXECUTED.CORE	Number of uops executed from any thread.	
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_1	Cycles at least 1 micro-op is executed from any thread on physical core.	CounterMask=1 CMSK1
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_2	Cycles at least 2 micro-op is executed from any thread on physical core.	CounterMask=2 CMSK2
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_3	Cycles at least 3 micro-op is executed from any thread on physical core.	CounterMask=3 CMSK3
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_4	Cycles at least 4 micro-op is executed from any thread on physical core.	CounterMask=4 CMSK4
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_NONE	Cycles with no micro-ops executed from any thread on physical core.	CounterMask=1 Invert=1 CMSK1, INV
B1H	10H	UOPS_EXECUTED.X87	Counts the number of x87 uops executed.	
B2H	01H	OFFCORE_REQUESTS_BUFFER_SEQ_FULL	Counts the number of cases when the offcore requests buffer cannot take more entries for the core. This can happen when the superqueue does not contain eligible entries, or when L1D writeback pending FIFO requests is full. Note: Writeback pending FIFO has six entries.	
BDH	01H	TLB_FLUSH.DTLB_THREAD	Counts the number of DTLB flush attempts of the thread-specific entries.	
BDH	20H	TLB_FLUSH.STLB_ANY	Counts the number of any STLB flush attempts (such as entire, VPID, PCID, InvPage, CR3 write, etc.).	
COH	00H	INST_RETIRED.ANY_P	Counts the number of instructions (EOMs) retired. Counting covers macro-fused instructions individually (that is, increments by two).	See Table 19-1.

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C0H	01H	INST_RETIRED.PREC_DIST	A version of INST_RETIRED that allows for a more unbiased distribution of samples across instructions retired. It utilizes the Precise Distribution of Instructions Retired (PDIR) feature to mitigate some bias in how retired instructions get sampled.	Precise event capable Requires PEBS on General Counter 1 (PDIR).
C1H	3FH	OTHER_ASSISTS.ANY	Number of times a microcode assist is invoked by HW other than FP-assist. Examples include AD (page Access Dirty) and AVX* related assists.	
C2H	01H	UOPS_RETIRED.STALL_CYCLES	This is a non-precise version (that is, does not use PEBS) of the event that counts cycles without actually retired uops.	CounterMask=1 Invert=1 CMSK1, INV
C2H	01H	UOPS_RETIRED.TOTAL_CYCLES	Number of cycles using always true condition (uops_ret < 16) applied to non PEBS uops retired event.	CounterMask=10 Invert=1 CMSK10, INV
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the retirement slots used.	
C3H	01H	MACHINE_CLEARS.COUNT	Number of machine clears (nukes) of any type.	EdgeDetect=1 CounterMask=1 CMSK1, EDG
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Counts the number of memory ordering Machine Clears detected. Memory Ordering Machine Clears can result from one of the following: a. memory disambiguation, b. external snoop, or c. cross SMT-HW-thread snoop (stores) hitting load buffer.	
C3H	04H	MACHINE_CLEARS.SMC	Counts self-modifying code (SMC) detected, which causes a machine clear.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Counts all (macro) branch instructions retired.	Precise event capable. See Table 19-1.
C4H	01H	BR_INST_RETIRED.CONDITIONAL	This is a non-precise version (that is, does not use PEBS) of the event that counts conditional branch instructions retired.	Precise event capable. PS
C4H	02H	BR_INST_RETIRED.NEAR_CALL	This is a non-precise version (that is, does not use PEBS) of the event that counts both direct and indirect near call instructions retired.	Precise event capable. PS
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	This is a non-precise version (that is, does not use PEBS) of the event that counts return instructions retired.	Precise event capable. PS
C4H	10H	BR_INST_RETIRED.NOT_TAKEN	This is a non-precise version (that is, does not use PEBS) of the event that counts not taken branch instructions retired.	
C4H	20H	BR_INST_RETIRED.NEAR_TAKEN	This is a non-precise version (that is, does not use PEBS) of the event that counts taken branch instructions retired.	Precise event capable. PS
C4H	40H	BR_INST_RETIRED.FAR_BRANCH	This is a non-precise version (that is, does not use PEBS) of the event that counts far branch instructions retired.	Precise event capable. PS

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C5H	00H	BR_MISP_RETIRED.ALL_BRANC HES	Counts all the retired branch instructions that were mispredicted by the processor. A branch misprediction occurs when the processor incorrectly predicts the destination of the branch. When the misprediction is discovered at execution, all the instructions executed in the wrong (speculative) path must be discarded, and the processor must start fetching from the correct path.	Precise event capable. See Table 19-1.
C5H	01H	BR_MISP_RETIRED.CONDITIONA L	This is a non-precise version (that is, does not use PEBS) of the event that counts mispredicted conditional branch instructions retired.	Precise event capable. PS
C5H	02H	BR_MISP_RETIRED.NEAR_CALL	Counts both taken and not taken retired mispredicted direct and indirect near calls, including both register and memory indirect.	Precise event capable.
C5H	20H	BR_MISP_RETIRED.NEAR_TAKE N	Number of near branch instructions retired that were mispredicted and taken.	Precise event capable. PS
C6H	01H	FRONTEND_RETIRED.DSB_MISS	Counts retired Instructions that experienced DSB (Decode stream buffer, i.e. the decoded instruction-cache) miss.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.L1I_MISS	Retired Instructions who experienced Instruction L1 Cache true miss.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.L2_MISS	Retired Instructions who experienced Instruction L2 Cache true miss.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.ITLB_MISS	Counts retired Instructions that experienced iTLB (Instruction TLB) true miss.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.STLB_MIS S	Counts retired Instructions that experienced STLB (2nd level TLB) true miss.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_ GE_2	Retired instructions that are fetched after an interval where the front end delivered no uops for a period of 2 cycles which was not interrupted by a back-end stall.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_ GE_4	Retired instructions that are fetched after an interval where the front end delivered no uops for a period of 4 cycles which was not interrupted by a back-end stall.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_ GE_8	Counts retired instructions that are delivered to the back end after a front-end stall of at least 8 cycles. During this period the front end delivered no uops.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_ GE_16	Counts retired instructions that are delivered to the back end after a front-end stall of at least 16 cycles. During this period the front end delivered no uops.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_ GE_32	Counts retired instructions that are delivered to the back end after a front-end stall of at least 32 cycles. During this period the front end delivered no uops.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_ GE_64	Retired instructions that are fetched after an interval where the front end delivered no uops for a period of 64 cycles which was not interrupted by a back-end stall.	Precise event capable.



**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C6H	01H	FRONTEND_RETIRED.LATENCY_GE_128	Retired instructions that are fetched after an interval where the front end delivered no uops for a period of 128 cycles which was not interrupted by a back-end stall.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_GE_256	Retired instructions that are fetched after an interval where the front end delivered no uops for a period of 256 cycles which was not interrupted by a back-end stall.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_GE_512	Retired instructions that are fetched after an interval where the front end delivered no uops for a period of 512 cycles which was not interrupted by a back-end stall.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_GE_2_BUBBLES_GE_1	Counts retired instructions that are delivered to the back end after the front end had at least 1 bubble-slot for a period of 2 cycles. A bubble-slot is an empty issue-pipeline slot while there was no RAT stall.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_GE_2_BUBBLES_GE_2	Retired instructions that are fetched after an interval where the front end had at least 2 bubble-slots for a period of 2 cycles which was not interrupted by a back-end stall.	Precise event capable.
C6H	01H	FRONTEND_RETIRED.LATENCY_GE_2_BUBBLES_GE_3	Retired instructions that are fetched after an interval where the front end had at least 3 bubble-slots for a period of 2 cycles which was not interrupted by a back-end stall.	Precise event capable.
C7H	01H	FP_ARITH_INST_RETIRED.SCALAR_DOUBLE	Number of SSE/AVX computational scalar double precision floating-point instructions retired. Each count represents 1 computation. Applies to SSE* and AVX* scalar double precision floating-point instructions: ADD SUB MUL DIV MIN MAX SQRT FM(N)ADD/SUB. FM(N)ADD/SUB instructions count twice as they perform multiple calculations per element.	Software may treat each count as one DP FLOP.
C7H	02H	FP_ARITH_INST_RETIRED.SCALAR_SINGLE	Number of SSE/AVX computational scalar single precision floating-point instructions retired. Each count represents 1 computation. Applies to SSE* and AVX* scalar single precision floating-point instructions: ADD SUB MUL DIV MIN MAX RCP RSQRT SQRT FM(N)ADD/SUB. FM(N)ADD/SUB instructions count twice as they perform multiple calculations per element.	Software may treat each count as one SP FLOP.
C7H	04H	FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE	Number of SSE/AVX computational 128-bit packed double precision floating-point instructions retired. Each count represents 2 computations. Applies to SSE* and AVX* packed double precision floating-point instructions: ADD SUB MUL DIV MIN MAX SQRT DPP FM(N)ADD/SUB. DPP and FM(N)ADD/SUB instructions count twice as they perform multiple calculations per element.	Software may treat each count as two DP FLOPs.



**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C7H	08H	FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE	Number of SSE/AVX computational 128-bit packed single precision floating-point instructions retired. Each count represents 4 computations. Applies to SSE* and AVX* packed single precision floating-point instructions: ADD SUB MUL DIV MIN MAX RCP RSQRT SQRT DPP FM(N)ADD/SUB. DPP and FM(N)ADD/SUB instructions count twice as they perform multiple calculations per element.	Software may treat each count as four SP FLOPs.
C7H	10H	FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE	Number of SSE/AVX computational 256-bit packed double precision floating-point instructions retired. Each count represents 4 computations. Applies to SSE* and AVX* packed double precision floating-point instructions: ADD SUB MUL DIV MIN MAX SQRT DPP FM(N)ADD/SUB. DPP and FM(N)ADD/SUB instructions count twice as they perform multiple calculations per element.	Software may treat each count as four DP FLOPs.
C7H	20H	FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE	Number of SSE/AVX computational 256-bit packed single precision floating-point instructions retired. Each count represents 8 computations. Applies to SSE* and AVX* packed single precision floating-point instructions: ADD SUB MUL DIV MIN MAX RCP RSQRT SQRT DPP FM(N)ADD/SUB. DPP and FM(N)ADD/SUB instructions count twice as they perform multiple calculations per element.	Software may treat each count as eight SP FLOPs.
C7H	40H	FP_ARITH_INST_RETIRED.512B_PACKED_DOUBLE	Number of Packed Double-Precision FP arithmetic instructions (use operation multiplier of 8).	Only applicable when AVX-512 is enabled.
C7H	80H	FP_ARITH_INST_RETIRED.512B_PACKED_SINGLE	Number of Packed Single-Precision FP arithmetic instructions (use operation multiplier of 16).	Only applicable when AVX-512 is enabled.
C8H	01H	HLE_RETIRED.START	Number of times we entered an HLE region. Does not count nested transactions.	
C8H	02H	HLE_RETIRED.COMMIT	Number of times HLE commit succeeded.	
C8H	04H	HLE_RETIRED.ABORTED	Number of times HLE abort was triggered.	Precise event capable.
C8H	08H	HLE_RETIRED.ABORTED_MEM	Number of times an HLE execution aborted due to various memory events (e.g., read/write capacity and conflicts).	
C8H	10H	HLE_RETIRED.ABORTED_TIMER	Number of times an HLE execution aborted due to hardware timer expiration.	
C8H	20H	HLE_RETIRED.ABORTED_UNFRIENDLY	Number of times an HLE execution aborted due to HLE-unfriendly instructions and certain unfriendly events (such as AD assists etc.).	
C8H	40H	HLE_RETIRED.ABORTED_MEMTYPE	Number of times an HLE execution aborted due to incompatible memory type.	
C8H	80H	HLE_RETIRED.ABORTED_EVENTS	Number of times an HLE execution aborted due to unfriendly events (such as interrupts).	
C9H	01H	RTM_RETIRED.START	Number of times we entered an RTM region. Does not count nested transactions.	
C9H	02H	RTM_RETIRED.COMMIT	Number of times RTM commit succeeded.	
C9H	04H	RTM_RETIRED.ABORTED	Number of times RTM abort was triggered.	Precise event capable.

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C9H	08H	RTM_RETIRED.ABORTED_MEM	Number of times an RTM execution aborted due to various memory events (e.g. read/write capacity and conflicts).	
C9H	10H	RTM_RETIRED.ABORTED_TIMER	Number of times an RTM execution aborted due to uncommon conditions.	
C9H	20H	RTM_RETIRED.ABORTED_UNFRIENDLY	Number of times an RTM execution aborted due to HLE-unfriendly instructions.	
C9H	40H	RTM_RETIRED.ABORTED_MEMTYPE	Number of times an RTM execution aborted due to incompatible memory type.	
C9H	80H	RTM_RETIRED.ABORTED_EVENTS	Number of times an RTM execution aborted due to none of the previous 4 categories (e.g. interrupt).	
CAH	1EH	FP_ASSIST.ANY	Counts cycles with any input and output SSE or x87 FP assist. If an input and output assist are detected on the same cycle the event increments by 1.	CounterMask=1 CMSK1
CBH	01H	HW_INTERRUPTS.RECEIVED	Counts the number of hardware interruptions received by the processor.	
CCH	20H	ROB_MISC_EVENTS.LBR_INSERTS	Increments when an entry is added to the Last Branch Record (LBR) array (or removed from the array in case of RETURNS in call stack mode). The event requires LBR enable via IA32_DEBUGCTL MSR and branch type selection via MSR_LBR_SELECT.	
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY_GT_4	Counts loads when the latency from first dispatch to completion is greater than 4 cycles. Reported latency may be longer than just the memory latency.	Precise event capable. Specify threshold in MSR 3F6H.
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY_GT_8	Counts loads when the latency from first dispatch to completion is greater than 8 cycles. Reported latency may be longer than just the memory latency.	Precise event capable. Specify threshold in MSR 3F6H.
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY_GT_16	Counts loads when the latency from first dispatch to completion is greater than 16 cycles. Reported latency may be longer than just the memory latency.	Precise event capable. Specify threshold in MSR 3F6H.
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY_GT_32	Counts loads when the latency from first dispatch to completion is greater than 32 cycles. Reported latency may be longer than just the memory latency.	Precise event capable. Specify threshold in MSR 3F6H.
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY_GT_64	Counts loads when the latency from first dispatch to completion is greater than 64 cycles. Reported latency may be longer than just the memory latency.	Precise event capable. Specify threshold in MSR 3F6H.
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY_GT_128	Counts loads when the latency from first dispatch to completion is greater than 128 cycles. Reported latency may be longer than just the memory latency.	Precise event capable. Specify threshold in MSR 3F6H.
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY_GT_256	Counts loads when the latency from first dispatch to completion is greater than 256 cycles. Reported latency may be longer than just the memory latency.	Precise event capable. Specify threshold in MSR 3F6H.
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY_GT_512	Counts loads when the latency from first dispatch to completion is greater than 512 cycles. Reported latency may be longer than just the memory latency.	Precise event capable. Specify threshold in MSR 3F6H.
DOH	11H	MEM_INST_RETIRED.STLB_MISS_LOADS	Retired load instructions that miss the STLB.	Precise event capable. PSDLA

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D0H	12H	MEM_INST_RETIRED.STLB_MISS_STORES	Retired store instructions that miss the STLB.	Precise event capable. PSDLA
D0H	21H	MEM_INST_RETIRED.LOCK_LOADS	Retired load instructions with locked access.	Precise event capable. PSDLA
D0H	41H	MEM_INST_RETIRED.SPLIT_LOADS	Counts retired load instructions that split across a cacheline boundary.	Precise event capable. PSDLA
D0H	42H	MEM_INST_RETIRED.SPLIT_STORES	Counts retired store instructions that split across a cacheline boundary.	Precise event capable. PSDLA
D0H	81H	MEM_INST_RETIRED.ALL_LOADS	All retired load instructions.	Precise event capable. PSDLA
D0H	82H	MEM_INST_RETIRED.ALL_STORES	All retired store instructions.	Precise event capable. PSDLA
D1H	01H	MEM_LOAD_RETIRED.L1_HIT	Counts retired load instructions with at least one uop that hit in the L1 data cache. This event includes all SW prefetches and lock instructions regardless of the data source.	Precise event capable. PSDLA
D1H	02H	MEM_LOAD_RETIRED.L2_HIT	Retired load instructions with L2 cache hits as data sources.	Precise event capable. PSDLA
D1H	04H	MEM_LOAD_RETIRED.L3_HIT	Counts retired load instructions with at least one uop that hit in the L3 cache.	Precise event capable. PSDLA
D1H	08H	MEM_LOAD_RETIRED.L1_MISS	Counts retired load instructions with at least one uop that missed in the L1 cache.	Precise event capable. PSDLA
D1H	10H	MEM_LOAD_RETIRED.L2_MISS	Retired load instructions missed L2 cache as data sources.	Precise event capable. PSDLA
D1H	20H	MEM_LOAD_RETIRED.L3_MISS	Counts retired load instructions with at least one uop that missed in the L3 cache.	Precise event capable. PSDLA
D1H	40H	MEM_LOAD_RETIRED.FB_HIT	Counts retired load instructions with at least one uop was load missed in L1 but hit FB (Fill Buffers) due to preceding miss to the same cache line with data not ready.	Precise event capable. PSDLA
D2H	01H	MEM_LOAD_L3_HIT_RETIRED.XSNP_MISS	Retired load instructions which data sources were L3 hit and cross-core snoop missed in on-pkg core cache.	Precise event capable. PSDLA
D2H	02H	MEM_LOAD_L3_HIT_RETIRED.XSNP_HIT	Retired load instructions which data sources were L3 and cross-core snoop hits in on-pkg core cache.	Precise event capable. PSDLA
D2H	04H	MEM_LOAD_L3_HIT_RETIRED.XSNP_HITM	Retired load instructions which data sources were HitM responses from shared L3.	Precise event capable. PSDLA
D2H	08H	MEM_LOAD_L3_HIT_RETIRED.XSNP_NONE	Retired load instructions which data sources were hits in L3 without snoops required.	Precise event capable. PSDLA
D3H	01H	MEM_LOAD_L3_MISS_RETIRED.LOCAL_DRAM	Retired load instructions which data sources missed L3 but serviced from local DRAM.	Precise event capable.
D3H	02H	MEM_LOAD_L3_MISS_RETIRED.REMOTE_DRAM	Retired load instructions which data sources missed L3 but serviced from remote dram.	Precise event capable.
D3H	04H	MEM_LOAD_L3_MISS_RETIRED.REMOTE_HITM	Retired load instructions whose data sources was remote HITM.	Precise event capable.
D3H	08H	MEM_LOAD_L3_MISS_RETIRED.REMOTE_FWD	Retired load instructions whose data sources was forwarded from a remote cache.	

**Table 19-3. Performance Events of the Processor Core Supported in Intel® Xeon® Processor Scalable Family with Skylake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D4H	04H	MEM_LOAD_MISC_RETIRED.UC	Retired instructions with at least 1 uncacheable load or lock.	Precise event capable.
E6H	01H	BACLEARS.ANY	Counts the number of times the front-end is resteered when it finds a branch instruction in a fetch line. This occurs for the first time a branch instruction is fetched or when the branch is not tracked by the BPU (Branch Prediction Unit) anymore.	
FOH	40H	L2_TRANS.L2_WB	Counts L2 writebacks that access L2 cache.	
F1H	1FH	L2_LINES_IN.ALL	Counts the number of L2 cache lines filling the L2. Counting does not cover rejects.	
F2H	01H	L2_LINES_OUT.SILENT	Counts the number of lines that are silently dropped by L2 cache when triggered by an L2 cache fill. These lines are typically in Shared state. A non-threaded event.	
F2H	02H	L2_LINES_OUT.NON_SILENT	Counts the number of lines that are evicted by L2 cache when triggered by an L2 cache fill. Those lines can be either in modified state or clean state. Modified lines may either be written back to L3 or directly written to memory and not allocated in L3. Clean lines may either be allocated in L3 or dropped.	
F2H	04H	L2_LINES_OUT.USELESS_PREF	Counts the number of lines that have been hardware prefetched but not used and now evicted by L2 cache.	
F2H	04H	L2_LINES_OUT.USELESS_HWPF	Counts the number of lines that have been hardware prefetched but not used and now evicted by L2 cache.	
F4H	10H	SQ_MISC.SPLIT_LOCK	Counts the number of cache line split locks sent to the uncore.	
FEH	02H	IDI_MISC.WB_UPGRADE	Counts number of cache lines that are allocated and written back to L3 with the intention that they are more likely to be reused shortly.	
FEH	04H	IDI_MISC.WB_DOWNGRADE	Counts number of cache lines that are dropped and not written back to L3 as they are deemed to be less likely to be reused shortly.	

## 19.3 PERFORMANCE MONITORING EVENTS FOR 6TH GENERATION INTEL® CORE™ PROCESSOR AND 7TH GENERATION INTEL® CORE™ PROCESSOR

6th Generation Intel® Core™ processors are based on the Skylake microarchitecture. They support the architectural performance monitoring events listed in Table 19-1. Fixed counters in the core PMU support the architecture events defined in Table 19-2. Model-specific performance monitoring events in the processor core are listed in Table 19-4. The events in Table 19-4 apply to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following values: 06\_4EH and 06\_5EH. Table 19-10 lists performance events supporting Intel TSX (see Section 18.3.6.5) and the events are applicable to processors based on Skylake microarchitecture. Where Skylake microarchitecture implements TSX-related event semantics that differ from Table 19-10, they are listed in Table 19-5.

7th Generation Intel® Core™ processors are based on the Kaby Lake microarchitecture. Model-specific performance monitoring events in the processor core are listed in Table 19-4. The events in Table 19-4 apply to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following values: 06\_8EH and 06\_9EH.

The comment column in Table 19-4 uses abbreviated letters to indicate additional conditions applicable to the Event Mask Mnemonic. For event umasks listed in Table 19-4 that do not show "AnyT", users should refrain from programming "AnyThread = 1" in IA32\_PERF\_EVTSELx.

**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture**

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.	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are temporarily blocked because all resources for handling the split accesses are in use.	
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	Load misses in all TLB levels that cause a page walk of any page size.	
08H	0EH	DTLB_LOAD_MISSES.WALK_COMPLETED	Load misses in all TLB levels causes a page walk that completes. (All page sizes.)	
08H	10H	DTLB_LOAD_MISSES.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a page walk for a load.	
08H	10H	DTLB_LOAD_MISSES.WALK_ACTIVE	Cycles when at least one PMH is busy with a walk for a load.	CMSK1
08H	20H	DTLB_LOAD_MISSES.STLB_HIT	Loads that miss the DTLB but hit STLB.	
0DH	01H	INT_MISC.RECOVERY_CYCLES	Core cycles the allocator was stalled due to recovery from earlier machine clear event for this thread (for example, misprediction or memory order conflict).	
0DH	01H	INT_MISC.RECOVERY_CYCLES_ANY	Core cycles the allocator was stalled due to recovery from earlier machine clear event for any logical thread in this processor core.	AnyT
0DH	80H	INT_MISC.CLEAR_RESTEER_CYCLES	Cycles the issue-stage is waiting for front end to fetch from resteeered path following branch misprediction or machine clear events.	
0EH	01H	UOPS_ISSUED.ANY	The number of uops issued by the RAT to RS.	
0EH	01H	UOPS_ISSUED.STALL_CYCLES	Cycles when the RAT does not issue uops to RS for the thread.	CMSK1, INV
0EH	02H	UOPS_ISSUED.VECTOR_WIDTH_MISMATCH	Uops inserted at issue-stage in order to preserve upper bits of vector registers.	
0EH	20H	UOPS_ISSUED.SLOW_LEA	Number of slow LEA or similar uops allocated. Such uop has 3 sources (for example, 2 sources + immediate) regardless of whether it is a result of LEA instruction or not.	
14H	01H	ARITH.FPU_DIVIDER_ACTIVE	Cycles when divider is busy executing divide or square root operations. Accounts for FP operations including integer divides.	
24H	21H	L2_RQSTS.DEMAND_DATA_RD_MISS	Demand Data Read requests that missed L2, no rejects.	
24H	22H	L2_RQSTS.RFO_MISS	RFO requests that missed L2.	
24H	24H	L2_RQSTS.CODE_RD_MISS	L2 cache misses when fetching instructions.	
24H	27H	L2_RQSTS.ALL_DEMAND_MISS	Demand requests that missed L2.	

**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	38H	L2_RQSTS.PF_MISS	Requests from the L1/L2/L3 hardware prefetchers or load software prefetches that miss L2 cache.	
24H	3FH	L2_RQSTS.MISS	All requests that missed L2.	
24H	41H	L2_RQSTS.DEMAND_DATA_RD_HIT	Demand Data Read requests that hit L2 cache.	
24H	42H	L2_RQSTS.RFO_HIT	RFO requests that hit L2 cache.	
24H	44H	L2_RQSTS.CODE_RD_HIT	L2 cache hits when fetching instructions.	
24H	D8H	L2_RQSTS.PF_HIT	Prefetches that hit L2.	
24H	E1H	L2_RQSTS.ALL_DEMAND_DATA_RD	All demand data read requests to L2.	
24H	E2H	L2_RQSTS.ALL_RFO	All L RFO requests to L2.	
24H	E4H	L2_RQSTS.ALL_CODE_RD	All L2 code requests.	
24H	E7H	L2_RQSTS.ALL_DEMAND_REFERENCES	All demand requests to L2.	
24H	F8H	L2_RQSTS.ALL_PF	All requests from the L1/L2/L3 hardware prefetchers or load software prefetches.	
24H	EFH	L2_RQSTS.REFERENCES	All requests to L2.	
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the L3 cache.	See Table 19-1.
2EH	41H	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the L3 cache.	See Table 19-1.
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Cycles while the logical processor is not in a halt state.	See Table 19-1.
3CH	00H	CPU_CLK_UNHALTED.THREAD_P_ANY	Cycles while at least one logical processor is not in a halt state.	AnyT
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK	Core crystal clock cycles when the thread is unhalted.	See Table 19-1.
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK_ANY	Core crystal clock cycles when at least one thread on the physical core is unhalted.	AnyT
3CH	02H	CPU_CLK_THREAD_UNHALTED.ONE_THREAD_ACTIVE	Core crystal clock cycles when this thread is unhalted and the other thread is halted.	
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle.	
48H	01H	L1D_PEND_MISS.PENDING_CYCLES	Cycles with at least one outstanding L1D misses from this logical processor.	CMSK1
48H	01H	L1D_PEND_MISS.PENDING_CYCLES_ANY	Cycles with at least one outstanding L1D misses from any logical processor in this core.	CMSK1, AnyT
48H	02H	L1D_PEND_MISS.FB_FULL	Number of times a request needed a FB entry but there was no entry available for it. That is, the FB unavailability was the dominant reason for blocking the request. A request includes cacheable/uncacheable demand that is load, store or SW prefetch. HWP are excluded.	
49H	01H	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Store misses in all TLB levels that cause page walks.	

**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
49H	0EH	DTLB_STORE_MISSES.WALK_COMPLETED	Counts completed page walks in any TLB levels due to store misses (all page sizes).	
49H	10H	DTLB_STORE_MISSES.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a page walk for a store.	
49H	10H	DTLB_STORE_MISSES.WALK_ACTIVE	Cycles when at least one PMH is busy with a page walk for a store.	CMSK1
49H	20H	DTLB_STORE_MISSES.STLB_HIT	Store misses that missed DTLB but hit STLB.	
4CH	01H	LOAD_HIT_PRE.HW_PF	Demand load dispatches that hit fill buffer allocated for software prefetch.	
4FH	10H	EPT.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with an EPT walk for any request type.	
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
5EH	01H	RS_EVENTS.EMPTY_CYCLES	Cycles the RS is empty for the thread.	
5EH	01H	RS_EVENTS.EMPTY_END	Counts end of periods where the Reservation Station (RS) was empty. Could be useful to precisely locate Front-end Latency Bound issues.	CMSK1, INV
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Increment each cycle of the number of offcore outstanding Demand Data Read transactions in SQ to uncure.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_DATA_RD	Cycles with at least one offcore outstanding Demand Data Read transactions in SQ to uncure.	CMSK1
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD_GE_6	Cycles with at least 6 offcore outstanding Demand Data Read transactions in SQ to uncure.	CMSK6
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD	Increment each cycle of the number of offcore outstanding demand code read transactions in SQ to uncure.	
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_CODE_RD	Cycles with at least one offcore outstanding demand code read transactions in SQ to uncure.	CMSK1
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Increment each cycle of the number of offcore outstanding RFO store transactions in SQ to uncure. Set Cmask=1 to count cycles.	
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_RFO	Cycles with at least one offcore outstanding RFO transactions in SQ to uncure.	CMSK1
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Increment each cycle of the number of offcore outstanding cacheable data read transactions in SQ to uncure. Set Cmask=1 to count cycles.	
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DATA_RD	Cycles with at least one offcore outstanding data read transactions in SQ to uncure.	CMSK1
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.L3_MISS_DEMAND_DATA_RD	Increment each cycle of the number of offcore outstanding demand data read requests from SQ that missed L3.	



**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_L3_MISS_DEMAND_DATA_RD	Cycles with at least one offcore outstanding demand data read requests from SQ that missed L3.	CMSK1
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.L3_MISS_DEMAND_DATA_RD_GE_6	Cycles with at least one offcore outstanding demand data read requests from SQ that missed L3.	CMSK6
63H	02H	LOCK_CYCLES.CACHE_LOCK_DURATION	Cycles in which the L1D is locked.	
79H	04H	IDQ.MITE_UOPS	Increment each cycle # of uops delivered to IDQ from MITE path.	
79H	04H	IDQ.MITE_CYCLES	Cycles when uops are being delivered to IDQ from MITE path.	CMSK1
79H	08H	IDQ.DSB_UOPS	Increment each cycle. # of uops delivered to IDQ from DSB path.	
79H	08H	IDQ.DSB_CYCLES	Cycles when uops are being delivered to IDQ from DSB path.	CMSK1
79H	10H	IDQ.MS_DSB_UOPS	Increment each cycle # of uops delivered to IDQ by DSB when MS_busy.	
79H	18H	IDQ.ALL_DSB_CYCLES_ANY_UOPS	Cycles DSB is delivered at least one uops.	CMSK1
79H	18H	IDQ.ALL_DSB_CYCLES_4_UOPS	Cycles DSB is delivered four uops.	CMSK4
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ by MITE when MS_busy.	
79H	24H	IDQ.ALL_MITE_CYCLES_ANY_UOPS	Counts cycles MITE is delivered at least one uops.	CMSK1
79H	24H	IDQ.ALL_MITE_CYCLES_4_UOPS	Counts cycles MITE is delivered four uops.	CMSK4
79H	30H	IDQ.MS_UOPS	Increment each cycle # of uops delivered to IDQ while MS is busy.	
79H	30H	IDQ.MS_SWITCHES	Number of switches from DSB or MITE to MS.	EDG
79H	30H	IDQ.MS_CYCLES	Cycles MS is delivered at least one uops.	CMSK1
80H	04H	ICACHE_16B.IFDATA_STALL	Cycles where a code fetch is stalled due to L1 instruction cache miss.	
80H	04H	ICACHE_64B.IFDATA_STALL	Cycles where a code fetch is stalled due to L1 instruction cache tag miss.	
83H	01H	ICACHE_64B.IFTAG_HIT	Instruction fetch tag lookups that hit in the instruction cache (L1). Counts at 64-byte cache-line granularity.	
83H	02H	ICACHE_64B.IFTAG_MISS	Instruction fetch tag lookups that miss in the instruction cache (L1). Counts at 64-byte cache-line granularity.	
85H	01H	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses at all ITLB levels that cause page walks.	
85H	0EH	ITLB_MISSES.WALK_COMPLETED	Counts completed page walks in any TLB level due to code fetch misses (all page sizes).	
85H	10H	ITLB_MISSES.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a page walk for an instruction fetch request.	



**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
85H	20H	ITLB_MISSES.STLB_HIT	ITLB misses that hit STLB.	
87H	01H	ILD_STALL.LCP	Stalls caused by changing prefix length of the instruction.	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CORE	Count issue pipeline slots where no uop was delivered from the front end to the back end when there is no back-end stall.	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_0_UOP_DELIV.CORE	Cycles which 4 issue pipeline slots had no uop delivered from the front end to the back end when there is no back-end stall.	CMSK4
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_LE_n_UOP_DELIV.CORE	Cycles which "4-n" issue pipeline slots had no uop delivered from the front end to the back end when there is no back-end stall.	Set CMSK = 4-n; n = 1, 2, 3
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_FE_WAS_OK	Cycles which front end delivered 4 uops or the RAT was stalling FE.	CMSK, INV
A1H	01H	UOPS_DISPATCHED_PORT.PORT_0	Counts the number of cycles in which a uop is dispatched to port 0.	
A1H	02H	UOPS_DISPATCHED_PORT.PORT_1	Counts the number of cycles in which a uop is dispatched to port 1.	
A1H	04H	UOPS_DISPATCHED_PORT.PORT_2	Counts the number of cycles in which a uop is dispatched to port 2.	
A1H	08H	UOPS_DISPATCHED_PORT.PORT_3	Counts the number of cycles in which a uop is dispatched to port 3.	
A1H	10H	UOPS_DISPATCHED_PORT.PORT_4	Counts the number of cycles in which a uop is dispatched to port 4.	
A1H	20H	UOPS_DISPATCHED_PORT.PORT_5	Counts the number of cycles in which a uop is dispatched to port 5.	
A1H	40H	UOPS_DISPATCHED_PORT.PORT_6	Counts the number of cycles in which a uop is dispatched to port 6.	
A1H	80H	UOPS_DISPATCHED_PORT.PORT_7	Counts the number of cycles in which a uop is dispatched to port 7.	
A2H	01H	RESOURCE_STALLS.ANY	Resource-related stall cycles.	
A2H	08H	RESOURCE_STALLS.SB	Cycles stalled due to no store buffers available (not including draining from sync).	
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_MISS	Cycles while L2 cache miss demand load is outstanding.	CMSK1
A3H	02H	CYCLE_ACTIVITY.CYCLES_L3_MISS	Cycles while L3 cache miss demand load is outstanding.	CMSK2
A3H	04H	CYCLE_ACTIVITY.STALLS_TOTAL	Total execution stalls.	CMSK4
A3H	05H	CYCLE_ACTIVITY.STALLS_L2_MISS	Execution stalls while L2 cache miss demand load is outstanding.	CMSK5
A3H	06H	CYCLE_ACTIVITY.STALLS_L3_MISS	Execution stalls while L3 cache miss demand load is outstanding.	CMSK6
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_MISS	Cycles while L1 data cache miss demand load is outstanding.	CMSK8
A3H	0CH	CYCLE_ACTIVITY.STALLS_L1D_MISS	Execution stalls while L1 data cache miss demand load is outstanding.	CMSK12

**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A3H	10H	CYCLE_ACTIVITY.CYCLES_MEM_ANY	Cycles while memory subsystem has an outstanding load.	CMSK16
A3H	14H	CYCLE_ACTIVITY.STALLS_MEM_ANY	Execution stalls while memory subsystem has an outstanding load.	CMSK20
A6H	01H	EXE_ACTIVITY.EXE_BOUND_0_PORTS	Cycles for which no uops began execution, the Reservation Station was not empty, the Store Buffer was full and there was no outstanding load.	
A6H	02H	EXE_ACTIVITY.1_PORTS_UTIL	Cycles for which one uop began execution on any port, and the Reservation Station was not empty.	
A6H	04H	EXE_ACTIVITY.2_PORTS_UTIL	Cycles for which two uops began execution, and the Reservation Station was not empty.	
A6H	08H	EXE_ACTIVITY.3_PORTS_UTIL	Cycles for which three uops began execution, and the Reservation Station was not empty.	
A6H	04H	EXE_ACTIVITY.4_PORTS_UTIL	Cycles for which four uops began execution, and the Reservation Station was not empty.	
A6H	40H	EXE_ACTIVITY.BOUND_ON_STORES	Cycles where the Store Buffer was full and no outstanding load.	
A8H	01H	LSD.UOPS	Number of uops delivered by the LSD.	
A8H	01H	LSD.CYCLES_ACTIVE	Cycles with at least one uop delivered by the LSD and none from the decoder.	CMSK1
A8H	01H	LSD.CYCLES_4_UOPS	Cycles with 4 uops delivered by the LSD and none from the decoder.	CMSK4
ABH	02H	DSB2MITE_SWITCHES.PENALTY_CYCLES	DSB-to-MITE switch true penalty cycles.	
AEH	01H	ITLB.ITLB_FLUSH	Flushing of the Instruction TLB (ITLB) pages, includes 4k/2M/4M pages.	
B0H	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data read requests sent to uncore.	
B0H	02H	OFFCORE_REQUESTS.DEMAND_CODE_RD	Demand code read requests sent to uncore.	
B0H	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ItoM.	
B0H	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	
B0H	10H	OFFCORE_REQUESTS.L3_MISS_DEMAND_DATA_RD	Demand data read requests that missed L3.	
B0H	80H	OFFCORE_REQUESTS.ALL_REQUESTS	Any memory transaction that reached the SQ.	
B1H	01H	UOPS_EXECUTED.THREAD	Counts the number of uops that begin execution across all ports.	
B1H	01H	UOPS_EXECUTED.STALL_CYCLES	Cycles where there were no uops that began execution.	CMSK, INV
B1H	01H	UOPS_EXECUTED.CYCLES_GE_1_UOP_EXEC	Cycles where there was at least one uop that began execution.	CMSK1
B1H	01H	UOPS_EXECUTED.CYCLES_GE_2_UOP_EXEC	Cycles where there were at least two uops that began execution.	CMSK2

**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B1H	01H	UOPS_EXECUTED.CYCLES_GE_3_UOP_EXEC	Cycles where there were at least three uops that began execution.	CMSK3
B1H	01H	UOPS_EXECUTED.CYCLES_GE_4_UOP_EXEC	Cycles where there were at least four uops that began execution.	CMSK4
B1H	02H	UOPS_EXECUTED.CORE	Counts the number of uops from any logical processor in this core that begin execution.	
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_1	Cycles where there was at least one uop, from any logical processor in this core, that began execution.	CMSK1
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_2	Cycles where there were at least two uops, from any logical processor in this core, that began execution.	CMSK2
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_3	Cycles where there were at least three uops, from any logical processor in this core, that began execution.	CMSK3
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_4	Cycles where there were at least four uops, from any logical processor in this core, that began execution.	CMSK4
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_NONE	Cycles where there were no uops from any logical processor in this core that began execution.	CMSK1, INV
B1H	10H	UOPS_EXECUTED.X87	Counts the number of X87 uops that begin execution.	
B2H	01H	OFF_CORE_REQUEST_BUFFER.SQ_FULL	Offcore requests buffer cannot take more entries for this core.	
B7H	01H	OFF_CORE_RESPONSE_0	See Section 18.3.4.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.3.4.5, "Off-core Response Performance Monitoring".	Requires MSR 01A7H
BDH	01H	TLB_FLUSH.DTLB_THREAD	DTLB flush attempts of the thread-specific entries.	
BDH	01H	TLB_FLUSH.STLB_ANY	STLB flush attempts.	
COH	00H	INST_RETIRED.ANY_P	Number of instructions at retirement.	See Table 19-1.
COH	01H	INST_RETIRED.PREC_DIST	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution.	PMC1 only;
COH	01H	INST_RETIRED.TOTAL_CYCLES	Number of cycles using always true condition applied to PEBS instructions retired event.	CMSK10, PS
C1H	3FH	OTHER_ASSISTS.ANY	Number of times a microcode assist is invoked by HW other than FP-assist. Examples include AD (page Access Dirty) and AVX* related assists.	
C2H	01H	UOPS_RETIRED.STALL_CYCLES	Cycles without actually retired uops.	CMSK1, INV
C2H	01H	UOPS_RETIRED.TOTAL_CYCLES	Cycles with less than 10 actually retired uops.	CMSK10, INV
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Retirement slots used.	
C3H	01H	MACHINE_CLEAR.SMC	Number of machine clears of any type.	CMSK1, EDG
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.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions that retired.	See Table 19-1.
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	PS

**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Direct and indirect near call instructions retired.	PS
C4H	04H	BR_INST_RETIRED.ALL_BRANC HES	Counts the number of branch instructions retired.	PS
C4H	08H	BR_INST_RETIRED.NEAR_RETU RN	Counts the number of near return instructions retired.	PS
C4H	10H	BR_INST_RETIRED.NOT_TAKEN	Counts the number of not taken branch instructions retired.	
C4H	20H	BR_INST_RETIRED.NEAR_TAKE N	Number of near taken branches retired.	PS
C4H	40H	BR_INST_RETIRED.FAR_BRANC H	Number of far branches retired.	PS
C5H	00H	BR_MISP_RETIRED.ALL_BRANC HES	Mispredicted branch instructions at retirement.	See Table 19-1.
C5H	01H	BR_MISP_RETIRED.CONDITIONA L	Mispredicted conditional branch instructions retired.	PS
C5H	04H	BR_MISP_RETIRED.ALL_BRANC HES	Mispredicted macro branch instructions retired.	PS
C5H	20H	BR_MISP_RETIRED.NEAR_TAKE N	Number of near branch instructions retired that were mispredicted and taken.	PS
C6H	01H	FRONTEND_RETIRED.DSB_MISS	Retired instructions which experienced DSB miss. Specify MSR_PEBS_FRONTEND.EVTSEL=11H.	PS
C6H	01H	FRONTEND_RETIRED.L1I_MISS	Retired instructions which experienced instruction L1 cache true miss. Specify MSR_PEBS_FRONTEND.EVTSEL=12H.	PS
C6H	01H	FRONTEND_RETIRED.L2_MISS	Retired instructions which experienced L2 cache true miss. Specify MSR_PEBS_FRONTEND.EVTSEL=13H.	PS
C6H	01H	FRONTEND_RETIRED.ITLB_MISS	Retired instructions which experienced ITLB true miss. Specify MSR_PEBS_FRONTEND.EVTSEL=14H.	PS
C6H	01H	FRONTEND_RETIRED.STLB_MIS S	Retired instructions which experienced STLB true miss. Specify MSR_PEBS_FRONTEND.EVTSEL=15H.	PS
C6H	01H	FRONTEND_RETIRED.LATENCY_ GE_16	Retired instructions that are fetched after an interval where the front end delivered no uops for at least 16 cycles. Specify the following fields in MSR_PEBS_FRONTEND: EVTSEL=16H, IDQ_Bubble_Length =16, IDQ_Bubble_Width = 4.	PS
C6H	01H	FRONTEND_RETIRED.LATENCY_ GE_2_BUBBLES_GE_m	Retired instructions that are fetched after an interval where the front end had 'm' IDQ slots delivered, no uops for at least 2 cycles. Specify the following fields in MSR_PEBS_FRONTEND: EVTSEL=16H, IDQ_Bubble_Length =2, IDQ_Bubble_Width = m.	PS, m = 1, 2, 3
C7H	01H	FP_ARITH_INST_RETIRED.SCAL AR_DOUBLE	Number of double-precision, floating-point, scalar SSE/AVX computational instructions that are retired. Each scalar FMA instruction counts as 2.	Software may treat each count as one DP FLOP.
C7H	02H	FP_ARITH_INST_RETIRED.SCAL AR_SINGLE	Number of single-precision, floating-point, scalar SSE/AVX computational instructions that are retired. Each scalar FMA instruction counts as 2.	Software may treat each count as one SP FLOP.

**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C7H	04H	FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE	Number of double-precision, floating-point, 128-bit SSE/AVX computational instructions that are retired. Each 128-bit FMA or (V)DPPD instruction counts as 2.	Software may treat each count as two DP FLOPs.
C7H	08H	FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE	Number of single-precision, floating-point, 128-bit SSE/AVX computational instructions that are retired. Each 128-bit FMA or (V)DPPS instruction counts as 2.	Software may treat each count as four SP FLOPs.
C7H	10H	FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE	Number of double-precision, floating-point, 256-bit SSE/AVX computational instructions that are retired. Each 256-bit FMA instruction counts as 2.	Software may treat each count as four DP FLOPs.
C7H	20H	FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE	Number of single-precision, floating-point, 256-bit SSE/AVX computational instructions that are retired. Each 256-bit FMA or VDPPS instruction counts as 2.	Software may treat each count as eight SP FLOPs.
CAH	1EH	FP_ASSIST.ANY	Cycles with any input/output SSE* or FP assists.	CMSK1
CBH	01H	HW_INTERRUPTS.RECEIVED	Number of hardware interrupts received by the processor.	
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY	Randomly sampled loads whose latency is above a user defined threshold. A small fraction of the overall loads are sampled due to randomization.	Specify threshold in MSR 3F6H. PSDLA
DOH	11H	MEM_INST_RETIRED.STLB_MISS_LOADS	Retired load instructions that miss the STLB.	PSDLA
DOH	12H	MEM_INST_RETIRED.STLB_MISS_STORES	Retired store instructions that miss the STLB.	PSDLA
DOH	21H	MEM_INST_RETIRED.LOCK_LOADS	Retired load instructions with locked access.	PSDLA
DOH	41H	MEM_INST_RETIRED.SPLIT_LOADS	Number of load instructions retired with cache-line splits that may impact performance.	PSDLA
DOH	42H	MEM_INST_RETIRED.SPLIT_STORES	Number of store instructions retired with line-split.	PSDLA
DOH	81H	MEM_INST_RETIRED.ALL_LOADS	All retired load instructions.	PSDLA
DOH	82H	MEM_INST_RETIRED.ALL_STORES	All retired store instructions.	PSDLA
D1H	01H	MEM_LOAD_RETIRED.L1_HIT	Retired load instructions with L1 cache hits as data sources.	PSDLA
D1H	02H	MEM_LOAD_RETIRED.L2_HIT	Retired load instructions with L2 cache hits as data sources.	PSDLA
D1H	04H	MEM_LOAD_RETIRED.L3_HIT	Retired load instructions with L3 cache hits as data sources.	PSDLA
D1H	08H	MEM_LOAD_RETIRED.L1_MISS	Retired load instructions missed L1 cache as data sources.	PSDLA
D1H	10H	MEM_LOAD_RETIRED.L2_MISS	Retired load instructions missed L2. Unknown data source excluded.	PSDLA
D1H	20H	MEM_LOAD_RETIRED.L3_MISS	Retired load instructions missed L3. Excludes unknown data source.	PSDLA

**Table 19-4. Performance Events of the Processor Core Supported by Skylake Microarchitecture and Kaby Lake Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D1H	40H	MEM_LOAD_RETIRED.FB_HIT	Retired load instructions where data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.	PSDLA
D2H	01H	MEM_LOAD_L3_HIT_RETIRED.X_SNP_MISS	Retired load instructions where data sources were L3 hit and cross-core snoop missed in on-pkg core cache.	PSDLA
D2H	02H	MEM_LOAD_L3_HIT_RETIRED.X_SNP_HIT	Retired load Instructions where data sources were L3 and cross-core snoop hits in on-pkg core cache.	PSDLA
D2H	04H	MEM_LOAD_L3_HIT_RETIRED.X_SNP_HITM	Retired load instructions where data sources were HitM responses from shared L3.	PSDLA
D2H	08H	MEM_LOAD_L3_HIT_RETIRED.X_SNP_NONE	Retired load instructions where data sources were hits in L3 without snoops required.	PSDLA
E6H	01H	BACLEAR.S.ANY	Number of front end re-steers due to BPU misprediction.	
FOH	40H	L2_TRANS.L2_WB	L2 writebacks that access L2 cache.	
F1H	07H	L2_LINES_IN.ALL	L2 cache lines filling L2.	
<p>CMSK1: CounterMask = 1 required; CMSK4: CounterMask = 4 required; CMSK6: CounterMask = 6 required; CMSK8: CounterMask = 8 required; CMSK10: CounterMask = 10 required; CMSK12: CounterMask = 12 required; CMSK16: CounterMask = 16 required; CMSK20: CounterMask = 20 required.</p> <p>AnyT: AnyThread = 1 required.</p> <p>INV: Invert = 1 required.</p> <p>EDG: EDGE = 1 required.</p> <p>PSDLA: Also supports PEBS and DataLA.</p> <p>PS: Also supports PEBS.</p>				

Table 19-10 lists performance events supporting Intel TSX (see Section 18.3.6.5) and the events are applicable to processors based on Skylake microarchitecture. Where Skylake microarchitecture implements TSX-related event semantics that differ from Table 19-10, they are listed in Table 19-5.

**Table 19-5. Intel® TSX Performance Event Addendum in Processors based on Skylake Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
54H	02H	TX_MEM.ABORT_CAPACITY	Number of times a transactional abort was signaled due to a data capacity limitation for transactional reads or writes.	

## 19.4 PERFORMANCE MONITORING EVENTS FOR INTEL® XEON PHI™ PROCESSOR 3200, 5200, 7200 SERIES

Intel® Xeon Phi™ processors 3200/5200/7200 series are based on the Knights Landing microarchitecture. Model-specific performance monitoring events in the processor core are listed in Table 19-6. The events in Table 19-6 apply to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following value 06\_57H.

**Table 19-6. Performance Events of the Processor Core Supported by Knights Landing Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	01H	RECYCLEQ.LD_BLOCK_ST_FORWARD	Counts the number of occurrences a retired load gets blocked because its address partially overlaps with a store.	PSDLA
03H	02H	RECYCLEQ.LD_BLOCK_STD_NOT_READY	Counts the number of occurrences a retired load gets blocked because its address overlaps with a store whose data is not ready.	
03H	04H	RECYCLEQ.ST_SPLITS	Counts the number of occurrences a retired store that is a cache line split. Each split should be counted only once.	
03H	08H	RECYCLEQ.LD_SPLITS	Counts the number of occurrences a retired load that is a cache line split. Each split should be counted only once.	PSDLA
03H	10H	RECYCLEQ.LOCK	Counts all the retired locked loads. It does not include stores because we would double count if we count stores.	
03H	20H	RECYCLEQ.STA_FULL	Counts the store micro-ops retired that were pushed in the recycle queue because the store address buffer is full.	
03H	40H	RECYCLEQ.ANY_LD	Counts any retired load that was pushed into the recycle queue for any reason.	
03H	80H	RECYCLEQ.ANY_ST	Counts any retired store that was pushed into the recycle queue for any reason.	
04H	01H	MEM_UOPS_RETIRED.L1_MISS_LOADS	Counts the number of load micro-ops retired that miss in L1 D cache.	
04H	02H	MEM_UOPS_RETIRED.L2_HIT_LOADS	Counts the number of load micro-ops retired that hit in the L2.	PSDLA
04H	04H	MEM_UOPS_RETIRED.L2_MISS_LOADS	Counts the number of load micro-ops retired that miss in the L2.	PSDLA
04H	08H	MEM_UOPS_RETIRED.DTLB_MISSES_LOADS	Counts the number of load micro-ops retired that cause a DTLB miss.	PSDLA
04H	10H	MEM_UOPS_RETIRED.UTLB_MISSES_LOADS	Counts the number of load micro-ops retired that caused micro TLB miss.	
04H	20H	MEM_UOPS_RETIRED.HITM	Counts the loads retired that get the data from the other core in the same tile in M state.	
04H	40H	MEM_UOPS_RETIRED.ALL_LOADS	Counts all the load micro-ops retired.	
04H	80H	MEM_UOPS_RETIRED.ALL_STORES	Counts all the store micro-ops retired.	
05H	01H	PAGE_WALKS.D_SIDE_WALKS	Counts the total D-side page walks that are completed or started. The page walks started in the speculative path will also be counted.	EdgeDetect=1
05H	01H	PAGE_WALKS.D_SIDE_CYCLES	Counts the total number of core cycles for all the D-side page walks. The cycles for page walks started in speculative path will also be included.	
05H	02H	PAGE_WALKS.I_SIDE_WALKS	Counts the total I-side page walks that are completed.	EdgeDetect=1



**Table 19-6. Performance Events of the Processor Core Supported by Knights Landing Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
05H	02H	PAGE_WALKS.I_SIDE_CYCLES	Counts the total number of core cycles for all the I-side page walks. The cycles for page walks started in speculative path will also be included.	
05H	03H	PAGE_WALKS.WALKS	Counts the total page walks that are completed (I-side and D-side).	EdgeDetect=1
05H	03H	PAGE_WALKS.CYCLES	Counts the total number of core cycles for all the page walks. The cycles for page walks started in speculative path will also be included.	
2EH	41H	LONGEST_LAT_CACHE.MISS	Counts the number of L2 cache misses. Also called L2_REQUESTS_MISS.	
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	Counts the total number of L2 cache references. Also called L2_REQUESTS_REFERENCE.	
30H	00H	L2_REQUESTS_REJECT.ALL	Counts the number of MEC requests from the L2Q that reference a cache line (cacheable requests) excluding SW prefetches filling only to L2 cache and L1 evictions (automatically excludes L2HWP, UC, WC) that were rejected - Multiple repeated rejects should be counted multiple times.	
31H	00H	CORE_REJECT_L2Q.ALL	Counts the number of MEC requests that were not accepted into the L2Q because of any L2 queue reject condition. There is no concept of at-ret here. It might include requests due to instructions in the speculative path.	
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Counts the number of unhalted core clock cycles.	
3CH	01H	CPU_CLK_UNHALTED.REF	Counts the number of unhalted reference clock cycles.	
3EH	04H	L2_PREFETCHER.ALLOC_XQ	Counts the number of L2HWP allocated into XQ GP.	
80H	01H	ICACHE.HIT	Counts all instruction fetches that hit the instruction cache.	
80H	02H	ICACHE.MISSES	Counts all instruction fetches that miss the instruction cache or produce memory requests. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.	
80H	03H	ICACHE.ACCESSSES	Counts all instruction fetches, including uncacheable fetches.	
86H	04H	FETCH_STALL.ICACHE_FILL_PENDING_CYCLES	Counts the number of core cycles the fetch stalls because of an icache miss. This is a cumulative count of core cycles the fetch stalled for all icache misses.	
B7H	01H	OFFCORE_RESPONSE_0	See Section 18.4.1.1.2.	Requires MSR_OFFCORE_RESP 0 to specify request type and response.
B7H	02H	OFFCORE_RESPONSE_1	See Section 18.4.1.1.2.	Requires MSR_OFFCORE_RESP 1 to specify request type and response.
COH	00H	INST_RETIRED.ANY_P	Counts the total number of instructions retired.	PS



**Table 19-6. Performance Events of the Processor Core Supported by Knights Landing Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C2H	01H	UOPS_RETIRED.MS	Counts the number of micro-ops retired that are from the complex flows issued by the micro-sequencer (MS).	
C2H	10H	UOPS_RETIRED.ALL	Counts the number of micro-ops retired.	
C2H	20H	UOPS_RETIRED.SCALAR_SIMD	Counts the number of scalar SSE, AVX, AVX2, and AVX-512 micro-ops except for loads (memory-to-register mov-type micro ops), division and sqrt.	
C2H	40H	UOPS_RETIRED.PACKED_SIMD	Counts the number of packed SSE, AVX, AVX2, and AVX-512 micro-ops (both floating point and integer) except for loads (memory-to-register mov-type micro-ops), packed byte and word multiplies.	
C3H	01H	MACHINE_CLEARS.SMC	Counts the number of times that the machine clears due to program modifying data within 1K of a recently fetched code page.	
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Counts the number of times the machine clears due to memory ordering hazards.	
C3H	04H	MACHINE_CLEARS.FP_ASSIST	Counts the number of floating operations retired that required microcode assists.	
C3H	08H	MACHINE_CLEARS.ALL	Counts all machine clears.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	PS
C4H	7EH	BR_INST_RETIRED.JCC	Counts the number of JCC branch instructions retired.	PS
C4H	BFH	BR_INST_RETIRED.FAR_BRANCH	Counts the number of far branch instructions retired.	PS
C4H	EBH	BR_INST_RETIRED.NON_RETURN_IND	Counts the number of branch instructions retired that were near indirect CALL or near indirect JMP.	PS
C4H	F7H	BR_INST_RETIRED.RETURN	Counts the number of near RET branch instructions retired.	PS
C4H	F9H	BR_INST_RETIRED.CALL	Counts the number of near CALL branch instructions retired.	PS
C4H	FBH	BR_INST_RETIRED.IND_CALL	Counts the number of near indirect CALL branch instructions retired.	PS
C4H	FDH	BR_INST_RETIRED.REL_CALL	Counts the number of near relative CALL branch instructions retired.	PS
C4H	FEH	BR_INST_RETIRED.TAKEN_JCC	Counts the number of branch instructions retired that were taken conditional jumps.	PS
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Counts the number of mispredicted branch instructions retired.	PS
C5H	7EH	BR_MISP_RETIRED.JCC	Counts the number of mispredicted JCC branch instructions retired.	PS
C5H	BFH	BR_MISP_RETIRED.FAR_BRANCH	Counts the number of mispredicted far branch instructions retired.	PS
C5H	EBH	BR_MISP_RETIRED.NON_RETURN_IND	Counts the number of mispredicted branch instructions retired that were near indirect CALL or near indirect JMP.	PS
C5H	F7H	BR_MISP_RETIRED.RETURN	Counts the number of mispredicted near RET branch instructions retired.	PS

**Table 19-6. Performance Events of the Processor Core Supported by Knights Landing Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C5H	F9H	BR_MISP_RETIREDCALL	Counts the number of mispredicted near CALL branch instructions retired.	PS
C5H	FBH	BR_MISP_RETIREDCALL	Counts the number of mispredicted near indirect CALL branch instructions retired.	PS
C5H	FDH	BR_MISP_RETIREDCALL	Counts the number of mispredicted near relative CALL branch instructions retired.	PS
C5H	FEH	BR_MISP_RETIREDTAKEN_JCC	Counts the number of mispredicted branch instructions retired that were taken conditional jumps.	PS
CAH	01H	NO_ALLOC_CYCLES.ROB_FULL	Counts the number of core cycles when no micro-ops are allocated and the ROB is full.	
CAH	04H	NO_ALLOC_CYCLES.MISPREDICTS	Counts the number of core cycles when no micro-ops are allocated and the alloc pipe is stalled waiting for a mispredicted branch to retire.	
CAH	20H	NO_ALLOC_CYCLES.RAT_STALL	Counts the number of core cycles when no micro-ops are allocated and a RATstall (caused by reservation station full) is asserted.	
CAH	90H	NO_ALLOC_CYCLES.NOT_DELIVERED	Counts the number of core cycles when no micro-ops are allocated, the IQ is empty, and no other condition is blocking allocation.	
CAH	7FH	NO_ALLOC_CYCLES.ALL	Counts the total number of core cycles when no micro-ops are allocated for any reason.	
CBH	01H	RS_FULL_STALL.MEC	Counts the number of core cycles when allocation pipeline is stalled and is waiting for a free MEC reservation station entry.	
CBH	1FH	RS_FULL_STALL.ALL	Counts the total number of core cycles the allocation pipeline is stalled when any one of the reservation stations is full.	
CDH	01H	CYCLES_DIV_BUSY.ALL	Cycles the number of core cycles when divider is busy. Does not imply a stall waiting for the divider.	
E6H	01H	BACLEARS.ALL	Counts the number of times the front end rereads for any branch as a result of another branch handling mechanism in the front end.	
E6H	08H	BACLEARS.RETURN	Counts the number of times the front end rereads for RET branches as a result of another branch handling mechanism in the front end.	
E6H	10H	BACLEARS.COND	Counts the number of times the front end rereads for conditional branches as a result of another branch handling mechanism in the front end.	
E7H	01H	MS_DECODED.MS_ENTRY	Counts the number of times the MSROM starts a flow of uops.	
PS: Also supports PEBS.				
PSDLA: Also supports PEBS and DataLA.				

## 19.5 PERFORMANCE MONITORING EVENTS FOR THE INTEL® CORE™ M AND 5TH GENERATION INTEL® CORE™ PROCESSORS

The Intel® Core™ M processors, the 5th generation Intel® Core™ processors and the Intel Xeon processor E3 1200 v4 product family are based on the Broadwell microarchitecture. They support the architectural performance monitoring events listed in Table 19-1. Model-specific performance monitoring events in the processor core are listed in Table 19-7. The events in Table 19-7 apply to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following values: 06\_3DH and 06\_47H. Table 19-10 lists performance events supporting Intel TSX (see Section 18.3.6.5) and the events are available on processors based on Broadwell microarchitecture. Fixed counters in the core PMU support the architecture events defined in Table 19-2.

Model-specific performance monitoring events that are located in the uncore sub-system are implementation specific between different platforms using processors based on Broadwell microarchitecture and with different DisplayFamily\_DisplayModel signatures. Processors with CPUID signature of DisplayFamily\_DisplayModel 06\_3DH and 06\_47H support uncore performance events listed in Table 19-11.

**Table 19-7. Performance Events of the Processor Core Supported by Broadwell Microarchitecture**

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.	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are temporarily blocked because all resources for handling the split accesses are in use.	
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	Load 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	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).	
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 add delay.	
0EH	20H	UOPS_ISSUED.SLOW_LEA	Number of slow LEA or similar uops allocated. Such uop has 3 sources (for example, 2 sources + immediate) regardless of whether it is a result of LEA instruction or not.	
0EH	40H	UOPS_ISSUED.SINGLE_MUL	Number of multiply packed/scalar single precision uops allocated.	

**Table 19-7. Performance Events of the Processor Core Supported by Broadwell Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
14H	01H	ARITH.FPU_DIV_ACTIVE	Cycles when divider is busy executing divide operations.	
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	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	E1H	L2_RQSTS.ALL_DEMAND_DATA_RD	Counts any demand and L1 HW prefetch data load requests to L2.	
24H	E2H	L2_RQSTS.ALL_RFO	Counts all L2 store RFO requests.	
24H	E4H	L2_RQSTS.ALL_CODE_RD	Counts all L2 code requests.	
24H	F8H	L2_RQSTS.ALL_PF	Counts all L2 HW prefetcher requests.	
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.
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle. Set Cmask = 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 a 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	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).	
4CH	02H	LOAD_HIT_PRE.HW_PF	Non-SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.	
4FH	10H	EPT.WALK_CYCLES	Cycles of Extended Page Table walks.	
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.	

**Table 19-7. Performance Events of the Processor Core Supported by Broadwell Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Offcore outstanding demand data read transactions in SQ to uncure. Set Cmask=1 to count cycles.	Use only when HTT is off.
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD	Offcore outstanding demand code read transactions in SQ to uncure. Set Cmask=1 to count cycles.	Use only when HTT is off.
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Offcore outstanding RFO store transactions in SQ to uncure. Set Cmask=1 to count cycles.	Use only when HTT is off.
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Offcore outstanding cacheable data read transactions in SQ to uncure. Set Cmask=1 to count cycles.	Use only when HTT is off.
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 uop. 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	Number of uops delivered to IDQ from any path.	

**Table 19-7. Performance Events of the Processor Core Supported by Broadwell Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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 cause a page walk of any page size.	
85H	02H	ITLB_MISSES.WALK_COMPLETE_D_4K	Completed page walks due to misses in ITLB 4K page entries.	
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).	
87H	01H	ILD_STALL.LCP	Stalls caused by changing prefix length of the instruction.	
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 or 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 or 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).	

**Table 19-7. Performance Events of the Processor Core Supported by Broadwell Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CO RE	Count issue pipeline slots where no uop was delivered from the front end to the back end when there is no back end stall.	Use Cmask to qualify uop b/w.
A1H	01H	UOPS_DISPATCHED_PORT.PORT _0	Counts the number of cycles in which a uop is dispatched to port 0.	Set AnyThread to count per core.
A1H	02H	UOPS_DISPATCHED_PORT.PORT _1	Counts the number of cycles in which a uop is dispatched to port 1.	Set AnyThread to count per core.
A1H	04H	UOPS_DISPATCHED_PORT.PORT _2	Counts the number of cycles in which a uop is dispatched to port 2.	Set AnyThread to count per core.
A1H	08H	UOPS_DISPATCHED_PORT.PORT _3	Counts the number of cycles in which a uop is dispatched to port 3.	Set AnyThread to count per core.
A1H	10H	UOPS_DISPATCHED_PORT.PORT _4	Counts the number of cycles in which a uop is dispatched to port 4.	Set AnyThread to count per core.
A1H	20H	UOPS_DISPATCHED_PORT.PORT _5	Counts the number of cycles in which a uop is dispatched to port 5.	Set AnyThread to count per core.
A1H	40H	UOPS_DISPATCHED_PORT.PORT _6	Counts the number of cycles in which a uop is dispatched to port 6.	Set AnyThread to count per core.
A1H	80H	UOPS_DISPATCHED_PORT.PORT _7	Counts the number of cycles in which a uop is dispatched to port 7.	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.	
A8H	01H	LSD.UOPS	Number of uops delivered by the LSD.	
ABH	02H	DSB2MITE_SWITCHES.PENALTY _CYCLES	Cycles of delay due to Decode Stream Buffer to MITE switches.	
AEH	01H	ITLB.ITLB_FLUSH	Counts the number of ITLB flushes; includes 4k/2M/4M pages.	
B0H	01H	OFFCORE_REQUESTS.DEMAND_ DATA_RD	Demand data read requests sent to uncore.	Use only when HTT is off.
B0H	02H	OFFCORE_REQUESTS.DEMAND_ CODE_RD	Demand code read requests sent to uncore.	Use only when HTT is off.
B0H	04H	OFFCORE_REQUESTS.DEMAND_ RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ltoM.	Use only when HTT is off.
B0H	08H	OFFCORE_REQUESTS.ALL_DATA _RD	Data read requests sent to uncore (demand and prefetch).	Use only when HTT is off.
B1H	01H	UOPS_EXECUTED.THREAD	Counts total number of uops to be executed per-logical-processor each cycle.	Use Cmask 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	OFF_CORE_RESPONSE_0	See Section 18.3.4.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H.



**Table 19-7. Performance Events of the Processor Core Supported by Broadwell Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.3.4.5, "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.	
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.	
COH	00H	INST_RETIRED.ANY_P	Number of instructions at retirement.	See Table 19-1.
COH	01H	INST_RETIRED.PREC_DIST	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution.	PMC1 only.
COH	02H	INST_RETIRED.X87	FP operations retired. X87 FP operations that have no exceptions.	
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 and DataLA, use Any=1 for core granular.
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	Supports PEBS.
C3H	01H	MACHINE_CLEARS.CYCLES	Counts cycles while a machine clears stalled forward progress of a logical processor or a processor core.	
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.	Supports PEBS.
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	Supports PEBS.



**Table 19-7. Performance Events of the Processor Core Supported by Broadwell Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	Supports PEBS.
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.	Supports PEBS.
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.	Supports PEBS.
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	Randomly sampled loads whose latency is above a user defined threshold. A small fraction of the overall loads are sampled due to randomization.	Specify threshold in MSR 3F6H.
D0H	11H	MEM_UOPS_RETIRED.STLB_MISSES_LOADS	Retired load uops that miss the STLB.	Supports PEBS and DataLA.
D0H	12H	MEM_UOPS_RETIRED.STLB_MISSES_STORES	Retired store uops that miss the STLB.	Supports PEBS and DataLA.
D0H	21H	MEM_UOPS_RETIRED.LOCK_LOADS	Retired load uops with locked access.	Supports PEBS and DataLA.
D0H	41H	MEM_UOPS_RETIRED.SPLIT_LOADS	Retired load uops that split across a cacheline boundary.	Supports PEBS and DataLA.
D0H	42H	MEM_UOPS_RETIRED.SPLIT_STORES	Retired store uops that split across a cacheline boundary.	Supports PEBS and DataLA.
D0H	81H	MEM_UOPS_RETIRED.ALL_LOADS	All retired load uops.	Supports PEBS and DataLA.
D0H	82H	MEM_UOPS_RETIRED.ALL_STORES	All retired store uops.	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.
D1H	04H	MEM_LOAD_UOPS_RETIRED.L3_HIT	Retired load uops with L3 cache hits as data sources.	Supports PEBS and DataLA.
D1H	08H	MEM_LOAD_UOPS_RETIRED.L1_MISS	Retired load uops missed L1 cache as data sources.	Supports PEBS and DataLA.

**Table 19-7. Performance Events of the Processor Core Supported by Broadwell Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D1H	10H	MEM_LOAD_UOPS_RETIRED.L2_MISS	Retired load uops missed L2. Unknown data source excluded.	Supports PEBS and DataLA.
D1H	20H	MEM_LOAD_UOPS_RETIRED.L3_MISS	Retired load uops missed L3. Excludes unknown data source.	Supports PEBS and DataLA.
D1H	40H	MEM_LOAD_UOPS_RETIRED.HIT_LFB	Retired load uops where data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.	Supports PEBS and DataLA.
D2H	01H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS	Retired load uops where data sources were L3 hit and cross-core snoop missed in on-pkg core cache.	Supports PEBS and DataLA.
D2H	02H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	Retired load uops where data sources were L3 and cross-core snoop hits in on-pkg core cache.	Supports PEBS and DataLA.
D2H	04H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM	Retired load uops where data sources were HitM responses from shared L3.	Supports PEBS and DataLA.
D2H	08H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_NONE	Retired load uops where data sources were hits in L3 without snoops required.	Supports PEBS and DataLA.
D3H	01H	MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCAL_DRAM	Retired load uops where data sources missed L3 but serviced from local dram.	Supports PEBS and DataLA.
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 L3 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.	

Table 19-10 lists performance events supporting Intel TSX (see Section 18.3.6.5) and the events are applicable to processors based on Broadwell microarchitecture. Where Broadwell microarchitecture implements TSX-related event semantics that differ from Table 19-10, they are listed in Table 19-8.

**Table 19-8. Intel® TSX Performance Event Addendum in Processors Based on Broadwell Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
54H	02H	TX_MEM.ABORT_CAPACITY	Number of times a transactional abort was signaled due to a data capacity limitation for transactional reads or writes.	

## 19.6 PERFORMANCE MONITORING EVENTS FOR THE 4TH GENERATION INTEL® CORE™ PROCESSORS

4th generation Intel® Core™ processors and Intel Xeon processor E3-1200 v3 product family are based on the Haswell microarchitecture. They support the architectural performance monitoring events listed in Table 19-1. Model-specific performance monitoring events in the processor core are listed in Table 19-9. The events in Table 19-9 apply to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following values: 06\_3CH, 06\_45H and 06\_46H. Table 19-10 lists performance events focused on supporting Intel TSX (see Section 18.3.6.5). Fixed counters in the core PMU support the architecture events defined in Table 19-2.

Additional information on event specifics (e.g., derivative events using specific IA32\_PERFVTSELx modifiers, limitations, special notes and recommendations) can be found at <https://software.intel.com/en-us/forums/software-tuning-performance-optimization-platform-monitoring>.

**Table 19-9. Performance Events in the Processor Core of 4th 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.	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are temporarily blocked because all resources for handling the split accesses are in use.	
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 add delay.	

**Table 19-9. Performance Events in the Processor Core of 4th 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 (for example, 2 sources + immediate) regardless of whether it is 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.
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle. Set Cmask = 1 and Edge = 1 to count occurrences.	Counter 2 only. Set Cmask = 1 to count cycles.

**Table 19-9. Performance Events in the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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_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.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Offcore outstanding demand data read transactions in SQ to uncore. Set Cmask=1 to count cycles.	Use only when HTT is off.
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD	Offcore outstanding Demand code Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	Use only when HTT is off.
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Offcore outstanding RFO store transactions in SQ to uncore. Set Cmask=1 to count cycles.	Use only when HTT is off.
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Offcore outstanding cacheable data read transactions in SQ to uncore. Set Cmask=1 to count cycles.	Use only when HTT is off.
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 19-9. Performance Events in the Processor Core of 4th 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 uop. 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_COMPLETE_D_4K	Completed page walks due to misses in ITLB 4K page entries.	
85H	04H	ITLB_MISSES.WALK_COMPLETE_D_2M_4M	Completed page walks due to misses in ITLB 2M/4M page entries.	
85H	0EH	ITLB_MISSES.WALK_COMPLETE_D	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.	
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.

**Table 19-9. Performance Events in the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
88H	04H	BR_INST_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify executed indirect near branch instructions that are not calls or 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 or 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.CO RE	Count issue pipeline slots where no uop was delivered from the front end to the back end when there is no back-end stall.	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.
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.



**Table 19-9. Performance Events in the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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	01H	CYCLE_ACTIVITY.CYCLES_L2_PENDING	Cycles with pending L2 miss loads. Set Cmask=2 to count cycle.	Use only when HTT is off.
A3H	02H	CYCLE_ACTIVITY.CYCLES_LDM_PENDING	Cycles with pending memory loads. Set Cmask=2 to count cycle.	
A3H	05H	CYCLE_ACTIVITY.STALLS_L2_PENDING	Number of loads missed L2.	Use only when HTT is off.
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_PENDING	Cycles with pending L1 data cache miss loads. Set Cmask=8 to count cycle.	PMC2 only.
A3H	0CH	CYCLE_ACTIVITY.STALLS_L1D_PENDING	Execution stalls due to L1 data cache miss loads. Set Cmask=0CH.	PMC2 only.
A8H	01H	LSD.UOPS	Number of uops delivered by the LSD.	
AEH	01H	ITLB.ITLB_FLUSH	Counts the number of ITLB flushes, includes 4k/2M/4M pages.	
B0H	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data read requests sent to uncore.	Use only when HTT is off.
B0H	02H	OFFCORE_REQUESTS.DEMAND_CODE_RD	Demand code read requests sent to uncore.	Use only when HTT is off.
B0H	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ItoM.	Use only when HTT is off.
B0H	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	Use only when HTT is off.
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 Table 18-28 or Table 18-29.	Requires MSR 01A6H.
BBH	01H	OFF_CORE_RESPONSE_1	See Table 18-28 or Table 18-29.	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.	
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.	



**Table 19-9. Performance Events in the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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.PREC_DIST	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution.	PMC1 only.
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 and DataLA; use Any=1 for core granular.
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	Supports PEBS.
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.	Supports PEBS.
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	Supports PEBS.
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	Supports PEBS.
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.	Supports PEBS.
C4H	40H	BR_INST_RETIRED.FAR_BRANCHES	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.	Supports PEBS.
C5H	20H	BR_MISP_RETIRED.NEAR_TAKEN	Number of near branch instructions retired that were taken but mispredicted.	

Table 19-9. Performance Events in the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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	Randomly sampled loads whose latency is above a user defined threshold. A small fraction of the overall loads are sampled due to randomization.	Specify threshold in MSR 3F6H.
DOH	11H	MEM_UOPS_RETIRED.STLB_MISSES_LOADS	Retired load uops that miss the STLB.	Supports PEBS and DataLA.
DOH	12H	MEM_UOPS_RETIRED.STLB_MISSES_STORES	Retired store uops that miss the STLB.	Supports PEBS and DataLA.
DOH	21H	MEM_UOPS_RETIRED.LOCK_LOADS	Retired load uops with locked access.	Supports PEBS and DataLA.
DOH	41H	MEM_UOPS_RETIRED.SPLIT_LOADS	Retired load uops that split across a cacheline boundary.	Supports PEBS and DataLA.
DOH	42H	MEM_UOPS_RETIRED.SPLIT_STORES	Retired store uops that split across a cacheline boundary.	Supports PEBS and DataLA.
DOH	81H	MEM_UOPS_RETIRED.ALL_LOADS	All retired load uops.	Supports PEBS and DataLA.
DOH	82H	MEM_UOPS_RETIRED.ALL_STORES	All retired store uops.	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.
D1H	04H	MEM_LOAD_UOPS_RETIRED.L3_HIT	Retired load uops with L3 cache hits as data sources.	Supports PEBS and DataLA.
D1H	08H	MEM_LOAD_UOPS_RETIRED.L1_MISS	Retired load uops missed L1 cache 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	20H	MEM_LOAD_UOPS_RETIRED.L3_MISS	Retired load uops missed L3. Excludes unknown data source .	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.	Supports PEBS and DataLA.
D2H	01H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS	Retired load uops which data sources were L3 hit and cross-core snoop missed in on-pkg core cache.	Supports PEBS and DataLA.
D2H	02H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	Retired load uops which data sources were L3 and cross-core snoop hits in on-pkg core cache.	Supports PEBS and DataLA.
D2H	04H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM	Retired load uops which data sources were HitM responses from shared L3.	Supports PEBS and DataLA.

**Table 19-9. Performance Events in the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D2H	08H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_NONE	Retired load uops which data sources were hits in L3 without snoops required.	Supports PEBS and DataLA.
D3H	01H	MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCAL_DRAM	Retired load uops which data sources missed L3 but serviced from local dram.	Supports PEBS and DataLA.
E6H	1FH	BACLEAR.S.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 L3 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 19-10. Intel TSX Performance Events in Processors Based on Haswell Microarchitecture**

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.	
54H	02H	TX_MEM.ABORT_CAPACITY_WRITE	Number of times a transactional abort was signaled due to a data capacity limitation for transactional writes.	
54H	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.	
54H	08H	TX_MEM.ABORT_HLE_ELISION_BUFFER_NOT_EMPTY	Number of times an HLE transactional execution aborted due to NoAllocatedElisionBuffer being non-zero.	
54H	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.	
54H	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.	

Table 19-10. Intel TSX Performance Events in Processors Based on Haswell Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
54H	40H	TX_MEM.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.	
5DH	02H	TX_EXEC.MISC2	Counts the number of times a class of instructions (for example, vzeroupper) that may cause a transactional abort was executed inside a transactional region.	
5DH	04H	TX_EXEC.MISC3	Counts the number of times an instruction execution caused the transactional nest count supported to be exceeded.	
5DH	08H	TX_EXEC.MISC4	Counts the number of times an XBEGIN instruction was executed inside an HLE transactional region.	
5DH	10H	TX_EXEC.MISC5	Counts the number of times an instruction with HLE-XACQUIRE semantic was executed inside an RTM transactional region.	
C8H	01H	HLE_RETIREDD.START	Number of times an HLE execution started.	IF HLE is supported.
C8H	02H	HLE_RETIREDD.COMMIT	Number of times an HLE execution successfully committed.	
C8H	04H	HLE_RETIREDD.ABORTED	Number of times an HLE execution aborted due to any reasons (multiple categories may count as one). Supports PEBS.	
C8H	08H	HLE_RETIREDD.ABORTED_MEM	Number of times an HLE execution aborted due to various memory events (for example, read/write capacity and conflicts).	
C8H	10H	HLE_RETIREDD.ABORTED_TIMER	Number of times an HLE execution aborted due to uncommon conditions.	
C8H	20H	HLE_RETIREDD.ABORTED_UNFRIENDLY	Number of times an HLE execution aborted due to HLE-unfriendly instructions.	
C8H	40H	HLE_RETIREDD.ABORTED_MEMORY_TYPE	Number of times an HLE execution aborted due to incompatible memory type.	
C8H	80H	HLE_RETIREDD.ABORTED_EVENTS	Number of times an HLE execution aborted due to none of the previous 4 categories (for example, interrupts).	
C9H	01H	RTM_RETIREDD.START	Number of times an RTM execution started.	IF RTM is supported.
C9H	02H	RTM_RETIREDD.COMMIT	Number of times an RTM execution successfully committed.	
C9H	04H	RTM_RETIREDD.ABORTED	Number of times an RTM execution aborted due to any reasons (multiple categories may count as one). Supports PEBS.	

**Table 19-10. Intel TSX Performance Events in Processors Based on Haswell Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C9H	08H	RTM_RETIRED.ABORTED_MEM	Number of times an RTM execution aborted due to various memory events (for example, read/write capacity and conflicts).	IF RTM is supported.
C9H	10H	RTM_RETIRED.ABORTED_TIME R	Number of times an RTM execution aborted due to uncommon conditions.	
C9H	20H	RTM_RETIRED.ABORTED_UNFRIENDLY	Number of times an RTM execution aborted due to HLE-unfriendly instructions.	
C9H	40H	RTM_RETIRED.ABORTED_MEMTYPE	Number of times an RTM execution aborted due to incompatible memory type.	
C9H	80H	RTM_RETIRED.ABORTED_EVENTS	Number of times an RTM execution aborted due to none of the previous 4 categories (for example, interrupt).	

Model-specific performance monitoring events that are located in the uncore sub-system are implementation specific between different platforms using processors based on Haswell microarchitecture and with different DisplayFamily\_DisplayModel signatures. Processors with CPUID signature of DisplayFamily\_DisplayModel 06\_3CH and 06\_45H support performance events listed in Table 19-11.

**Table 19-11. Uncore Performance Events in the 4th Generation Intel® Core™ Processors**

Event Num. <sup>1</sup>	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.INVALID	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.INVALID_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 L3 eviction.	
34H	01H	UNC_CBO_CACHE_LOOKUP.M	L3 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	06H	UNC_CBO_CACHE_LOOKUP.E	L3 lookup request that access cache and found line in E or S state.	
34H	08H	UNC_CBO_CACHE_LOOKUP.I	L3 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.	

**Table 19-11. Uncore Performance Events in the 4th Generation Intel® Core™ Processors (Contd.)**

Event Num. <sup>1</sup>	Umask Value	Event Mask Mnemonic	Description	Comment
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 L3.	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 L3.	
81H	20H	UNC_ARB_TRK_REQUEST.WRITES	Counts the number of allocated write entries, include full, partial, and L3 evictions.	
81H	80H	UNC_ARB_TRK_REQUEST.EVICTIONS	Counts the number of L3 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.

### 19.6.1 Performance Monitoring Events in the Processor Core of Intel Xeon Processor E5 v3 Family

Model-specific performance monitoring events in the processor core that are applicable only to Intel Xeon processor E5 v3 family based on the Haswell-E microarchitecture, with CPUID signature of DisplayFamily\_DisplayModel 06\_3FH, are listed in Table 19-12. The performance events listed in Table 19-9 and Table 19-10 also apply Intel Xeon processor E5 v3 family, except that the OFF\_CORE\_RESPONSE\_x event listed in Table 19-9 should reference Table 18-30.

Uncore performance monitoring events for Intel Xeon Processor E5 v3 families are described in “Intel® Xeon® Processor E5 v3 Uncore Performance Monitoring Programming Reference Manual”.

**Table 19-12. Performance Events Applicable only to the Processor Core of Intel® Xeon® Processor E5 v3 Family**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D3H	04H	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_DRAM	Retired load uops whose data sources were remote DRAM (snoop not needed, Snoop Miss).	Supports PEBS.
D3H	10H	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_HITM	Retired load uops whose data sources were remote cache HITM.	Supports PEBS.
D3H	20H	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_FWD	Retired load uops whose data sources were forwards from a remote cache.	Supports PEBS.

## 19.7 PERFORMANCE MONITORING EVENTS FOR 3RD GENERATION INTEL® CORE™ PROCESSORS

3rd generation Intel® Core™ processors and Intel Xeon processor E3-1200 v2 product family are based on Intel microarchitecture code name Ivy Bridge. They support architectural performance monitoring events listed in Table 19-1. Model-specific performance monitoring events in the processor core are listed in Table 19-13. The events in Table 19-13 apply to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following values: 06\_3AH. Fixed counters in the core PMU support the architecture events defined in Table 19-24.

Additional information on event specifics (e.g. derivative events using specific IA32\_PERFVTSELx modifiers, limitations, special notes and recommendations) can be found at <https://software.intel.com/en-us/forums/software-tuning-performance-optimization-platform-monitoring>.

**Table 19-13. 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.	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are temporarily blocked because all resources for handling the split accesses are in use.	
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_CAUSE_S_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.	
08H	88H	DTLB_LOAD_MISSES.LARGE_PAGE_WALK_DURATION	Page walk for a large page completed for Demand load.	
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.	
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.	
10H	01H	FP_COMP_OPS_EXE.X87	Counts number of X87 uops executed.	
10H	10H	FP_COMP_OPS_EXE.SSE_FP_PACKED_DOUBLE	Counts number of SSE* or AVX-128 double precision FP packed uops executed.	
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE	Counts number of SSE* or AVX-128 single precision FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_PACKED_SINGLE	Counts number of SSE* or AVX-128 single precision FP packed uops executed.	



**Table 19-13. 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
10H	80H	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE	Counts number of SSE* or AVX-128 double precision FP scalar uops executed.	
11H	01H	SIMD_FP_256.PACKED_SINGLE	Counts 256-bit packed single-precision floating-point instructions.	
11H	02H	SIMD_FP_256.PACKED_DOUBLE	Counts 256-bit packed double-precision floating-point instructions.	
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
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.



**Table 19-13. 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
3CH	01H	CPU_CLK_THREAD_UNHALTED.R EF_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 Cmask = 1 and Edge =1 to count occurrences.	PMC2 only; Set Cmask = 1 to count cycles.
49H	01H	DTLB_STORE_MISSES.MISS_CAUS ES_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_CO MPLETED	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_DUR ATION	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_EL IMINATED	Number of integer Move Elimination candidate uops that were not eliminated.	
58H	08H	MOVE_ELIMINATION.SIMD_NOT_E LIMINATED	Number of SIMD Move Elimination candidate uops that were not eliminated.	
58H	01H	MOVE_ELIMINATION.INT_ELIMINA TED	Number of integer Move Elimination candidate uops that were eliminated.	
58H	02H	MOVE_ELIMINATION.SIMD_ELIMIN ATED	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	04H	DTLB_LOAD_MISSES.STLB_HIT	Counts load operations that missed 1st level DTLB but hit the 2nd level.	
60H	01H	OFFCORE_REQUESTS_OUTSTAN DING.DEMAND_DATA_RD	Offcore outstanding Demand Data Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	02H	OFFCORE_REQUESTS_OUTSTAN DING.DEMAND_CODE_RD	Offcore outstanding Demand Code Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	04H	OFFCORE_REQUESTS_OUTSTAN DING.DEMAND_RFO	Offcore outstanding RFO store transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	08H	OFFCORE_REQUESTS_OUTSTAN DING.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_L OCK_DURATION	Cycles in which the L1D and L2 are locked, due to a UC lock or split lock.	

**Table 19-13. 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
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	04H	ICACHE.IFETCH_STALL	Cycles where a code-fetch stalled due to L1 instruction-cache miss or an iTLB miss.	
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 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 or 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.

**Table 19-13. 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
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 or 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 issue pipeline slots where no uop was delivered from the front end to the back end when there is no back-end stall.	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	0CH	UOPS_DISPATCHED_PORT.PORT_2	Cycles which a Uop is dispatched on port 2.	
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.	

**Table 19-13. 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
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.	Restricted to counters 0-3 when HTT is disabled.
A3H	04H	CYCLE_ACTIVITY.CYCLES_NO_EXECUTE	Cycles of dispatch stalls. Set AnyThread to count per core.	Restricted to counters 0-3 when HTT is disabled.
A3H	05H	CYCLE_ACTIVITY.STALLS_L2_PENDING	Number of loads missed L2.	Restricted to counters 0-3 when HTT is disabled.
A3H	06H	CYCLE_ACTIVITY.STALLS_LDM_PENDING		Restricted to counters 0-3 when HTT is disabled.
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_PENDING	Cycles with pending L1 cache miss loads. Set AnyThread to count per core.	PMC2 only.
A3H	0CH	CYCLE_ACTIVITY.STALLS_L1D_PENDING	Execution stalls due to L1 data cache miss loads. Set Cmask=0CH.	PMC2 only.
A8H	01H	LSD.UOPS	Number of Uops delivered by the LSD.	
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.	
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 18.3.4.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H.
BBH	01H	OFFCORE_RESPONSE_1	See Section 18.3.4.5, "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.PREC_DIST	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.	

**Table 19-13. 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
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.	
C1H	80H	OTHER_ASSISTS.WB	Number of times microcode assist is invoked by hardware 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.	Supports PEBS.
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.	Supports PEBS.
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	Supports PEBS.
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	Supports PEBS.
C4H	10H	BR_INST_RETIRED.NOT_TAKEN	Counts the number of not taken branch instructions retired.	Supports PEBS.
C4H	20H	BR_INST_RETIRED.NEAR_TAKEN	Number of near taken branches retired.	Supports PEBS.
C4H	40H	BR_INST_RETIRED.FAR_BRANCH	Number of far branches retired.	Supports PEBS.
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.	Supports PEBS.
C5H	20H	BR_MISP_RETIRED.NEAR_TAKEN	Mispredicted taken branch instructions retired.	Supports PEBS.
CAH	02H	FP_ASSIST.X87_OUTPUT	Number of X87 FP assists due to output values.	Supports PEBS.
CAH	04H	FP_ASSIST.X87_INPUT	Number of X87 FP assists due to input values.	Supports PEBS.
CAH	08H	FP_ASSIST.SIMD_OUTPUT	Number of SIMD FP assists due to output values.	Supports PEBS.
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.	

**Table 19-13. 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
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY	Randomly sampled loads whose latency is above a user defined threshold. A small fraction of the overall loads are sampled due to randomization.	Specify threshold in MSR 3F6H. PMC 3 only.
CDH	02H	MEM_TRANS_RETIRED.PRECISE_STORE	Sample stores and collect precise store operation via PEBS record. PMC3 only.	See Section 18.3.4.4.3.
DOH	11H	MEM_UOPS_RETIRED.STLB_MISS_LOADS	Retired load uops that miss the STLB.	Supports PEBS.
DOH	12H	MEM_UOPS_RETIRED.STLB_MISS_STORES	Retired store uops that miss the STLB.	Supports PEBS.
DOH	21H	MEM_UOPS_RETIRED.LOCK_LOADS	Retired load uops with locked access.	Supports PEBS.
DOH	41H	MEM_UOPS_RETIRED.SPLIT_LOADS	Retired load uops that split across a cacheline boundary.	Supports PEBS.
DOH	42H	MEM_UOPS_RETIRED.SPLIT_STORES	Retired store uops that split across a cacheline boundary.	Supports PEBS.
DOH	81H	MEM_UOPS_RETIRED.ALL_LOADS	All retired load uops.	Supports PEBS.
DOH	82H	MEM_UOPS_RETIRED.ALL_STORES	All retired store uops.	Supports PEBS.
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS.
D1H	02H	MEM_LOAD_UOPS_RETIRED.L2_HIT	Retired load uops with L2 cache hits as data sources.	Supports PEBS.
D1H	04H	MEM_LOAD_UOPS_RETIRED.LLC_HIT	Retired load uops whose data source was LLC hit with no snoop required.	Supports PEBS.
D1H	08H	MEM_LOAD_UOPS_RETIRED.L1_MISS	Retired load uops whose data source followed an L1 miss.	Supports PEBS.
D1H	10H	MEM_LOAD_UOPS_RETIRED.L2_MISS	Retired load uops that missed L2, excluding unknown sources.	Supports PEBS.
D1H	20H	MEM_LOAD_UOPS_RETIRED.LLC_MISS	Retired load uops whose data source is LLC miss.	Supports PEBS. Restricted to counters 0-3 when HTT is disabled.
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.	Supports PEBS.
D2H	01H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS	Retired load uops whose data source was an on-package core cache LLC hit and cross-core snoop missed.	Supports PEBS.
D2H	02H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT	Retired load uops whose data source was an on-package LLC hit and cross-core snoop hits.	Supports PEBS.
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Retired load uops whose data source was an on-package core cache with HitM responses.	Supports PEBS.
D2H	08H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE	Retired load uops whose data source was LLC hit with no snoop required.	Supports PEBS.
D3H	01H	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM	Retired load uops whose data source was local memory (cross-socket snoop not needed or missed).	Supports PEBS.
E6H	1FH	BACLEARS.ANY	Number of front end re-steers due to BPU misprediction.	

**Table 19-13. 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	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	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.

### 19.7.1 Performance Monitoring Events in the Processor Core of Intel Xeon Processor E5 v2 Family and Intel Xeon Processor E7 v2 Family

Model-specific performance monitoring events in the processor core that are applicable only to Intel Xeon processor E5 v2 family and Intel Xeon processor E7 v2 family based on the Ivy Bridge-E microarchitecture, with CPUID signature of DisplayFamily\_DisplayModel 06\_3EH, are listed in Table 19-14.

**Table 19-14. Performance Events Applicable Only to the Processor Core of Intel® Xeon® Processor E5 v2 Family and Intel® Xeon® Processor E7 v2 Family**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D3H	03H	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM	Retired load uops whose data sources were local DRAM (snoop not needed, Snoop Miss, or Snoop Hit data not forwarded).	Supports PEBS.
D3H	0CH	MEM_LOAD_UOPS_LLC_MISS_RETIRED.REMOTE_DRAM	Retired load uops whose data source was remote DRAM (snoop not needed, Snoop Miss, or Snoop Hit data not forwarded).	Supports PEBS.
D3H	10H	MEM_LOAD_UOPS_LLC_MISS_RETIRED.REMOTE_HITM	Retired load uops whose data sources were remote HITM.	Supports PEBS.
D3H	20H	MEM_LOAD_UOPS_LLC_MISS_RETIRED.REMOTE_FWD	Retired load uops whose data sources were forwards from a remote cache.	Supports PEBS.



## 19.8 PERFORMANCE MONITORING EVENTS FOR 2ND GENERATION INTEL® CORE™ I7-2XXX, INTEL® CORE™ I5-2XXX, INTEL® CORE™ I3-2XXX PROCESSOR SERIES

2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel Xeon processor E3-1200 product family are based on the Intel microarchitecture code name Sandy Bridge. They support architectural performance monitoring events listed in Table 19-1. Model-specific performance monitoring events in the processor core are listed in Table 19-15, Table 19-16, and Table 19-17. The events in Table 19-15 apply to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following values: 06\_2AH and 06\_2DH. The events in Table 19-16 apply to processors with CPUID signature 06\_2AH. The events in Table 19-17 apply to processors with CPUID signature 06\_2DH. Fixed counters in the core PMU support the architecture events defined in Table 19-2.

Additional information on event specifics (e.g. derivative events using specific IA32\_PERFVTSELx modifiers, limitations, special notes and recommendations) can be found at <https://software.intel.com/en-us/forums/software-tuning-performance-optimization-platform-monitoring>.

**Table 19-15. 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 E3 and E5 Family**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	01H	LD_BLOCKS.DATA_UNKNOWN	Blocked loads due to store buffer blocks with unknown data.	
03H	02H	LD_BLOCKS.STORE_FORWARD	Loads blocked by overlapping with store buffer that cannot be forwarded.	
03H	08H	LD_BLOCKS.NO_SR	# of Split loads blocked due to resource not available.	
03H	10H	LD_BLOCKS.ALL_BLOCK	Number of cases where any load is blocked but has no DCU miss.	
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.	
07H	08H	LD_BLOCKS_PARTIAL.ALL_STORE_BLOCK	The number of times that load operations are temporarily blocked because of older stores, with addresses that are not yet known. A load operation may incur more than one block of this type.	
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	Misses in all TLB levels that caused page walk completed of any size.	
08H	04H	DTLB_LOAD_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
08H	10H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.	
0DH	03H	INT_MISC.RECOVERY_CYCLES	Cycles waiting to recover after Machine Clears or JEClear. Set Cmask= 1.	Set Edge to count occurrences.
0DH	40H	INT_MISC.RAT_STALL_CYCLES	Cycles RAT external stall is sent to IDQ for this thread.	



**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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.
10H	01H	FP_COMP_OPS_EXE.X87	Counts number of X87 uops executed.	
10H	10H	FP_COMP_OPS_EXE.SSE_FP_PACKED_DOUBLE	Counts number of SSE* double precision FP packed uops executed.	
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE	Counts number of SSE* single precision FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_PACKED_SINGLE	Counts number of SSE* single precision FP packed uops executed.	
10H	80H	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE	Counts number of SSE* double precision FP scalar uops executed.	
11H	01H	SIMD_FP_256.PACKED_SINGLE	Counts 256-bit packed single-precision floating-point instructions.	
11H	02H	SIMD_FP_256.PACKED_DOUBLE	Counts 256-bit packed double-precision floating-point instructions.	
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.	
17H	01H	INSTS_WRITTEN_TO_IQ.INSTS	Counts the number of instructions written into the IQ every cycle.	
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	Requests from L2 Hardware prefetcher that hit L2.	
24H	80H	L2_RQSTS.PF_MISS	Requests from L2 Hardware prefetcher that missed L2.	
24H	C0H	L2_RQSTS.ALL_PF	Any requests from L2 Hardware prefetchers.	
27H	01H	L2_STORE_LOCK_RQSTS.MISS	RFOs that miss cache lines.	
27H	04H	L2_STORE_LOCK_RQSTS.HIT_E	RFOs that hit cache lines in E state.	
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.	

**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
28H	01H	L2_L1D_WB_RQSTS.MISS	Not rejected writebacks from L1D to L2 cache lines that missed L2.	
28H	02H	L2_L1D_WB_RQSTS.HIT_S	Not rejected writebacks from L1D to L2 cache lines in S state.	
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.	
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.
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle. Set Cmask = 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	Not SW-prefetch load dispatches that hit fill buffer allocated for S/W prefetch.	
4CH	02H	LOAD_HIT_PRE.HW_PF	Not SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.	
4EH	02H	HW_PRE_REQ.DL1_MISS	Hardware Prefetch requests that miss the L1D cache. A request is being counted each time it access the cache & miss it, including if a block is applicable or if hit the Fill Buffer for example.	This accounts for both L1 streamer and IP-based (IPP) HW prefetchers.
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
51H	02H	L1D.ALLOCATED_IN_M	Counts the number of allocations of modified L1D cache lines.	
51H	04H	L1D.EVICTION	Counts the number of modified lines evicted from the L1 data cache due to replacement.	

**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
51H	08H	L1D.ALL_M_REPLACEMENT	Cache lines in M state evicted out of L1D due to Snoop HitM or dirty line replacement.	
59H	20H	PARTIAL_RAT_STALLS.FLAGS_MERGE_UOP	Increments the number of flags-merge uops in flight each cycle. Set Cmask = 1 to count cycles.	
59H	40H	PARTIAL_RAT_STALLS.SLOW_LEA_WINDOW	Cycles with at least one slow LEA uop allocated.	
59H	80H	PARTIAL_RAT_STALLS.MUL_SINGLE_UOP	Number of Multiply packed/scalar single precision uops allocated.	
5BH	0CH	RESOURCE_STALLS2.ALL_FL_EMPTY	Cycles stalled due to free list empty.	PMCO-3 only regardless HTT.
5BH	0FH	RESOURCE_STALLS2.ALL_PRF_CONTROL	Cycles stalled due to control structures full for physical registers.	
5BH	40H	RESOURCE_STALLS2.BOB_FULL	Cycles Allocator is stalled due Branch Order Buffer.	
5BH	4FH	RESOURCE_STALLS2.OOO_RESOURCE	Cycles stalled due to out of order resources full.	
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.	
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	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 MS is busy. Set Cmask=1 and Edge = 1 to count MS activations.	Can combine Umask 08H and 10H.
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ when MS is busy by MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H and 20H.

**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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 and 30H.
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 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	41H	BR_INST_EXEC.NONTAKEN_CONDITIONAL	Not-taken macro conditional branches.	
88H	81H	BR_INST_EXEC.TAKEN_CONDITIONAL	Taken speculative and retired conditional branches.	
88H	82H	BR_INST_EXEC.TAKEN_DIRECT_JUMP	Taken speculative and retired conditional branches excluding calls and indirects.	
88H	84H	BR_INST_EXEC.TAKEN_INDIRECT_JUMP_NON_CALL_RET	Taken speculative and retired indirect branches excluding calls and returns.	
88H	88H	BR_INST_EXEC.TAKEN_INDIRECT_NEAR_RETURN	Taken speculative and retired indirect branches that are returns.	
88H	90H	BR_INST_EXEC.TAKEN_DIRECT_NEAR_CALL	Taken speculative and retired direct near calls.	
88H	A0H	BR_INST_EXEC.TAKEN_INDIRECT_NEAR_CALL	Taken speculative and retired indirect near calls.	
88H	C1H	BR_INST_EXEC.ALL_CONDITIONAL	Speculative and retired conditional branches.	
88H	C2H	BR_INST_EXEC.ALL_DIRECT_JUMP	Speculative and retired conditional branches excluding calls and indirects.	
88H	C4H	BR_INST_EXEC.ALL_INDIRECT_JUMP_NON_CALL_RET	Speculative and retired indirect branches excluding calls and returns.	
88H	C8H	BR_INST_EXEC.ALL_INDIRECT_NEAR_RETURN	Speculative and retired indirect branches that are returns.	
88H	D0H	BR_INST_EXEC.ALL_NEAR_CALL	Speculative and retired direct near calls.	
88H	FFH	BR_INST_EXEC.ALL_BRANCHES	Speculative and retired branches.	
89H	41H	BR_MISP_EXEC.NONTAKEN_CONDITIONAL	Not-taken mispredicted macro conditional branches.	
89H	81H	BR_MISP_EXEC.TAKEN_CONDITIONAL	Taken speculative and retired mispredicted conditional branches.	

**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
89H	84H	BR_MISP_EXEC.TAKEN_INDIRECT_JUMP_NON_CALL_RET	Taken speculative and retired mispredicted indirect branches excluding calls and returns.	
89H	88H	BR_MISP_EXEC.TAKEN_RETURN_NEAR	Taken speculative and retired mispredicted indirect branches that are returns.	
89H	90H	BR_MISP_EXEC.TAKEN_DIRECT_NEAR_CALL	Taken speculative and retired mispredicted direct near calls.	
89H	A0H	BR_MISP_EXEC.TAKEN_INDIRECT_NEAR_CALL	Taken speculative and retired mispredicted indirect near calls.	
89H	C1H	BR_MISP_EXEC.ALL_CONDITIONAL	Speculative and retired mispredicted conditional branches.	
89H	C4H	BR_MISP_EXEC.ALL_INDIRECT_JUMP_NON_CALL_RET	Speculative and retired mispredicted indirect branches excluding calls and returns.	
89H	D0H	BR_MISP_EXEC.ALL_NEAR_CALL	Speculative and retired mispredicted direct near calls.	
89H	FFH	BR_MISP_EXEC.ALL_BRANCHES	Speculative and retired mispredicted branches.	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CORE	Count issue pipeline slots where no uop was delivered from the front end to the back end when there is no back-end stall.	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	0CH	UOPS_DISPATCHED_PORT.PORT_2	Cycles which a Uop is dispatched on port 2.	
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	02H	RESOURCE_STALLS.LB	Counts the cycles of stall due to lack of load buffers.	
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 from sync).	
A2H	10H	RESOURCE_STALLS.ROB	Cycles stalled due to re-order buffer full.	
A2H	20H	RESOURCE_STALLS.FCSW	Cycles stalled due to writing the FPU control word.	
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_PENDING	Cycles with pending L2 miss loads. Set AnyThread to count per core.	
A3H	02H	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_DISPATCH	Cycles of dispatch stalls. Set AnyThread to count per core.	PMCO-3 only.

**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A3H	05H	CYCLE_ACTIVITY.STALL_CYCLE_S_L2_PENDING		PMCO-3 only.
A3H	06H	CYCLE_ACTIVITY.STALL_CYCLE_S_L1D_PENDING		PMC2 only.
A8H	01H	LSD.UOPS	Number of Uops delivered by the LSD.	
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	02H	DSB_FILL.OTHER_CANCEL	Cases of cancelling valid DSB fill not because of exceeding way limit.	
ACH	08H	DSB_FILL.EXCEED_DSB_LINES	DSB Fill encountered > 3 DSB lines.	
AEH	01H	ITLB.ITLB_FLUSH	Counts the number of ITLB flushes; includes 4k/2M/4M pages.	
B0H	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data read requests sent to uncore.	
B0H	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ItoM.	
B0H	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	
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.
B1H	02H	UOPS_DISPATCHED.CORE	Counts total number of uops to be dispatched per-core each cycle.	Do not need to set ANY.
B2H	01H	OFFCORE_REQUESTS_BUFFER_SQ_FULL	Offcore requests buffer cannot take more entries for this thread core.	
B6H	01H	AGU_BYPASS_CANCEL.COUNT	Counts executed load operations with all the following traits: 1. Addressing of the format [base + offset], 2. The offset is between 1 and 2047, 3. The address specified in the base register is in one page and the address [base+offset] is in another page.	
B7H	01H	OFF_CORE_RESPONSE_0	See Section 18.3.4.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H.
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.3.4.5, "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.	
BFH	05H	L1D_BLOCKS.BANK_CONFLICT_CYCLES	Cycles when dispatched loads are cancelled due to L1D bank conflicts with other load ports.	Cmask=1.
COH	00H	INST_RETIRED.ANY_P	Number of instructions at retirement.	See Table 19-1.
COH	01H	INST_RETIRED.PREC_DIST	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution.	PMC1 only; must quiesce other PMCs.
C1H	02H	OTHER_ASSISTS.ITLB_MISS_RETIRED	Instructions that experienced an ITLB miss.	

**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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.
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	Supports PEBS.
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEARS.SMC	Counts the number of times that a program writes to a code section.	
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.	Supports PEBS.
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	Supports PEBS.
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	Supports PEBS.
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.	Supports PEBS.
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.	Supports PEBS.
C5H	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	Supports PEBS.
C5H	10H	BR_MISP_RETIRED.NOT_TAKEN	Mispredicted not taken branch instructions retired.	Supports PEBS.
C5H	20H	BR_MISP_RETIRED.TAKEN	Mispredicted taken branch instructions retired.	Supports PEBS.
CAH	02H	FP_ASSIST.X87_OUTPUT	Number of X87 assists due to output value.	



**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
CAH	04H	FP_ASSIST.X87_INPUT	Number of X87 assists due to input value.	
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_INSE RTS	Count cases of saving new LBR records by hardware.	
CDH	01H	MEM_TRANS_RETIRED.LOAD_ LATENCY	Randomly sampled loads whose latency is above a user defined threshold. A small fraction of the overall loads are sampled due to randomization. PMC3 only.	Specify threshold in MSR 3F6H.
CDH	02H	MEM_TRANS_RETIRED.PRECIS E_STORE	Sample stores and collect precise store operation via PEBS record. PMC3 only.	See Section 18.3.4.4.3.
D0H	11H	MEM_UOPS_RETIRED.STLB_MI SS_LOADS	Retired load uops that miss the STLB.	Supports PEBS. PMCO-3 only regardless HTT.
D0H	12H	MEM_UOPS_RETIRED.STLB_MI SS_STORES	Retired store uops that miss the STLB.	Supports PEBS. PMCO-3 only regardless HTT.
D0H	21H	MEM_UOPS_RETIRED.LOCK_LO ADS	Retired load uops with locked access.	Supports PEBS. PMCO-3 only regardless HTT.
D0H	41H	MEM_UOPS_RETIRED.SPLIT_L OADS	Retired load uops that split across a cacheline boundary.	Supports PEBS. PMCO-3 only regardless HTT.
D0H	42H	MEM_UOPS_RETIRED.SPLIT_S TORES	Retired store uops that split across a cacheline boundary.	Supports PEBS. PMCO-3 only regardless HTT.
D0H	81H	MEM_UOPS_RETIRED.ALL_LOA DS	All retired load uops.	Supports PEBS. PMCO-3 only regardless HTT.
D0H	82H	MEM_UOPS_RETIRED.ALL_STO RES	All retired store uops.	Supports PEBS. PMCO-3 only regardless HTT.
D1H	01H	MEM_LOAD_UOPS_RETIRED.L 1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS. PMCO-3 only regardless HTT.
D1H	02H	MEM_LOAD_UOPS_RETIRED.L 2_HIT	Retired load uops with L2 cache hits as data sources.	Supports PEBS.
D1H	04H	MEM_LOAD_UOPS_RETIRED.LL C_HIT	Retired load uops which data sources were data hits in LLC without snoops required.	Supports PEBS.
D1H	20H	MEM_LOAD_UOPS_RETIRED.LL C_MISS	Retired load uops which data sources were data missed LLC (excluding unknown data source).	Supports PEBS.
D1H	40H	MEM_LOAD_UOPS_RETIRED.HI T_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.	Supports PEBS.
D2H	01H	MEM_LOAD_UOPS_LLC_HIT_R ETIRED.XSNP_MISS	Retired load uops whose data source was an on-package core cache LLC hit and cross-core snoop missed.	Supports PEBS.
D2H	02H	MEM_LOAD_UOPS_LLC_HIT_R ETIRED.XSNP_HIT	Retired load uops whose data source was an on-package LLC hit and cross-core snoop hits.	Supports PEBS.
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_R ETIRED.XSNP_HITM	Retired load uops whose data source was an on-package core cache with HitM responses.	Supports PEBS.
D2H	08H	MEM_LOAD_UOPS_LLC_HIT_R ETIRED.XSNP_NONE	Retired load uops whose data source was LLC hit with no snoop required.	Supports PEBS.



**Table 19-15. 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 E3 and E5 Family (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
E6H	01H	BACLEARS.ANY	Counts the number of times the front end is re-steered, mainly when the BPU cannot provide a correct prediction and this is corrected by other branch handling mechanisms at the front end.	
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	L2 or LLC HW prefetches that access L2 cache.	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	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 L2 prefetch.	
F2H	08H	L2_LINES_OUT.PF_DIRTY	Dirty L2 cache lines evicted by L2 prefetch.	
F2H	0AH	L2_LINES_OUT.DIRTY_ALL	Dirty L2 cache lines filling the L2.	Counting does not cover rejects.
F4H	10H	SQ_MISC.SPLIT_LOCK	Split locks in SQ.	

Non-architecture performance monitoring events in the processor core that are applicable only to Intel processors with CPUID signature of DisplayFamily\_DisplayModel 06\_2AH are listed in Table 19-16.

**Table 19-16. 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.	Supports PEBS.
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Retired load uops which data sources were HitM responses from shared LLC.	Supports PEBS.
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.

**Table 19-16. 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 (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D4H	02H	MEM_LOAD_UOPS_MISC_RETI RED.LLC_MISS	Retired load uops with unknown information as data source in cache serviced the load.	Supports PEBS. PMCO-3 only regardless HTT.
B7H/BBH	01H	OFFCORE_RESPONSE_N	Sub-events of OFFCORE_RESPONSE_N (suffix N = 0, 1) programmed using MSR 01A6H/01A7H with values shown in the comment column.	
		OFFCORE_RESPONSE.ALL_CODE_RD.LLC_HIT_N		10003C0244H
		OFFCORE_RESPONSE.ALL_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0244H
		OFFCORE_RESPONSE.ALL_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0244H
		OFFCORE_RESPONSE.ALL_CODE_RD.LLC_HIT.MISS_DRAM_N		300400244H
		OFFCORE_RESPONSE.ALL_DATA_RD.LLC_HIT.ANY_RESPONSE_N		3F803C0091H
		OFFCORE_RESPONSE.ALL_DATA_RD.LLC_MISS.DRAM_N		300400091H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.ANY_RESPONSE_N		3F803C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.HITM_OTHER_CORE_N		10003C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_MISS.DRAM_N		300400240H
		OFFCORE_RESPONSE.ALL_PF_DATA_RD.LLC_MISS.DRAM_N		300400090H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.SNOOP_MISS_N		2003C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_MISS.DRAM_N		300400120H
		OFFCORE_RESPONSE.ALL_READS.LLC_MISS.DRAM_N		3004003F7H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.SNOOP_MISS_N		2003C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_MISS.DRAM_N		300400122H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_HIT.HITM_OTHER_CORE_N		10003C0004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.DRAM_N		300400004H
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.DRAM_N		300400001H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0002H

**Table 19-16. 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 (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.SNOOP_MISS_N		2003C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_MISS.DRAM_N		300400002H
		OFFCORE_RESPONSE.OTHER.ANY_RESPONSE_N		18000H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0040H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_HIT.HITM_OTHER_CORE_N		10003C0040H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0040H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0040H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_MISS.DRAM_N		300400040H
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.DRAM_N		300400010H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.SNOOP_MISS_N		2003C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_MISS.DRAM_N		300400020H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0200H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_HIT.HITM_OTHER_CORE_N		10003C0200H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0200H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0200H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_MISS.DRAM_N		300400200H
		OFFCORE_RESPONSE.PF_LLC_DATA_RD.LLC_MISS.DRAM_N		300400080H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.SNOOP_MISS_N		2003C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_MISS.DRAM_N		300400100H

Non-architecture performance monitoring events in the processor core that are applicable only to Intel Xeon processor E5 family (and Intel Core i7-3930 processor) based on Intel microarchitecture code name Sandy Bridge, with CPUID signature of DisplayFamily\_DisplayModel 06\_2DH, are listed in Table 19-17.

**Table 19-17. Performance Events Applicable only to the Processor Core of Intel® Xeon® Processor E5 Family**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY	Additional Configuration: Disable BL bypass and direct2core, and if the memory is remotely homed. The count is not reliable If the memory is locally homed.	
D1H	04H	MEM_LOAD_UOPS_RETIRED.LLC_HIT	Additional Configuration: Disable BL bypass. Supports PEBS.	

**Table 19-17. Performance Events Applicable only to the Processor Core of Intel® Xeon® Processor E5 Family**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D1H	20H	MEM_LOAD_UOPS_RETIRED.LLC_MISS	Additional Configuration: Disable BL bypass and direct2core. Supports PEBS.	
D2H	01H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS	Additional Configuration: Disable bypass. Supports PEBS.	
D2H	02H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT	Additional Configuration: Disable bypass. Supports PEBS.	
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Additional Configuration: Disable bypass. Supports PEBS.	
D2H	08H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE	Additional Configuration: Disable bypass. Supports PEBS.	
D3H	01H	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM	Retired load uops which data sources were data missed LLC but serviced by local DRAM. Supports PEBS.	Disable BL bypass and direct2core (see MSR 3C9H).
D3H	04H	MEM_LOAD_UOPS_LLC_MISS_RETIRED.REMOTE_DRAM	Retired load uops which data sources were data missed LLC but serviced by remote DRAM. Supports PEBS.	Disable BL bypass and direct2core (see MSR 3C9H).
B7H/BBH	01H	OFF_CORE_RESPONSE_N	Sub-events of OFF_CORE_RESPONSE_N (suffix N = 0, 1) programmed using MSR 01A6H/01A7H with values shown in the comment column.	
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.ANY_RESPONSE_N		3FFF00004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.LOCAL_DRAM_N		600400004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.REMOTE_DRAM_N		67F800004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.REMOTE_HIT_FWD_N		87F800004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.REMOTE_HITM_N		107FC00004H
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.ANY_DRAM_N		67FC00001H
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.ANY_RESPONSE_N		3F803C0001H
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.LOCAL_DRAM_N		600400001H
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.REMOTE_DRAM_N		67F800001H
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.REMOTE_HIT_FWD_N		87F800001H
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.REMOTE_HITM_N		107FC00001H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_MISS.ANY_RESPONSE_N		3F803C0040H
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.ANY_DRAM_N		67FC00010H
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.ANY_RESPONSE_N		3F803C0010H
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.LOCAL_DRAM_N		600400010H
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.REMOTE_DRAM_N		67F800010H
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.REMOTE_HIT_FWD_N		87F800010H
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.REMOTE_HITM_N		107FC00010H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_MISS.ANY_RESPONSE_N		3FFF00200H
		OFFCORE_RESPONSE.PF_LLC_DATA_RD.LLC_MISS.ANY_RESPONSE_N		3FFF00080H

Model-specific performance monitoring events that are located in the uncore sub-system are implementation specific between different platforms using processors based on Intel microarchitecture code name Sandy Bridge. Processors with CPUID signature of DisplayFamily\_DisplayModel 06\_2AH support performance events listed in Table 19-18.

**Table 19-18. Performance Events In the Processor Uncore for 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series**

Event Num. <sup>1</sup>	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.	
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.	

**Table 19-18. Performance Events In the Processor Uncore for 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series (Contd.)**

Event Num. <sup>1</sup>	Umask Value	Event Mask Mnemonic	Description	Comment
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.

## 19.9 PERFORMANCE MONITORING EVENTS FOR INTEL® CORE™ I7 PROCESSOR FAMILY AND INTEL® XEON® PROCESSOR FAMILY

Processors based on the Intel microarchitecture code name Nehalem support the architectural and model-specific performance monitoring events listed in Table 19-1 and Table 19-19. The events in Table 19-19 generally applies to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following values: 06\_1AH, 06\_1EH, 06\_1FH, and 06\_2EH. However, Intel Xeon processors with CPUID signature of DisplayFamily\_DisplayModel 06\_2EH have a small number of events that are not supported in processors with CPUID signature 06\_1AH, 06\_1EH, and 06\_1FH. These events are noted in the comment column.

In addition, these processors (CPUID signature of DisplayFamily\_DisplayModel 06\_1AH, 06\_1EH, 06\_1FH) also support the following model-specific, product-specific uncore performance monitoring events listed in Table 19-20.

Fixed counters in the core PMU support the architecture events defined in Table 19-2.

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
04H	07H	SB_DRAIN.ANY	Counts the number of store buffer drains.	
06H	04H	STORE_BLOCKS.AT_RET	Counts number of loads delayed with at-Retirement block code. The following loads need to be executed at retirement and wait for all senior stores on the same thread to be drained: load splitting across 4K boundary (page split), load accessing uncacheable (UC or WC) memory, load lock, and load with page table in UC or WC memory region.	
06H	08H	STORE_BLOCKS.L1D_BLOCK	Cacheable loads delayed with L1D block code.	
07H	01H	PARTIAL_ADDRESS_ALIAS	Counts false dependency due to partial address aliasing.	
08H	01H	DTLB_LOAD_MISSES.ANY	Counts all load misses that cause a page walk.	
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED	Counts number of completed page walks due to load miss in the STLB.	
08H	10H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits.	
08H	20H	DTLB_LOAD_MISSES.PDE_MISSES	Number of DTLB cache load misses where the low part of the linear to physical address translation was missed.	
08H	80H	DTLB_LOAD_MISSES.LARGE_WALK_COMPLETED	Counts number of completed large page walks due to load miss in the STLB.	

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
OBH	01H	MEM_INST_RETIRED.LOADS	Counts the number of instructions with an architecturally-visible load retired on the architected path.	
OBH	02H	MEM_INST_RETIRED.STORES	Counts the number of instructions with an architecturally-visible store retired on the architected path.	
OBH	10H	MEM_INST_RETIRED.LATENCY_ABOVE_THRESHOLD	Counts the number of instructions exceeding the latency specified with Id_lat facility.	In conjunction with Id_lat facility.
OCH	01H	MEM_STORE_RETIRED.DTLB_MISS	The event counts the number of retired stores that missed the DTLB. The DTLB miss is not counted if the store operation causes a fault. Does not counter prefetches. Counts both primary and secondary misses to the TLB.	
OEH	01H	UOPS_ISSUED.ANY	Counts the number of Uops issued by the Register Allocation Table to the Reservation Station, i.e. the UOPs issued from the front end to the back end.	
OEH	01H	UOPS_ISSUED.STALLED_CYCLE S	Counts the number of cycles no Uops issued by the Register Allocation Table to the Reservation Station, i.e. the UOPs issued from the front end to the back end.	Set "invert=1, cmask = 1".
OEH	02H	UOPS_ISSUED.FUSED	Counts the number of fused Uops that were issued from the Register Allocation Table to the Reservation Station.	
OFH	01H	MEM_UNCORE_RETIRED.L3_DATA_MISS_UNKNOWN	Counts number of memory load instructions retired where the memory reference missed L3 and data source is unknown.	Available only for CPUID signature 06_2EH.
OFH	02H	MEM_UNCORE_RETIRED.OTHER_CORE_L2_HITM	Counts number of memory load instructions retired where the memory reference hit modified data in a sibling core residing on the same socket.	
OFH	08H	MEM_UNCORE_RETIRED.REMOTE_CACHE_LOCAL_HOME_HIT	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and L3 caches and HIT in a remote socket's cache. Only counts locally homed lines.	
OFH	10H	MEM_UNCORE_RETIRED.REMOTE_DRAM	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and L3 caches and was remotely homed. This includes both DRAM access and HITM in a remote socket's cache for remotely homed lines.	
OFH	20H	MEM_UNCORE_RETIRED.LOCAL_DRAM	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and L3 caches and required a local socket memory reference. This includes locally homed cachelines that were in a modified state in another socket.	
OFH	80H	MEM_UNCORE_RETIRED.UNCACHEABLE	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and L3 caches and to perform I/O.	Available only for CPUID signature 06_2EH.



**Table 19-19. Performance Events In the Processor Core for  
Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
10H	01H	FP_COMP_OPS_EXE.X87	Counts the number of FP Computational Uops Executed. The number of FADD, FSUB, FCOM, FMULs, integer MULs and IMULs, FDIVs, FPREMs, FSQRTS, integer DIVs, and IDIVs. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	
10H	02H	FP_COMP_OPS_EXE.MMX	Counts number of MMX Uops executed.	
10H	04H	FP_COMP_OPS_EXE.SSE_FP	Counts number of SSE and SSE2 FP uops executed.	
10H	08H	FP_COMP_OPS_EXE.SSE2_INTEGER	Counts number of SSE2 integer uops executed.	
10H	10H	FP_COMP_OPS_EXE.SSE_FP_PACKED	Counts number of SSE FP packed uops executed.	
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR	Counts number of SSE FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_SINGLE_PRECISION	Counts number of SSE* FP single precision uops executed.	
10H	80H	FP_COMP_OPS_EXE.SSE_DOUBLE_PRECISION	Counts number of SSE* FP double precision uops executed.	
12H	01H	SIMD_INT_128.PACKED_MPY	Counts number of 128 bit SIMD integer multiply operations.	
12H	02H	SIMD_INT_128.PACKED_SHIFT	Counts number of 128 bit SIMD integer shift operations.	
12H	04H	SIMD_INT_128.PACK	Counts number of 128 bit SIMD integer pack operations.	
12H	08H	SIMD_INT_128.UNPACK	Counts number of 128 bit SIMD integer unpack operations.	
12H	10H	SIMD_INT_128.PACKED_LOGICAL	Counts number of 128 bit SIMD integer logical operations.	
12H	20H	SIMD_INT_128.PACKED_ARITH	Counts number of 128 bit SIMD integer arithmetic operations.	
12H	40H	SIMD_INT_128.SHUFFLE_MOVE	Counts number of 128 bit SIMD integer shuffle and move operations.	
13H	01H	LOAD_DISPATCH.RS	Counts number of loads dispatched from the Reservation Station that bypass the Memory Order Buffer.	
13H	02H	LOAD_DISPATCH.RS_DELAYED	Counts the number of delayed RS dispatches at the stage latch. If an RS dispatch cannot bypass to LB, it has another chance to dispatch from the one-cycle delayed staging latch before it is written into the LB.	
13H	04H	LOAD_DISPATCH.MOB	Counts the number of loads dispatched from the Reservation Station to the Memory Order Buffer.	
13H	07H	LOAD_DISPATCH.ANY	Counts all loads dispatched from the Reservation Station.	



**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
14H	01H	ARITH.CYCLES_DIV_BUSY	Counts the number of cycles the divider is busy executing divide or square root operations. The divide can be integer, X87 or Streaming SIMD Extensions (SSE). The square root operation can be either X87 or SSE. Set 'edge =1, invert=1, cmask=1' to count the number of divides.	Count may be incorrect When SMT is on.
14H	02H	ARITH.MUL	Counts the number of multiply operations executed. This includes integer as well as floating point multiply operations but excludes DPPS mul and MPSAD.	Count may be incorrect When SMT is on.
17H	01H	INST_QUEUE_WRITES	Counts the number of instructions written into the instruction queue every cycle.	
18H	01H	INST_DECODED.DECO	Counts number of instructions that require decoder 0 to be decoded. Usually, this means that the instruction maps to more than 1 uop.	
19H	01H	TWO_UOP_INSTS_DECODED	An instruction that generates two uops was decoded.	
1EH	01H	INST_QUEUE_WRITE_CYCLES	This event counts the number of cycles during which instructions are written to the instruction queue. Dividing this counter by the number of instructions written to the instruction queue (INST_QUEUE_WRITES) yields the average number of instructions decoded each cycle. If this number is less than four and the pipe stalls, this indicates that the decoder is failing to decode enough instructions per cycle to sustain the 4-wide pipeline.	If SSE* instructions that are 6 bytes or longer arrive one after another, then front end throughput may limit execution speed.
20H	01H	LSD_OVERFLOW	Counts number of loops that can't stream from the instruction queue.	
24H	01H	L2_RQSTS.LD_HIT	Counts number of loads that hit the L2 cache. L2 loads include both L1D demand misses as well as L1D prefetches. L2 loads can be rejected for various reasons. Only non rejected loads are counted.	
24H	02H	L2_RQSTS.LD_MISS	Counts the number of loads that miss the L2 cache. L2 loads include both L1D demand misses as well as L1D prefetches.	
24H	03H	L2_RQSTS.LOADS	Counts all L2 load requests. L2 loads include both L1D demand misses as well as L1D prefetches.	
24H	04H	L2_RQSTS.RFO_HIT	Counts the number of store RFO requests that hit the L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches. Count includes WC memory requests, where the data is not fetched but the permission to write the line is required.	
24H	08H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches.	

**Table 19-19. Performance Events In the Processor Core for  
Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	0CH	L2_RQSTS.RFOS	Counts all L2 store RFO requests. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches.	
24H	10H	L2_RQSTS.IFETCH_HIT	Counts number of instruction fetches that hit the L2 cache. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	20H	L2_RQSTS.IFETCH_MISS	Counts number of instruction fetches that miss the L2 cache. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	30H	L2_RQSTS.IFETCHES	Counts all instruction fetches. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	40H	L2_RQSTS.PREFETCH_HIT	Counts L2 prefetch hits for both code and data.	
24H	80H	L2_RQSTS.PREFETCH_MISS	Counts L2 prefetch misses for both code and data.	
24H	C0H	L2_RQSTS.PREFETCHES	Counts all L2 prefetches for both code and data.	
24H	AAH	L2_RQSTS.MISS	Counts all L2 misses for both code and data.	
24H	FFH	L2_RQSTS.REFERENCES	Counts all L2 requests for both code and data.	
26H	01H	L2_DATA_RQSTS.DEMAND.I_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the I (invalid) state, i.e., a cache miss. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	02H	L2_DATA_RQSTS.DEMAND.S_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the S (shared) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	04H	L2_DATA_RQSTS.DEMAND.E_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the E (exclusive) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	08H	L2_DATA_RQSTS.DEMAND.M_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the M (modified) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	0FH	L2_DATA_RQSTS.DEMAND.MESI	Counts all L2 data demand requests. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	10H	L2_DATA_RQSTS.PREFETCH.I_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the I (invalid) state, i.e., a cache miss.	
26H	20H	L2_DATA_RQSTS.PREFETCH.S_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the S (shared) state. A prefetch RFO will miss on an S state line, while a prefetch read will hit on an S state line.	
26H	40H	L2_DATA_RQSTS.PREFETCH.E_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the E (exclusive) state.	

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
26H	80H	L2_DATA_RQSTS.PREFETCH.M_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the M (modified) state.	
26H	F0H	L2_DATA_RQSTS.PREFETCH.MESI	Counts all L2 prefetch requests.	
26H	FFH	L2_DATA_RQSTS.ANY	Counts all L2 data requests.	
27H	01H	L2_WRITE.RFO.I_STATE	Counts number of L2 demand store RFO requests where the cache line to be loaded is in the I (invalid) state, i.e., a cache miss. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	02H	L2_WRITE.RFO.S_STATE	Counts number of L2 store RFO requests where the cache line to be loaded is in the S (shared) state. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	08H	L2_WRITE.RFO.M_STATE	Counts number of L2 store RFO requests where the cache line to be loaded is in the M (modified) state. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	0EH	L2_WRITE.RFO.HIT	Counts number of L2 store RFO requests where the cache line to be loaded is in either the S, E or M states. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	0FH	L2_WRITE.RFO.MESI	Counts all L2 store RFO requests. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	10H	L2_WRITE.LOCK.I_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the I (invalid) state, for example, a cache miss.	
27H	20H	L2_WRITE.LOCK.S_STATE	Counts number of L2 lock RFO requests where the cache line to be loaded is in the S (shared) state.	
27H	40H	L2_WRITE.LOCK.E_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the E (exclusive) state.	
27H	80H	L2_WRITE.LOCK.M_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the M (modified) state.	
27H	E0H	L2_WRITE.LOCK.HIT	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in either the S, E, or M state.	
27H	F0H	L2_WRITE.LOCK.MESI	Counts all L2 demand lock RFO requests.	
28H	01H	L1D_WB_L2.I_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the I (invalid) state, i.e., a cache miss.	
28H	02H	L1D_WB_L2.S_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the S state.	
28H	04H	L1D_WB_L2.E_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the E (exclusive) state.	
28H	08H	L1D_WB_L2.M_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the M (modified) state.	

**Table 19-19. Performance Events In the Processor Core for  
Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
28H	0FH	L1D_WB_L2.MESI	Counts all L1 writebacks to the L2 .	
2EH	4FH	L3_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache. The event count includes speculative traffic but excludes cache line fills due to a L2 hardware-prefetch. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	See Table 19-1.
2EH	41H	L3_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache. The event count may include speculative traffic but excludes cache line fills due to L2 hardware-prefetches. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	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_UNHALTED.REF_P	Increments at the frequency of TSC when not halted.	See Table 19-1.
40H	01H	L1D_CACHE_LD.I_STATE	Counts L1 data cache read requests where the cache line to be loaded is in the I (invalid) state, i.e. the read request missed the cache.	Counter 0, 1 only.
40H	02H	L1D_CACHE_LD.S_STATE	Counts L1 data cache read requests where the cache line to be loaded is in the S (shared) state.	Counter 0, 1 only.
40H	04H	L1D_CACHE_LD.E_STATE	Counts L1 data cache read requests where the cache line to be loaded is in the E (exclusive) state.	Counter 0, 1 only.
40H	08H	L1D_CACHE_LD.M_STATE	Counts L1 data cache read requests where the cache line to be loaded is in the M (modified) state.	Counter 0, 1 only.
40H	0FH	L1D_CACHE_LD.MESI	Counts L1 data cache read requests.	Counter 0, 1 only.
41H	02H	L1D_CACHE_ST.S_STATE	Counts L1 data cache store RFO requests where the cache line to be loaded is in the S (shared) state.	Counter 0, 1 only.
41H	04H	L1D_CACHE_ST.E_STATE	Counts L1 data cache store RFO requests where the cache line to be loaded is in the E (exclusive) state.	Counter 0, 1 only.
41H	08H	L1D_CACHE_ST.M_STATE	Counts L1 data cache store RFO requests where cache line to be loaded is in the M (modified) state.	Counter 0, 1 only.
42H	01H	L1D_CACHE_LOCK.HIT	Counts retired load locks that hit in the L1 data cache or hit in an already allocated fill buffer. The lock portion of the load lock transaction must hit in the L1D.	The initial load will pull the lock into the L1 data cache. Counter 0, 1 only.
42H	02H	L1D_CACHE_LOCK.S_STATE	Counts L1 data cache retired load locks that hit the target cache line in the shared state.	Counter 0, 1 only.
42H	04H	L1D_CACHE_LOCK.E_STATE	Counts L1 data cache retired load locks that hit the target cache line in the exclusive state.	Counter 0, 1 only.

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
42H	08H	L1D_CACHE_LOCK.M_STATE	Counts L1 data cache retired load locks that hit the target cache line in the modified state.	Counter 0, 1 only.
43H	01H	L1D_ALL_REF.ANY	Counts all references (uncached, speculated and retired) to the L1 data cache, including all loads and stores with any memory types. The event counts memory accesses only when they are actually performed. For example, a load blocked by unknown store address and later performed is only counted once.	The event does not include non-memory accesses, such as I/O accesses. Counter 0, 1 only.
43H	02H	L1D_ALL_REF.CACHEABLE	Counts all data reads and writes (speculated and retired) from cacheable memory, including locked operations.	Counter 0, 1 only.
49H	01H	DTLB_MISSES.ANY	Counts the number of misses in the STLB which causes a page walk.	
49H	02H	DTLB_MISSES.WALK_COMPLETED	Counts number of misses in the STLB which resulted in a completed page walk.	
49H	10H	DTLB_MISSES.STLB_HIT	Counts the number of DTLB first level misses that hit in the second level TLB. This event is only relevant if the core contains multiple DTLB levels.	
49H	20H	DTLB_MISSES.PDE_MISS	Number of DTLB misses caused by low part of address, includes references to 2M pages because 2M pages do not use the PDE.	
49H	80H	DTLB_MISSES.LARGE_WALK_COMPLETED	Counts number of misses in the STLB which resulted in a completed page walk for large pages.	
4CH	01H	LOAD_HIT_PRE	Counts load operations sent to the L1 data cache while a previous SSE prefetch instruction to the same cache line has started prefetching but has not yet finished.	
4EH	01H	L1D_PREFETCH.REQUESTS	Counts number of hardware prefetch requests dispatched out of the prefetch FIFO.	
4EH	02H	L1D_PREFETCH.MISS	Counts number of hardware prefetch requests that miss the L1D. There are two prefetchers in the L1D. A streamer, which predicts lines sequentially after this one should be fetched, and the IP prefetcher that remembers access patterns for the current instruction. The streamer prefetcher stops on an L1D hit, while the IP prefetcher does not.	
4EH	04H	L1D_PREFETCH.TRIGGERS	Counts number of prefetch requests triggered by the Finite State Machine and pushed into the prefetch FIFO. Some of the prefetch requests are dropped due to overwrites or competition between the IP index prefetcher and streamer prefetcher. The prefetch FIFO contains 4 entries.	
51H	01H	L1D.REPL	Counts the number of lines brought into the L1 data cache.	Counter 0, 1 only.
51H	02H	L1D.M_REPL	Counts the number of modified lines brought into the L1 data cache.	Counter 0, 1 only.
51H	04H	L1D.M_EVICT	Counts the number of modified lines evicted from the L1 data cache due to replacement.	Counter 0, 1 only.

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
51H	08H	L1D.M_SNOOP_EVICT	Counts the number of modified lines evicted from the L1 data cache due to snoop HITM intervention.	Counter 0, 1 only.
52H	01H	L1D_CACHE_PREFETCH_LOCK_FB_HIT	Counts the number of cacheable load lock speculated instructions accepted into the fill buffer.	
53H	01H	L1D_CACHE_LOCK_FB_HIT	Counts the number of cacheable load lock speculated or retired instructions accepted into the fill buffer.	
63H	01H	CACHE_LOCK_CYCLES.L1D_L2	Cycle count during which the L1D and L2 are locked. A lock is asserted when there is a locked memory access, due to uncacheable memory, a locked operation that spans two cache lines, or a page walk from an uncacheable page table.	Counter 0, 1 only. L1D and L2 locks have a very high performance penalty and it is highly recommended to avoid such accesses.
63H	02H	CACHE_LOCK_CYCLES.L1D	Counts the number of cycles that cacheline in the L1 data cache unit is locked.	Counter 0, 1 only.
6CH	01H	IO_TRANSACTIONS	Counts the number of completed I/O transactions.	
80H	01H	L1I.HITS	Counts all instruction fetches that hit the L1 instruction cache.	
80H	02H	L1I.MISSES	Counts all instruction fetches that miss the L1I cache. This includes instruction cache misses, streaming buffer misses, victim cache misses and uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.	
80H	03H	L1I.READS	Counts all instruction fetches, including uncacheable fetches that bypass the L1I.	
80H	04H	L1I.CYCLES_STALLED	Cycle counts for which an instruction fetch stalls due to a L1I cache miss, ITLB miss or ITLB fault.	
82H	01H	LARGE_ITLB.HIT	Counts number of large ITLB hits.	
85H	01H	ITLB_MISSES.ANY	Counts the number of misses in all levels of the ITLB which causes a page walk.	
85H	02H	ITLB_MISSES.WALK_COMPLETED	Counts number of misses in all levels of the ITLB which resulted in a completed page walk.	
87H	01H	ILD_STALL.LCP	Cycles Instruction Length Decoder stalls due to length changing prefixes: 66, 67 or REX.W (for Intel 64) instructions which change the length of the decoded instruction.	
87H	02H	ILD_STALL.MRU	Instruction Length Decoder stall cycles due to Branch Prediction Unit (PBU) Most Recently Used (MRU) bypass.	
87H	04H	ILD_STALL.IQ_FULL	Stall cycles due to a full instruction queue.	
87H	08H	ILD_STALL.REGEN	Counts the number of regen stalls.	
87H	0FH	ILD_STALL.ANY	Counts any cycles the Instruction Length Decoder is stalled.	
88H	01H	BR_INST_EXEC.COND	Counts the number of conditional near branch instructions executed, but not necessarily retired.	

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
88H	02H	BR_INST_EXEC.DIRECT	Counts all unconditional near branch instructions excluding calls and indirect branches.	
88H	04H	BR_INST_EXEC.INDIRECT_NON_CALL	Counts the number of executed indirect near branch instructions that are not calls.	
88H	07H	BR_INST_EXEC.NON_CALLS	Counts all non-call near branch instructions executed, but not necessarily retired.	
88H	08H	BR_INST_EXEC.RETURN_NEAR	Counts indirect near branches that have a return mnemonic.	
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Counts unconditional near call branch instructions, excluding non-call branch, executed.	
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Counts indirect near calls, including both register and memory indirect, executed.	
88H	30H	BR_INST_EXEC.NEAR_CALLS	Counts all near call branches executed, but not necessarily retired.	
88H	40H	BR_INST_EXEC.TAKEN	Counts taken near branches executed, but not necessarily retired.	
88H	7FH	BR_INST_EXEC.ANY	Counts all near executed branches (not necessarily retired). This includes only instructions and not micro-op branches. Frequent branching is not necessarily a major performance issue. However frequent branch mispredictions may be a problem.	
89H	01H	BR_MISP_EXEC.COND	Counts the number of mispredicted conditional near branch instructions executed, but not necessarily retired.	
89H	02H	BR_MISP_EXEC.DIRECT	Counts mispredicted macro unconditional near branch instructions, excluding calls and indirect branches (should always be 0).	
89H	04H	BR_MISP_EXEC.INDIRECT_NON_CALL	Counts the number of executed mispredicted indirect near branch instructions that are not calls.	
89H	07H	BR_MISP_EXEC.NON_CALLS	Counts mispredicted non-call near branches executed, but not necessarily retired.	
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Counts mispredicted indirect branches that have a near return mnemonic.	
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Counts mispredicted non-indirect near calls executed, (should always be 0).	
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Counts mispredicted indirect near calls executed, including both register and memory indirect.	
89H	30H	BR_MISP_EXEC.NEAR_CALLS	Counts all mispredicted near call branches executed, but not necessarily retired.	
89H	40H	BR_MISP_EXEC.TAKEN	Counts executed mispredicted near branches that are taken, but not necessarily retired.	
89H	7FH	BR_MISP_EXEC.ANY	Counts the number of mispredicted near branch instructions that were executed, but not necessarily retired.	

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A2H	01H	RESOURCE_STALLS.ANY	Counts the number of Allocator resource related stalls. Includes register renaming buffer entries, memory buffer entries. In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, such as if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	Does not include stalls due to SuperQ (off core) queue full, too many cache misses, etc.
A2H	02H	RESOURCE_STALLS.LOAD	Counts the cycles of stall due to lack of load buffer for load operation.	
A2H	04H	RESOURCE_STALLS.RS_FULL	This event counts the number of cycles when the number of instructions in the pipeline waiting for execution reaches the limit the processor can handle. A high count of this event indicates that there are long latency operations in the pipe (possibly load and store operations that miss the L2 cache, or instructions dependent upon instructions further down the pipeline that have yet to retire.	When RS is full, new instructions cannot enter the reservation station and start execution.
A2H	08H	RESOURCE_STALLS.STORE	This event counts the number of cycles that a resource related stall will occur due to the number of store instructions reaching the limit of the pipeline, (i.e. all store buffers are used). The stall ends when a store instruction commits its data to the cache or memory.	
A2H	10H	RESOURCE_STALLS.ROB_FULL	Counts the cycles of stall due to re-order buffer full.	
A2H	20H	RESOURCE_STALLS.FPCW	Counts the number of cycles while execution was stalled due to writing the floating-point unit (FPU) control word.	
A2H	40H	RESOURCE_STALLS.MXCSR	Stalls due to the MXCSR register rename occurring to close to a previous MXCSR rename. The MXCSR provides control and status for the MMX registers.	
A2H	80H	RESOURCE_STALLS.OTHER	Counts the number of cycles while execution was stalled due to other resource issues.	
A6H	01H	MACRO_INSTS.FUSIONS_DECODED	Counts the number of instructions decoded that are macro-fused but not necessarily executed or retired.	
A7H	01H	BACLEAR_FORCE_IQ	Counts number of times a BACLEAR was forced by the Instruction Queue. The IQ is also responsible for providing conditional branch prediction direction based on a static scheme and dynamic data provided by the L2 Branch Prediction Unit. If the conditional branch target is not found in the Target Array and the IQ predicts that the branch is taken, then the IQ will force the Branch Address Calculator to issue a BACLEAR. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline.	
A8H	01H	LSD.UOPS	Counts the number of micro-ops delivered by loop stream detector.	Use cmask=1 and invert to count cycles.
AEH	01H	ITLB_FLUSH	Counts the number of ITLB flushes.	



**Table 19-19. Performance Events In the Processor Core for  
Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B0H	40H	OFFCORE_REQUESTS.L1D_WRITEBACK	Counts number of L1D writebacks to the uncore.	
B1H	01H	UOPS_EXECUTED.PORT0	Counts number of uops executed that were issued on port 0. Port 0 handles integer arithmetic, SIMD and FP add uops.	
B1H	02H	UOPS_EXECUTED.PORT1	Counts number of uops executed that were issued on port 1. Port 1 handles integer arithmetic, SIMD, integer shift, FP multiply and FP divide uops.	
B1H	04H	UOPS_EXECUTED.PORT2_CORE	Counts number of uops executed that were issued on port 2. Port 2 handles the load uops. This is a core count only and cannot be collected per thread.	
B1H	08H	UOPS_EXECUTED.PORT3_CORE	Counts number of uops executed that were issued on port 3. Port 3 handles store uops. This is a core count only and cannot be collected per thread.	
B1H	10H	UOPS_EXECUTED.PORT4_CORE	Counts number of uops executed that where issued on port 4. Port 4 handles the value to be stored for the store uops issued on port 3. This is a core count only and cannot be collected per thread.	
B1H	1FH	UOPS_EXECUTED.CORE_ACTIVE_CYCLES_NO_PORT5	Counts cycles when the uops executed were issued from any ports except port 5. Use Cmask=1 for active cycles; Cmask=0 for weighted cycles. Use CMask=1, Invert=1 to count P0-4 stalled cycles. Use Cmask=1, Edge=1, Invert=1 to count P0-4 stalls.	
B1H	20H	UOPS_EXECUTED.PORT5	Counts number of uops executed that where issued on port 5.	
B1H	3FH	UOPS_EXECUTED.CORE_ACTIVE_CYCLES	Counts cycles when the uops are executing. Use Cmask=1 for active cycles; Cmask=0 for weighted cycles. Use CMask=1, Invert=1 to count P0-4 stalled cycles. Use Cmask=1, Edge=1, Invert=1 to count P0-4 stalls.	
B1H	40H	UOPS_EXECUTED.PORT015	Counts number of uops executed that where issued on port 0, 1, or 5.	Use cmask=1, invert=1 to count stall cycles.
B1H	80H	UOPS_EXECUTED.PORT234	Counts number of uops executed that where issued on port 2, 3, or 4.	
B2H	01H	OFFCORE_REQUESTS_SQ_FULL	Counts number of cycles the SQ is full to handle off-core requests.	
B7H	01H	OFF_CORE_RESPONSE_0	See Section 18.3.1.1.3, "Off-core Response Performance Monitoring in the Processor Core".	Requires programming MSR 01A6H.
B8H	01H	SNOOP_RESPONSE.HIT	Counts HIT snoop response sent by this thread in response to a snoop request.	
B8H	02H	SNOOP_RESPONSE.HITE	Counts HIT E snoop response sent by this thread in response to a snoop request.	
B8H	04H	SNOOP_RESPONSE.HITM	Counts HIT M snoop response sent by this thread in response to a snoop request.	
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.6.3, "Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)".	Requires programming MSR 01A7H.

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C0H	00H	INST_RETIRED.ANY_P	See Table 19-1. Notes: INST_RETIRED.ANY is counted by a designated fixed counter. INST_RETIRED.ANY_P is counted by a programmable counter and is an architectural performance event. Event is supported if CPUID.A.EBX[1] = 0.	Counting: Faulting executions of GETSEC/VM entry/VM Exit/MWait will not count as retired instructions.
C0H	02H	INST_RETIRED.X87	Counts the number of MMX instructions retired.	
C0H	04H	INST_RETIRED.MMX	Counts the number of floating point computational operations retired: floating point computational operations executed by the assist handler and sub-operations of complex floating point instructions like transcendental instructions.	
C2H	01H	UOPS_RETIRED.ANY	Counts the number of micro-ops retired, (macro-fused=1, micro-fused=2, others=1; maximum count of 8 per cycle). Most instructions are composed of one or two micro-ops. Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists.	Use cmask=1 and invert to count active cycles or stalled cycles.
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	
C2H	04H	UOPS_RETIRED.MACRO_FUSED	Counts number of macro-fused uops retired.	
C3H	01H	MACHINE_CLEAR.CYCLES	Counts the cycles machine clear is asserted.	
C3H	02H	MACHINE_CLEAR.MEM_ORDER	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEAR.SMC	Counts the number of times that a program writes to a code section. Self-modifying code causes a severe penalty in all Intel 64 and IA-32 processors. The modified cache line is written back to the L2 and L3caches.	
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.	
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Counts the number of direct & indirect near unconditional calls retired.	
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement.	See Table 19-1.
C5H	02H	BR_MISP_RETIRED.NEAR_CALL	Counts mispredicted direct & indirect near unconditional retired calls.	
C7H	01H	SSEX_UOPS_RETIRED.PACKED_SINGLE	Counts SIMD packed single-precision floating point Uops retired.	
C7H	02H	SSEX_UOPS_RETIRED.SCALAR_SINGLE	Counts SIMD scalar single-precision floating point Uops retired.	
C7H	04H	SSEX_UOPS_RETIRED.PACKED_DOUBLE	Counts SIMD packed double-precision floating point Uops retired.	

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C7H	08H	SSEX_UOPS_RETIRED.SCALAR_DOUBLE	Counts SIMD scalar double-precision floating point Uops retired.	
C7H	10H	SSEX_UOPS_RETIRED.VECTOR_INTEGER	Counts 128-bit SIMD vector integer Uops retired.	
C8H	20H	ITLB_MISS_RETIRED	Counts the number of retired instructions that missed the ITLB when the instruction was fetched.	
CBH	01H	MEM_LOAD_RETIRED.L1D_HIT	Counts number of retired loads that hit the L1 data cache.	
CBH	02H	MEM_LOAD_RETIRED.L2_HIT	Counts number of retired loads that hit the L2 data cache.	
CBH	04H	MEM_LOAD_RETIRED.L3_UNSHARED_HIT	Counts number of retired loads that hit their own, unshared lines in the L3 cache.	
CBH	08H	MEM_LOAD_RETIRED.OTHER_CORE_L2_HIT_HITM	Counts number of retired loads that hit in a sibling core's L2 (on die core). Since the L3 is inclusive of all cores on the package, this is an L3 hit. This counts both clean and modified hits.	
CBH	10H	MEM_LOAD_RETIRED.L3_MISS	Counts number of retired loads that miss the L3 cache. The load was satisfied by a remote socket, local memory or an IOH.	
CBH	40H	MEM_LOAD_RETIRED.HIT_LFB	Counts number of retired loads that miss the L1D and the address is located in an allocated line fill buffer and will soon be committed to cache. This is counting secondary L1D misses.	
CBH	80H	MEM_LOAD_RETIRED.DTLB_MISS	Counts the number of retired loads that missed the DTLB. The DTLB miss is not counted if the load operation causes a fault. This event counts loads from cacheable memory only. The event does not count loads by software prefetches. Counts both primary and secondary misses to the TLB.	
CCH	01H	FP_MMX_TRANS.TO_FP	Counts the first floating-point instruction following any MMX instruction. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
CCH	02H	FP_MMX_TRANS.TO_MMX	Counts the first MMX instruction following a floating-point instruction. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
CCH	03H	FP_MMX_TRANS.ANY	Counts all transitions from floating point to MMX instructions and from MMX instructions to floating point instructions. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
D0H	01H	MACRO_INSTS.DECODED	Counts the number of instructions decoded, (but not necessarily executed or retired).	
D1H	02H	UOPS_DECODED.MS	Counts the number of Uops decoded by the Microcode Sequencer, MS. The MS delivers uops when the instruction is more than 4 uops long or a microcode assist is occurring.	

**Table 19-19. Performance Events In the Processor Core for  
Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D1H	04H	UOPS_DECODED.ESP_FOLDING	Counts number of stack pointer (ESP) instructions decoded: push, pop, call, ret, etc. ESP instructions do not generate a Uop to increment or decrement ESP. Instead, they update an ESP_Offset register that keeps track of the delta to the current value of the ESP register.	
D1H	08H	UOPS_DECODED.ESP_SYNC	Counts number of stack pointer (ESP) sync operations where an ESP instruction is corrected by adding the ESP offset register to the current value of the ESP register.	
D2H	01H	RAT_STALLS.FLAGS	Counts the number of cycles during which execution stalled due to several reasons, one of which is a partial flag register stall. A partial register stall may occur when two conditions are met: 1) an instruction modifies some, but not all, of the flags in the flag register and 2) the next instruction, which depends on flags, depends on flags that were not modified by this instruction.	
D2H	02H	RAT_STALLS.REGISTERS	This event counts the number of cycles instruction execution latency became longer than the defined latency because the instruction used a register that was partially written by previous instruction.	
D2H	04H	RAT_STALLS.ROB_READ_PORT	Counts the number of cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the out-of-order pipeline. Note that, at this stage in the pipeline, additional stalls may occur at the same cycle and prevent the stalled micro-ops from entering the pipe. In such a case, micro-ops retry entering the execution pipe in the next cycle and the ROB-read port stall is counted again.	
D2H	08H	RAT_STALLS.SCOREBOARD	Counts the cycles where we stall due to microarchitecturally required serialization. Microcode scoreboarding stalls.	
D2H	0FH	RAT_STALLS.ANY	Counts all Register Allocation Table stall cycles due to: Cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the execution pipe. Cycles when partial register stalls occurred. Cycles when flag stalls occurred. Cycles floating-point unit (FPU) status word stalls occurred. To count each of these conditions separately use the events: RAT_STALLS.ROB_READ_PORT, RAT_STALLS.PARTIAL, RAT_STALLS.FLAGS, and RAT_STALLS.FPSW.	
D4H	01H	SEG_RENAME_STALLS	Counts the number of stall cycles due to the lack of renaming resources for the ES, DS, FS, and GS segment registers. If a segment is renamed but not retired and a second update to the same segment occurs, a stall occurs in the front end of the pipeline until the renamed segment retires.	
D5H	01H	ES_REG_RENAMES	Counts the number of times the ES segment register is renamed.	

**Table 19-19. Performance Events In the Processor Core for  
Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
DBH	01H	UOP_UNFUSION	Counts unfusion events due to floating-point exception to a fused uop.	
E0H	01H	BR_INST_DECODED	Counts the number of branch instructions decoded.	
E5H	01H	BPU_MISSED_CALL_RET	Counts number of times the Branch Prediction Unit missed predicting a call or return branch.	
E6H	01H	BACLEAR.CLEAR	Counts the number of times the front end is resteeered, mainly when the Branch Prediction Unit cannot provide a correct prediction and this is corrected by the Branch Address Calculator at the front end. This can occur if the code has many branches such that they cannot be consumed by the BPU. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline. The effect on total execution time depends on the surrounding code.	
E6H	02H	BACLEAR.BAD_TARGET	Counts number of Branch Address Calculator clears (BACLEAR) asserted due to conditional branch instructions in which there was a target hit but the direction was wrong. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline.	
E8H	01H	BPU_CLEAR.EARLY	Counts early (normal) Branch Prediction Unit clears: BPU predicted a taken branch after incorrectly assuming that it was not taken.	The BPU clear leads to 2 cycle bubble in the front end.
E8H	02H	BPU_CLEAR.LATE	Counts late Branch Prediction Unit clears due to Most Recently Used conflicts. The PBU clear leads to a 3 cycle bubble in the front end.	
FOH	01H	L2_TRANSACTION.LOAD	Counts L2 load operations due to HW prefetch or demand loads.	
FOH	02H	L2_TRANSACTION.RFO	Counts L2 RFO operations due to HW prefetch or demand RFOs.	
FOH	04H	L2_TRANSACTION.IFETCH	Counts L2 instruction fetch operations due to HW prefetch or demand ifetch.	
FOH	08H	L2_TRANSACTION.PREFETCH	Counts L2 prefetch operations.	
FOH	10H	L2_TRANSACTION.L1D_WB	Counts L1D writeback operations to the L2.	
FOH	20H	L2_TRANSACTION.FILL	Counts L2 cache line fill operations due to load, RFO, L1D writeback or prefetch.	
FOH	40H	L2_TRANSACTION.WB	Counts L2 writeback operations to the L3.	
FOH	80H	L2_TRANSACTION.ANY	Counts all L2 cache operations.	
F1H	02H	L2_LINES_IN.S_STATE	Counts the number of cache lines allocated in the L2 cache in the S (shared) state.	
F1H	04H	L2_LINES_IN.E_STATE	Counts the number of cache lines allocated in the L2 cache in the E (exclusive) state.	
F1H	07H	L2_LINES_IN.ANY	Counts the number of cache lines allocated in the L2 cache.	
F2H	01H	L2_LINES_OUT.DEMAND_CLEAN	Counts L2 clean cache lines evicted by a demand request.	

**Table 19-19. Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
F2H	02H	L2_LINES_OUT.DEMAND_DIRTY	Counts L2 dirty (modified) cache lines evicted by a demand request.	
F2H	04H	L2_LINES_OUT.PREFETCH_CLEAN	Counts L2 clean cache line evicted by a prefetch request.	
F2H	08H	L2_LINES_OUT.PREFETCH_DIRTY	Counts L2 modified cache line evicted by a prefetch request.	
F2H	0FH	L2_LINES_OUT.ANY	Counts all L2 cache lines evicted for any reason.	
F4H	10H	SQ_MISC.SPLIT_LOCK	Counts the number of SQ lock splits across a cache line.	
F6H	01H	SQ_FULL_STALL_CYCLES	Counts cycles the Super Queue is full. Neither of the threads on this core will be able to access the uncore.	
F7H	01H	FP_ASSIST.ALL	Counts the number of floating point operations executed that required micro-code assist intervention. Assists are required in the following cases: SSE instructions (denormal input when the DAZ flag is off or underflow result when the FTZ flag is off); x87 instructions (NaN or denormal are loaded to a register or used as input from memory, division by 0 or underflow output).	
F7H	02H	FP_ASSIST.OUTPUT	Counts number of floating point micro-code assist when the output value (destination register) is invalid.	
F7H	04H	FP_ASSIST.INPUT	Counts number of floating point micro-code assist when the input value (one of the source operands to an FP instruction) is invalid.	
FDH	01H	SIMD_INT_64.PACKED_MPY	Counts number of SIMD integer 64 bit packed multiply operations.	
FDH	02H	SIMD_INT_64.PACKED_SHIFT	Counts number of SIMD integer 64 bit packed shift operations.	
FDH	04H	SIMD_INT_64.PACK	Counts number of SIMD integer 64 bit pack operations.	
FDH	08H	SIMD_INT_64.UNPACK	Counts number of SIMD integer 64 bit unpack operations.	
FDH	10H	SIMD_INT_64.PACKED_LOGICAL	Counts number of SIMD integer 64 bit logical operations.	
FDH	20H	SIMD_INT_64.PACKED_ARITH	Counts number of SIMD integer 64 bit arithmetic operations.	
FDH	40H	SIMD_INT_64.SHUFFLE_MOVE	Counts number of SIMD integer 64 bit shift or move operations.	

Model-specific performance monitoring events that are located in the uncore sub-system are implementation specific between different platforms using processors based on Intel microarchitecture code name Nehalem. Processors with CPUID signature of DisplayFamily\_DisplayModel 06\_1AH, 06\_1EH, and 06\_1FH support performance events listed in Table 19-20.

**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
00H	01H	UNC_GQ_CYCLES_FULL.READ_TRACKER	Uncore cycles Global Queue read tracker is full.	
00H	02H	UNC_GQ_CYCLES_FULL.WRITE_TRACKER	Uncore cycles Global Queue write tracker is full.	
00H	04H	UNC_GQ_CYCLES_FULL.PEER_PROBE_TRACKER	Uncore cycles Global Queue peer probe tracker is full. The peer probe tracker queue tracks snoops from the IOH and remote sockets.	
01H	01H	UNC_GQ_CYCLES_NOT_EMPTY.READ_TRACKER	Uncore cycles were Global Queue read tracker has at least one valid entry.	
01H	02H	UNC_GQ_CYCLES_NOT_EMPTY.WRITE_TRACKER	Uncore cycles were Global Queue write tracker has at least one valid entry.	
01H	04H	UNC_GQ_CYCLES_NOT_EMPTY.PEER_PROBE_TRACKER	Uncore cycles were Global Queue peer probe tracker has at least one valid entry. The peer probe tracker queue tracks IOH and remote socket snoops.	
03H	01H	UNC_GQ_ALLOC.READ_TRACKER	Counts the number of read tracker allocate to deallocate entries. The GQ read tracker allocate to deallocate occupancy count is divided by the count to obtain the average read tracker latency.	
03H	02H	UNC_GQ_ALLOC.RT_L3_MISS	Counts the number GQ read tracker entries for which a full cache line read has missed the L3. The GQ read tracker L3 miss to fill occupancy count is divided by this count to obtain the average cache line read L3 miss latency. The latency represents the time after which the L3 has determined that the cache line has missed. The time between a GQ read tracker allocation and the L3 determining that the cache line has missed is the average L3 hit latency. The total L3 cache line read miss latency is the hit latency + L3 miss latency.	
03H	04H	UNC_GQ_ALLOC.RT_TO_L3_RESP	Counts the number of GQ read tracker entries that are allocated in the read tracker queue that hit or miss the L3. The GQ read tracker L3 hit occupancy count is divided by this count to obtain the average L3 hit latency.	
03H	08H	UNC_GQ_ALLOC.RT_TO_RTID_ACQUIRED	Counts the number of GQ read tracker entries that are allocated in the read tracker, have missed in the L3 and have not acquired a Request Transaction ID. The GQ read tracker L3 miss to RTID acquired occupancy count is divided by this count to obtain the average latency for a read L3 miss to acquire an RTID.	
03H	10H	UNC_GQ_ALLOC.WT_TO_RTID_ACQUIRED	Counts the number of GQ write tracker entries that are allocated in the write tracker, have missed in the L3 and have not acquired a Request Transaction ID. The GQ write tracker L3 miss to RTID occupancy count is divided by this count to obtain the average latency for a write L3 miss to acquire an RTID.	
03H	20H	UNC_GQ_ALLOC.WRITE_TRACKER	Counts the number of GQ write tracker entries that are allocated in the write tracker queue that miss the L3. The GQ write tracker occupancy count is divided by this count to obtain the average L3 write miss latency.	



**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	40H	UNC_GQ_ALLOC.PEER_PROBE_TRACKER	Counts the number of GQ peer probe tracker (snoop) entries that are allocated in the peer probe tracker queue that miss the L3. The GQ peer probe occupancy count is divided by this count to obtain the average L3 peer probe miss latency.	
04H	01H	UNC_GQ_DATA.FROM_QPI	Cycles Global Queue Quickpath Interface input data port is busy importing data from the Quickpath Interface. Each cycle the input port can transfer 8 or 16 bytes of data.	
04H	02H	UNC_GQ_DATA.FROM_QMC	Cycles Global Queue Quickpath Memory Interface input data port is busy importing data from the Quickpath Memory Interface. Each cycle the input port can transfer 8 or 16 bytes of data.	
04H	04H	UNC_GQ_DATA.FROM_L3	Cycles GQ L3 input data port is busy importing data from the Last Level Cache. Each cycle the input port can transfer 32 bytes of data.	
04H	08H	UNC_GQ_DATA.FROM_CORES_02	Cycles GQ Core 0 and 2 input data port is busy importing data from processor cores 0 and 2. Each cycle the input port can transfer 32 bytes of data.	
04H	10H	UNC_GQ_DATA.FROM_CORES_13	Cycles GQ Core 1 and 3 input data port is busy importing data from processor cores 1 and 3. Each cycle the input port can transfer 32 bytes of data.	
05H	01H	UNC_GQ_DATA.TO_QPI_QMC	Cycles GQ QPI and QMC output data port is busy sending data to the Quickpath Interface or Quickpath Memory Interface. Each cycle the output port can transfer 32 bytes of data.	
05H	02H	UNC_GQ_DATA.TO_L3	Cycles GQ L3 output data port is busy sending data to the Last Level Cache. Each cycle the output port can transfer 32 bytes of data.	
05H	04H	UNC_GQ_DATA.TO_CORES	Cycles GQ Core output data port is busy sending data to the Cores. Each cycle the output port can transfer 32 bytes of data.	
06H	01H	UNC_SNP_RESP_TO_LOCAL_HOME.I_STATE	Number of snoop responses to the local home that L3 does not have the referenced cache line.	
06H	02H	UNC_SNP_RESP_TO_LOCAL_HOME.S_STATE	Number of snoop responses to the local home that L3 has the referenced line cached in the S state.	
06H	04H	UNC_SNP_RESP_TO_LOCAL_HOME.FWD_S_STATE	Number of responses to code or data read snoops to the local home that the L3 has the referenced cache line in the E state. The L3 cache line state is changed to the S state and the line is forwarded to the local home in the S state.	
06H	08H	UNC_SNP_RESP_TO_LOCAL_HOME.FWD_I_STATE	Number of responses to read invalidate snoops to the local home that the L3 has the referenced cache line in the M state. The L3 cache line state is invalidated and the line is forwarded to the local home in the M state.	
06H	10H	UNC_SNP_RESP_TO_LOCAL_HOME.CONFLICT	Number of conflict snoop responses sent to the local home.	



**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
06H	20H	UNC_SNP_RESP_TO_LOCAL_HOME.WB	Number of responses to code or data read snoops to the local home that the L3 has the referenced line cached in the M state.	
07H	01H	UNC_SNP_RESP_TO_REMOTE_HOME.I_STATE	Number of snoop responses to a remote home that L3 does not have the referenced cache line.	
07H	02H	UNC_SNP_RESP_TO_REMOTE_HOME.S_STATE	Number of snoop responses to a remote home that L3 has the referenced line cached in the S state.	
07H	04H	UNC_SNP_RESP_TO_REMOTE_HOME.FWD_S_STATE	Number of responses to code or data read snoops to a remote home that the L3 has the referenced cache line in the E state. The L3 cache line state is changed to the S state and the line is forwarded to the remote home in the S state.	
07H	08H	UNC_SNP_RESP_TO_REMOTE_HOME.FWD_I_STATE	Number of responses to read invalidate snoops to a remote home that the L3 has the referenced cache line in the M state. The L3 cache line state is invalidated and the line is forwarded to the remote home in the M state.	
07H	10H	UNC_SNP_RESP_TO_REMOTE_HOME.CONFLICT	Number of conflict snoop responses sent to the local home.	
07H	20H	UNC_SNP_RESP_TO_REMOTE_HOME.WB	Number of responses to code or data read snoops to a remote home that the L3 has the referenced line cached in the M state.	
07H	24H	UNC_SNP_RESP_TO_REMOTE_HOME.HITM	Number of HITM snoop responses to a remote home.	
08H	01H	UNC_L3_HITS.READ	Number of code read, data read and RFO requests that hit in the L3.	
08H	02H	UNC_L3_HITS.WRITE	Number of writeback requests that hit in the L3. Writebacks from the cores will always result in L3 hits due to the inclusive property of the L3.	
08H	04H	UNC_L3_HITS.PROBE	Number of snoops from IOH or remote sockets that hit in the L3.	
08H	03H	UNC_L3_HITS.ANY	Number of reads and writes that hit the L3.	
09H	01H	UNC_L3_MISS.READ	Number of code read, data read and RFO requests that miss the L3.	
09H	02H	UNC_L3_MISS.WRITE	Number of writeback requests that miss the L3. Should always be zero as writebacks from the cores will always result in L3 hits due to the inclusive property of the L3.	
09H	04H	UNC_L3_MISS.PROBE	Number of snoops from IOH or remote sockets that miss the L3.	
09H	03H	UNC_L3_MISS.ANY	Number of reads and writes that miss the L3.	
0AH	01H	UNC_L3_LINES_IN.M_STATE	Counts the number of L3 lines allocated in M state. The only time a cache line is allocated in the M state is when the line was forwarded in M state is forwarded due to a Snoop Read Invalidate Own request.	
0AH	02H	UNC_L3_LINES_IN.E_STATE	Counts the number of L3 lines allocated in E state.	
0AH	04H	UNC_L3_LINES_IN.S_STATE	Counts the number of L3 lines allocated in S state.	

**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
0AH	08H	UNC_L3_LINES_IN.F_STATE	Counts the number of L3 lines allocated in F state.	
0AH	0FH	UNC_L3_LINES_IN.ANY	Counts the number of L3 lines allocated in any state.	
0BH	01H	UNC_L3_LINES_OUT.M_STATE	Counts the number of L3 lines victimized that were in the M state. When the victim cache line is in M state, the line is written to its home cache agent which can be either local or remote.	
0BH	02H	UNC_L3_LINES_OUT.E_STATE	Counts the number of L3 lines victimized that were in the E state.	
0BH	04H	UNC_L3_LINES_OUT.S_STATE	Counts the number of L3 lines victimized that were in the S state.	
0BH	08H	UNC_L3_LINES_OUT.I_STATE	Counts the number of L3 lines victimized that were in the I state.	
0BH	10H	UNC_L3_LINES_OUT.F_STATE	Counts the number of L3 lines victimized that were in the F state.	
0BH	1FH	UNC_L3_LINES_OUT.ANY	Counts the number of L3 lines victimized in any state.	
20H	01H	UNC_QHL_REQUESTS.IOH_READS	Counts number of Quickpath Home Logic read requests from the IOH.	
20H	02H	UNC_QHL_REQUESTS.IOH_WRITES	Counts number of Quickpath Home Logic write requests from the IOH.	
20H	04H	UNC_QHL_REQUESTS.REMOTE_READS	Counts number of Quickpath Home Logic read requests from a remote socket.	
20H	08H	UNC_QHL_REQUESTS.REMOTE_WRITES	Counts number of Quickpath Home Logic write requests from a remote socket.	
20H	10H	UNC_QHL_REQUESTS.LOCAL_READS	Counts number of Quickpath Home Logic read requests from the local socket.	
20H	20H	UNC_QHL_REQUESTS.LOCAL_WRITES	Counts number of Quickpath Home Logic write requests from the local socket.	
21H	01H	UNC_QHL_CYCLES_FULL.IOH	Counts uclk cycles all entries in the Quickpath Home Logic IOH are full.	
21H	02H	UNC_QHL_CYCLES_FULL.REMOTE	Counts uclk cycles all entries in the Quickpath Home Logic remote tracker are full.	
21H	04H	UNC_QHL_CYCLES_FULL.LOCAL	Counts uclk cycles all entries in the Quickpath Home Logic local tracker are full.	
22H	01H	UNC_QHL_CYCLES_NOT_EMPTY.IOH	Counts uclk cycles all entries in the Quickpath Home Logic IOH is busy.	
22H	02H	UNC_QHL_CYCLES_NOT_EMPTY.REMOTE	Counts uclk cycles all entries in the Quickpath Home Logic remote tracker is busy.	
22H	04H	UNC_QHL_CYCLES_NOT_EMPTY.LOCAL	Counts uclk cycles all entries in the Quickpath Home Logic local tracker is busy.	
23H	01H	UNC_QHL_OCCUPANCY.IOH	QHL IOH tracker allocate to deallocate read occupancy.	
23H	02H	UNC_QHL_OCCUPANCY.REMOTE	QHL remote tracker allocate to deallocate read occupancy.	
23H	04H	UNC_QHL_OCCUPANCY.LOCAL	QHL local tracker allocate to deallocate read occupancy.	

**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	02H	UNC_QHL_ADDRESS_CONFLICTS.2WAY	Counts number of QHL Active Address Table (AAT) entries that saw a max of 2 conflicts. The AAT is a structure that tracks requests that are in conflict. The requests themselves are in the home tracker entries. The count is reported when an AAT entry deallocates.	
24H	04H	UNC_QHL_ADDRESS_CONFLICTS.3WAY	Counts number of QHL Active Address Table (AAT) entries that saw a max of 3 conflicts. The AAT is a structure that tracks requests that are in conflict. The requests themselves are in the home tracker entries. The count is reported when an AAT entry deallocates.	
25H	01H	UNC_QHL_CONFLICT_CYCLES.IOH	Counts cycles the Quickpath Home Logic IOH Tracker contains two or more requests with an address conflict. A max of 3 requests can be in conflict.	
25H	02H	UNC_QHL_CONFLICT_CYCLES.REMOTE	Counts cycles the Quickpath Home Logic Remote Tracker contains two or more requests with an address conflict. A max of 3 requests can be in conflict.	
25H	04H	UNC_QHL_CONFLICT_CYCLES.LOCAL	Counts cycles the Quickpath Home Logic Local Tracker contains two or more requests with an address conflict. A max of 3 requests can be in conflict.	
26H	01H	UNC_QHL_TO_QMC_BYPASS	Counts number or requests to the Quickpath Memory Controller that bypass the Quickpath Home Logic. All local accesses can be bypassed. For remote requests, only read requests can be bypassed.	
27H	01H	UNC_QMC_NORMAL_FULL.READ.CH0	Uncore cycles all the entries in the DRAM channel 0 medium or low priority queue are occupied with read requests.	
27H	02H	UNC_QMC_NORMAL_FULL.READ.CH1	Uncore cycles all the entries in the DRAM channel 1 medium or low priority queue are occupied with read requests.	
27H	04H	UNC_QMC_NORMAL_FULL.READ.CH2	Uncore cycles all the entries in the DRAM channel 2 medium or low priority queue are occupied with read requests.	
27H	08H	UNC_QMC_NORMAL_FULL.WRITE.CH0	Uncore cycles all the entries in the DRAM channel 0 medium or low priority queue are occupied with write requests.	
27H	10H	UNC_QMC_NORMAL_FULL.WRITE.CH1	Counts cycles all the entries in the DRAM channel 1 medium or low priority queue are occupied with write requests.	
27H	20H	UNC_QMC_NORMAL_FULL.WRITE.CH2	Uncore cycles all the entries in the DRAM channel 2 medium or low priority queue are occupied with write requests.	
28H	01H	UNC_QMC_ISOC_FULL.READ.CH0	Counts cycles all the entries in the DRAM channel 0 high priority queue are occupied with isochronous read requests.	
28H	02H	UNC_QMC_ISOC_FULL.READ.CH1	Counts cycles all the entries in the DRAM channel 1 high priority queue are occupied with isochronous read requests.	

**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
28H	04H	UNC_QMC_ISSOC_FULLL.READ.CH2	Counts cycles all the entries in the DRAM channel 2 high priority queue are occupied with isochronous read requests.	
28H	08H	UNC_QMC_ISSOC_FULLL.WRITE.CH0	Counts cycles all the entries in the DRAM channel 0 high priority queue are occupied with isochronous write requests.	
28H	10H	UNC_QMC_ISSOC_FULLL.WRITE.CH1	Counts cycles all the entries in the DRAM channel 1 high priority queue are occupied with isochronous write requests.	
28H	20H	UNC_QMC_ISSOC_FULLL.WRITE.CH2	Counts cycles all the entries in the DRAM channel 2 high priority queue are occupied with isochronous write requests.	
29H	01H	UNC_QMC_BUSY.READ.CH0	Counts cycles where Quickpath Memory Controller has at least 1 outstanding read request to DRAM channel 0.	
29H	02H	UNC_QMC_BUSY.READ.CH1	Counts cycles where Quickpath Memory Controller has at least 1 outstanding read request to DRAM channel 1.	
29H	04H	UNC_QMC_BUSY.READ.CH2	Counts cycles where Quickpath Memory Controller has at least 1 outstanding read request to DRAM channel 2.	
29H	08H	UNC_QMC_BUSY.WRITE.CH0	Counts cycles where Quickpath Memory Controller has at least 1 outstanding write request to DRAM channel 0.	
29H	10H	UNC_QMC_BUSY.WRITE.CH1	Counts cycles where Quickpath Memory Controller has at least 1 outstanding write request to DRAM channel 1.	
29H	20H	UNC_QMC_BUSY.WRITE.CH2	Counts cycles where Quickpath Memory Controller has at least 1 outstanding write request to DRAM channel 2.	
2AH	01H	UNC_QMC_OCCUPANCY.CH0	IMC channel 0 normal read request occupancy.	
2AH	02H	UNC_QMC_OCCUPANCY.CH1	IMC channel 1 normal read request occupancy.	
2AH	04H	UNC_QMC_OCCUPANCY.CH2	IMC channel 2 normal read request occupancy.	
2BH	01H	UNC_QMC_ISSOC_OCCUPANCY.CH0	IMC channel 0 issoc read request occupancy.	
2BH	02H	UNC_QMC_ISSOC_OCCUPANCY.CH1	IMC channel 1 issoc read request occupancy.	
2BH	04H	UNC_QMC_ISSOC_OCCUPANCY.CH2	IMC channel 2 issoc read request occupancy.	
2BH	07H	UNC_QMC_ISSOC_READS.ANY	IMC issoc read request occupancy.	
2CH	01H	UNC_QMC_NORMAL_READS.CH0	Counts the number of Quickpath Memory Controller channel 0 medium and low priority read requests. The QMC channel 0 normal read occupancy divided by this count provides the average QMC channel 0 read latency.	

**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
2CH	02H	UNC_QMC_NORMAL_READS.CH1	Counts the number of Quickpath Memory Controller channel 1 medium and low priority read requests. The QMC channel 1 normal read occupancy divided by this count provides the average QMC channel 1 read latency.	
2CH	04H	UNC_QMC_NORMAL_READS.CH2	Counts the number of Quickpath Memory Controller channel 2 medium and low priority read requests. The QMC channel 2 normal read occupancy divided by this count provides the average QMC channel 2 read latency.	
2CH	07H	UNC_QMC_NORMAL_READS.ANY	Counts the number of Quickpath Memory Controller medium and low priority read requests. The QMC normal read occupancy divided by this count provides the average QMC read latency.	
2DH	01H	UNC_QMC_HIGH_PRIORITY_READS.CH0	Counts the number of Quickpath Memory Controller channel 0 high priority isochronous read requests.	
2DH	02H	UNC_QMC_HIGH_PRIORITY_READS.CH1	Counts the number of Quickpath Memory Controller channel 1 high priority isochronous read requests.	
2DH	04H	UNC_QMC_HIGH_PRIORITY_READS.CH2	Counts the number of Quickpath Memory Controller channel 2 high priority isochronous read requests.	
2DH	07H	UNC_QMC_HIGH_PRIORITY_READS.ANY	Counts the number of Quickpath Memory Controller high priority isochronous read requests.	
2EH	01H	UNC_QMC_CRITICAL_PRIORITY_READS.CH0	Counts the number of Quickpath Memory Controller channel 0 critical priority isochronous read requests.	
2EH	02H	UNC_QMC_CRITICAL_PRIORITY_READS.CH1	Counts the number of Quickpath Memory Controller channel 1 critical priority isochronous read requests.	
2EH	04H	UNC_QMC_CRITICAL_PRIORITY_READS.CH2	Counts the number of Quickpath Memory Controller channel 2 critical priority isochronous read requests.	
2EH	07H	UNC_QMC_CRITICAL_PRIORITY_READS.ANY	Counts the number of Quickpath Memory Controller critical priority isochronous read requests.	
2FH	01H	UNC_QMC_WRITES.FULL.CH0	Counts number of full cache line writes to DRAM channel 0.	
2FH	02H	UNC_QMC_WRITES.FULL.CH1	Counts number of full cache line writes to DRAM channel 1.	
2FH	04H	UNC_QMC_WRITES.FULL.CH2	Counts number of full cache line writes to DRAM channel 2.	
2FH	07H	UNC_QMC_WRITES.FULL.ANY	Counts number of full cache line writes to DRAM.	
2FH	08H	UNC_QMC_WRITES.PARTIAL.CH0	Counts number of partial cache line writes to DRAM channel 0.	
2FH	10H	UNC_QMC_WRITES.PARTIAL.CH1	Counts number of partial cache line writes to DRAM channel 1.	
2FH	20H	UNC_QMC_WRITES.PARTIAL.CH2	Counts number of partial cache line writes to DRAM channel 2.	
2FH	38H	UNC_QMC_WRITES.PARTIAL.ANY	Counts number of partial cache line writes to DRAM.	
30H	01H	UNC_QMC_CANCEL.CH0	Counts number of DRAM channel 0 cancel requests.	
30H	02H	UNC_QMC_CANCEL.CH1	Counts number of DRAM channel 1 cancel requests.	

**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
30H	04H	UNC_QMC_CANCEL.CH2	Counts number of DRAM channel 2 cancel requests.	
30H	07H	UNC_QMC_CANCEL.ANY	Counts number of DRAM cancel requests.	
31H	01H	UNC_QMC_PRIORITY_UPDATE S.CH0	Counts number of DRAM channel 0 priority updates. A priority update occurs when an ISOC high or critical request is received by the QHL and there is a matching request with normal priority that has already been issued to the QMC. In this instance, the QHL will send a priority update to QMC to expedite the request.	
31H	02H	UNC_QMC_PRIORITY_UPDATE S.CH1	Counts number of DRAM channel 1 priority updates. A priority update occurs when an ISOC high or critical request is received by the QHL and there is a matching request with normal priority that has already been issued to the QMC. In this instance, the QHL will send a priority update to QMC to expedite the request.	
31H	04H	UNC_QMC_PRIORITY_UPDATE S.CH2	Counts number of DRAM channel 2 priority updates. A priority update occurs when an ISOC high or critical request is received by the QHL and there is a matching request with normal priority that has already been issued to the QMC. In this instance, the QHL will send a priority update to QMC to expedite the request.	
31H	07H	UNC_QMC_PRIORITY_UPDATE S.ANY	Counts number of DRAM priority updates. A priority update occurs when an ISOC high or critical request is received by the QHL and there is a matching request with normal priority that has already been issued to the QMC. In this instance, the QHL will send a priority update to QMC to expedite the request.	
33H	04H	UNC_QHL_FRC_ACK_CNFLTS.L OCAL	Counts number of Force Acknowledge Conflict messages sent by the Quickpath Home Logic to the local home.	
40H	01H	UNC_QPI_TX_STALLED_SINGL E_FLIT.HOME.LINK_0	Counts cycles the Quickpath outbound link 0 HOME virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	02H	UNC_QPI_TX_STALLED_SINGL E_FLIT.SNOOP.LINK_0	Counts cycles the Quickpath outbound link 0 SNOOP virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	04H	UNC_QPI_TX_STALLED_SINGL E_FLIT.NDR.LINK_0	Counts cycles the Quickpath outbound link 0 non-data response virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	08H	UNC_QPI_TX_STALLED_SINGL E_FLIT.HOME.LINK_1	Counts cycles the Quickpath outbound link 1 HOME virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	

**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
40H	10H	UNC_QPI_TX_STALLED_SINGLE_FLIT.SNOOP.LINK_1	Counts cycles the Quickpath outbound link 1 SNOOP virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	20H	UNC_QPI_TX_STALLED_SINGLE_FLIT.NDR.LINK_1	Counts cycles the Quickpath outbound link 1 non-data response virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	07H	UNC_QPI_TX_STALLED_SINGLE_FLIT.LINK_0	Counts cycles the Quickpath outbound link 0 virtual channels are stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	38H	UNC_QPI_TX_STALLED_SINGLE_FLIT.LINK_1	Counts cycles the Quickpath outbound link 1 virtual channels are stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	01H	UNC_QPI_TX_STALLED_MULTIFLIT.DRS.LINK_0	Counts cycles the Quickpath outbound link 0 Data Response virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	02H	UNC_QPI_TX_STALLED_MULTIFLIT.NCB.LINK_0	Counts cycles the Quickpath outbound link 0 Non-Coherent Bypass virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	04H	UNC_QPI_TX_STALLED_MULTIFLIT.NCS.LINK_0	Counts cycles the Quickpath outbound link 0 Non-Coherent Standard virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	08H	UNC_QPI_TX_STALLED_MULTIFLIT.DRS.LINK_1	Counts cycles the Quickpath outbound link 1 Data Response virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	10H	UNC_QPI_TX_STALLED_MULTIFLIT.NCB.LINK_1	Counts cycles the Quickpath outbound link 1 Non-Coherent Bypass virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	



**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
41H	20H	UNC_QPI_TX_STALLED_MULTIFLIT.NCS.LINK_1	Counts cycles the Quickpath outbound link 1 Non-Coherent Standard virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	07H	UNC_QPI_TX_STALLED_MULTIFLIT.LINK_0	Counts cycles the Quickpath outbound link 0 virtual channels are stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	38H	UNC_QPI_TX_STALLED_MULTIFLIT.LINK_1	Counts cycles the Quickpath outbound link 1 virtual channels are stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
42H	02H	UNC_QPI_TX_HEADER.BUSY.LINK_0	Number of cycles that the header buffer in the Quickpath Interface outbound link 0 is busy.	
42H	08H	UNC_QPI_TX_HEADER.BUSY.LINK_1	Number of cycles that the header buffer in the Quickpath Interface outbound link 1 is busy.	
43H	01H	UNC_QPI_RX_NO_PPT_CREDIT.STALLS.LINK_0	Number of cycles that snoop packets incoming to the Quickpath Interface link 0 are stalled and not sent to the GQ because the GQ Peer Probe Tracker (PPT) does not have any available entries.	
43H	02H	UNC_QPI_RX_NO_PPT_CREDIT.STALLS.LINK_1	Number of cycles that snoop packets incoming to the Quickpath Interface link 1 are stalled and not sent to the GQ because the GQ Peer Probe Tracker (PPT) does not have any available entries.	
60H	01H	UNC_DRAM_OPEN.CH0	Counts number of DRAM Channel 0 open commands issued either for read or write. To read or write data, the referenced DRAM page must first be opened.	
60H	02H	UNC_DRAM_OPEN.CH1	Counts number of DRAM Channel 1 open commands issued either for read or write. To read or write data, the referenced DRAM page must first be opened.	
60H	04H	UNC_DRAM_OPEN.CH2	Counts number of DRAM Channel 2 open commands issued either for read or write. To read or write data, the referenced DRAM page must first be opened.	
61H	01H	UNC_DRAM_PAGE_CLOSE.CH0	DRAM channel 0 command issued to CLOSE a page due to page idle timer expiration. Closing a page is done by issuing a precharge.	
61H	02H	UNC_DRAM_PAGE_CLOSE.CH1	DRAM channel 1 command issued to CLOSE a page due to page idle timer expiration. Closing a page is done by issuing a precharge.	
61H	04H	UNC_DRAM_PAGE_CLOSE.CH2	DRAM channel 2 command issued to CLOSE a page due to page idle timer expiration. Closing a page is done by issuing a precharge.	



**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
62H	01H	UNC_DRAM_PAGE_MISS.CH0	Counts the number of precharges (PRE) that were issued to DRAM channel 0 because there was a page miss. A page miss refers to a situation in which a page is currently open and another page from the same bank needs to be opened. The new page experiences a page miss. Closing of the old page is done by issuing a precharge.	
62H	02H	UNC_DRAM_PAGE_MISS.CH1	Counts the number of precharges (PRE) that were issued to DRAM channel 1 because there was a page miss. A page miss refers to a situation in which a page is currently open and another page from the same bank needs to be opened. The new page experiences a page miss. Closing of the old page is done by issuing a precharge.	
62H	04H	UNC_DRAM_PAGE_MISS.CH2	Counts the number of precharges (PRE) that were issued to DRAM channel 2 because there was a page miss. A page miss refers to a situation in which a page is currently open and another page from the same bank needs to be opened. The new page experiences a page miss. Closing of the old page is done by issuing a precharge.	
63H	01H	UNC_DRAM_READ_CAS.CH0	Counts the number of times a read CAS command was issued on DRAM channel 0.	
63H	02H	UNC_DRAM_READ_CAS.AUTO PRE_CH0	Counts the number of times a read CAS command was issued on DRAM channel 0 where the command issued used the auto-precharge (auto page close) mode.	
63H	04H	UNC_DRAM_READ_CAS.CH1	Counts the number of times a read CAS command was issued on DRAM channel 1.	
63H	08H	UNC_DRAM_READ_CAS.AUTO PRE_CH1	Counts the number of times a read CAS command was issued on DRAM channel 1 where the command issued used the auto-precharge (auto page close) mode.	
63H	10H	UNC_DRAM_READ_CAS.CH2	Counts the number of times a read CAS command was issued on DRAM channel 2.	
63H	20H	UNC_DRAM_READ_CAS.AUTO PRE_CH2	Counts the number of times a read CAS command was issued on DRAM channel 2 where the command issued used the auto-precharge (auto page close) mode.	
64H	01H	UNC_DRAM_WRITE_CAS.CH0	Counts the number of times a write CAS command was issued on DRAM channel 0.	
64H	02H	UNC_DRAM_WRITE_CAS.AUTO PRE_CH0	Counts the number of times a write CAS command was issued on DRAM channel 0 where the command issued used the auto-precharge (auto page close) mode.	
64H	04H	UNC_DRAM_WRITE_CAS.CH1	Counts the number of times a write CAS command was issued on DRAM channel 1.	
64H	08H	UNC_DRAM_WRITE_CAS.AUTO PRE_CH1	Counts the number of times a write CAS command was issued on DRAM channel 1 where the command issued used the auto-precharge (auto page close) mode.	
64H	10H	UNC_DRAM_WRITE_CAS.CH2	Counts the number of times a write CAS command was issued on DRAM channel 2.	

**Table 19-20. Performance Events In the Processor Uncore for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
64H	20H	UNC_DRAM_WRITE_CAS.AUTO PRE_CH2	Counts the number of times a write CAS command was issued on DRAM channel 2 where the command issued used the auto-precharge (auto page close) mode.	
65H	01H	UNC_DRAM_REFRESH.CH0	Counts number of DRAM channel 0 refresh commands. DRAM loses data content over time. In order to keep correct data content, the data values have to be refreshed periodically.	
65H	02H	UNC_DRAM_REFRESH.CH1	Counts number of DRAM channel 1 refresh commands. DRAM loses data content over time. In order to keep correct data content, the data values have to be refreshed periodically.	
65H	04H	UNC_DRAM_REFRESH.CH2	Counts number of DRAM channel 2 refresh commands. DRAM loses data content over time. In order to keep correct data content, the data values have to be refreshed periodically.	
66H	01H	UNC_DRAM_PRE_ALL.CH0	Counts number of DRAM Channel 0 precharge-all (PREALL) commands that close all open pages in a rank. PREALL is issued when the DRAM needs to be refreshed or needs to go into a power down mode.	
66H	02H	UNC_DRAM_PRE_ALL.CH1	Counts number of DRAM Channel 1 precharge-all (PREALL) commands that close all open pages in a rank. PREALL is issued when the DRAM needs to be refreshed or needs to go into a power down mode.	
66H	04H	UNC_DRAM_PRE_ALL.CH2	Counts number of DRAM Channel 2 precharge-all (PREALL) commands that close all open pages in a rank. PREALL is issued when the DRAM needs to be refreshed or needs to go into a power down mode.	

Intel Xeon processors with CPUID signature of DisplayFamily\_DisplayModel 06\_2EH have a distinct uncore subsystem that is significantly different from the uncore found in processors with CPUID signature 06\_1AH, 06\_1EH, and 06\_1FH. Model-specific performance monitoring events for its uncore will be available in future documentation.

## 19.10 PERFORMANCE MONITORING EVENTS FOR PROCESSORS BASED ON INTEL® MICROARCHITECTURE CODE NAME WESTMERE

Intel 64 processors based on Intel® microarchitecture code name Westmere support the architectural and model-specific performance monitoring events listed in Table 19-1 and Table 19-21. Table 19-21 applies to processors with CPUID signature of DisplayFamily\_DisplayModel encoding with the following values: 06\_25H, 06\_2CH. In addition, these processors (CPUID signature of DisplayFamily\_DisplayModel 06\_25H, 06\_2CH) also support the following model-specific, product-specific uncore performance monitoring events listed in Table 19-22. Fixed counters support the architecture events defined in Table 19-2.

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	LOAD_BLOCK.OVERLAP_STORE	Loads that partially overlap an earlier store.	
04H	07H	SB_DRAIN.ANY	All Store buffer stall cycles.	
05H	02H	MISALIGN_MEMORY.STORE	All store referenced with misaligned address.	
06H	04H	STORE_BLOCKS.AT_RET	Counts number of loads delayed with at-Retirement block code. The following loads need to be executed at retirement and wait for all senior stores on the same thread to be drained: load splitting across 4K boundary (page split), load accessing uncacheable (UC or WC) memory, load lock, and load with page table in UC or WC memory region.	
06H	08H	STORE_BLOCKS.L1D_BLOCK	Cacheable loads delayed with L1D block code.	
07H	01H	PARTIAL_ADDRESS_ALIAS	Counts false dependency due to partial address aliasing.	
08H	01H	DTLB_LOAD_MISSES.ANY	Counts all load misses that cause a page walk.	
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED	Counts number of completed page walks due to load miss in the STLB.	
08H	04H	DTLB_LOAD_MISSES.WALK_CYCLES	Cycles PMH is busy with a page walk due to a load miss in the STLB.	
08H	10H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits.	
08H	20H	DTLB_LOAD_MISSES.PDE_MISSES	Number of DTLB cache load misses where the low part of the linear to physical address translation was missed.	
0BH	01H	MEM_INST_RETIRED.LOADS	Counts the number of instructions with an architecturally-visible load retired on the architected path.	
0BH	02H	MEM_INST_RETIRED.STORES	Counts the number of instructions with an architecturally-visible store retired on the architected path.	
0BH	10H	MEM_INST_RETIRED.LATENCY_ABOVE_THRESHOLD	Counts the number of instructions exceeding the latency specified with <code>ld_lat</code> facility.	In conjunction with <code>ld_lat</code> facility.
0CH	01H	MEM_STORE_RETIRED.DTLB_MISS	The event counts the number of retired stores that missed the DTLB. The DTLB miss is not counted if the store operation causes a fault. Does not counter prefetches. Counts both primary and secondary misses to the TLB.	
0EH	01H	UOPS_ISSUED.ANY	Counts the number of Uops issued by the Register Allocation Table to the Reservation Station, i.e. the UOPs issued from the front end to the back end.	
0EH	01H	UOPS_ISSUED.STALLED_CYCLES	Counts the number of cycles no uops issued by the Register Allocation Table to the Reservation Station, i.e. the UOPs issued from the front end to the back end.	Set "invert=1, cmask = 1".
0EH	02H	UOPS_ISSUED.FUSED	Counts the number of fused Uops that were issued from the Register Allocation Table to the Reservation Station.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
0FH	01H	MEM_UNCORE_RETIRE.UNKNOWN_SOURCE	Load instructions retired with unknown LLC miss (Precise Event).	Applicable to one and two sockets.
0FH	02H	MEM_UNCORE_RETIRE.OTHER_CORE_L2_HIT	Load instructions retired that HIT modified data in sibling core (Precise Event).	Applicable to one and two sockets.
0FH	04H	MEM_UNCORE_RETIRE.REMOTE_HITM	Load instructions retired that HIT modified data in remote socket (Precise Event).	Applicable to two sockets only.
0FH	08H	MEM_UNCORE_RETIRE.LOCAL_DRAM_AND_REMOTE_CACHE_HIT	Load instructions retired local dram and remote cache HIT data sources (Precise Event).	Applicable to one and two sockets.
0FH	10H	MEM_UNCORE_RETIRE.REMOTE_DRAM	Load instructions retired remote DRAM and remote home-remote cache HITM (Precise Event).	Applicable to two sockets only.
0FH	20H	MEM_UNCORE_RETIRE.OTHER_LLC_MISS	Load instructions retired other LLC miss (Precise Event).	Applicable to two sockets only.
0FH	80H	MEM_UNCORE_RETIRE.UNCACHEABLE	Load instructions retired I/O (Precise Event).	Applicable to one and two sockets.
10H	01H	FP_COMP_OPS_EXE.X87	Counts the number of FP Computational Uops Executed. The number of FADD, FSUB, FCOM, FMULs, integer MULs and IMULs, FDIVs, FPREMs, FSQRTS, integer DIVs, and IDIVs. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	
10H	02H	FP_COMP_OPS_EXE.MMX	Counts number of MMX Uops executed.	
10H	04H	FP_COMP_OPS_EXE.SSE_FP	Counts number of SSE and SSE2 FP uops executed.	
10H	08H	FP_COMP_OPS_EXE.SSE2_INTEGER	Counts number of SSE2 integer uops executed.	
10H	10H	FP_COMP_OPS_EXE.SSE_FP_PACKED	Counts number of SSE FP packed uops executed.	
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR	Counts number of SSE FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_SINGLE_PRECISION	Counts number of SSE* FP single precision uops executed.	
10H	80H	FP_COMP_OPS_EXE.SSE_DOUBLE_PRECISION	Counts number of SSE* FP double precision uops executed.	
12H	01H	SIMD_INT_128.PACKED_MPY	Counts number of 128 bit SIMD integer multiply operations.	
12H	02H	SIMD_INT_128.PACKED_SHIFT	Counts number of 128 bit SIMD integer shift operations.	
12H	04H	SIMD_INT_128.PACK	Counts number of 128 bit SIMD integer pack operations.	
12H	08H	SIMD_INT_128.UNPACK	Counts number of 128 bit SIMD integer unpack operations.	
12H	10H	SIMD_INT_128.PACKED_LOGICAL	Counts number of 128 bit SIMD integer logical operations.	
12H	20H	SIMD_INT_128.PACKED_ARITH	Counts number of 128 bit SIMD integer arithmetic operations.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
12H	40H	SIMD_INT_128.SHUFFLE_MOVE	Counts number of 128 bit SIMD integer shuffle and move operations.	
13H	01H	LOAD_DISPATCH.RS	Counts number of loads dispatched from the Reservation Station that bypass the Memory Order Buffer.	
13H	02H	LOAD_DISPATCH.RS_DELAYED	Counts the number of delayed RS dispatches at the stage latch. If an RS dispatch cannot bypass to LB, it has another chance to dispatch from the one-cycle delayed staging latch before it is written into the LB.	
13H	04H	LOAD_DISPATCH.MOB	Counts the number of loads dispatched from the Reservation Station to the Memory Order Buffer.	
13H	07H	LOAD_DISPATCH.ANY	Counts all loads dispatched from the Reservation Station.	
14H	01H	ARITH.CYCLES_DIV_BUSY	Counts the number of cycles the divider is busy executing divide or square root operations. The divide can be integer, X87 or Streaming SIMD Extensions (SSE). The square root operation can be either X87 or SSE. Set 'edge =1, invert=1, cmask=1' to count the number of divides.	Count may be incorrect When SMT is on.
14H	02H	ARITH.MUL	Counts the number of multiply operations executed. This includes integer as well as floating point multiply operations but excludes DPPS mul and MPSAD.	Count may be incorrect When SMT is on.
17H	01H	INST_QUEUE_WRITES	Counts the number of instructions written into the instruction queue every cycle.	
18H	01H	INST_DECODED.DECO	Counts number of instructions that require decoder 0 to be decoded. Usually, this means that the instruction maps to more than 1 uop.	
19H	01H	TWO_UOP_INSTS_DECODED	An instruction that generates two uops was decoded.	
1EH	01H	INST_QUEUE_WRITE_CYCLES	This event counts the number of cycles during which instructions are written to the instruction queue. Dividing this counter by the number of instructions written to the instruction queue (INST_QUEUE_WRITES) yields the average number of instructions decoded each cycle. If this number is less than four and the pipe stalls, this indicates that the decoder is failing to decode enough instructions per cycle to sustain the 4-wide pipeline.	If SSE* instructions that are 6 bytes or longer arrive one after another, then front end throughput may limit execution speed.
20H	01H	LSD_OVERFLOW	Number of loops that cannot stream from the instruction queue.	
24H	01H	L2_RQSTS.LD_HIT	Counts number of loads that hit the L2 cache. L2 loads include both L1D demand misses as well as L1D prefetches. L2 loads can be rejected for various reasons. Only non rejected loads are counted.	
24H	02H	L2_RQSTS.LD_MISS	Counts the number of loads that miss the L2 cache. L2 loads include both L1D demand misses as well as L1D prefetches.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	03H	L2_RQSTS.LOADS	Counts all L2 load requests. L2 loads include both L1D demand misses as well as L1D prefetches.	
24H	04H	L2_RQSTS.RFO_HIT	Counts the number of store RFO requests that hit the L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches. Count includes WC memory requests, where the data is not fetched but the permission to write the line is required.	
24H	08H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches.	
24H	0CH	L2_RQSTS.RFOS	Counts all L2 store RFO requests. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches.	
24H	10H	L2_RQSTS.IFETCH_HIT	Counts number of instruction fetches that hit the L2 cache. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	20H	L2_RQSTS.IFETCH_MISS	Counts number of instruction fetches that miss the L2 cache. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	30H	L2_RQSTS.IFETCHES	Counts all instruction fetches. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	40H	L2_RQSTS.PREFETCH_HIT	Counts L2 prefetch hits for both code and data.	
24H	80H	L2_RQSTS.PREFETCH_MISS	Counts L2 prefetch misses for both code and data.	
24H	C0H	L2_RQSTS.PREFETCHES	Counts all L2 prefetches for both code and data.	
24H	AAH	L2_RQSTS.MISS	Counts all L2 misses for both code and data.	
24H	FFH	L2_RQSTS.REFERENCES	Counts all L2 requests for both code and data.	
26H	01H	L2_DATA_RQSTS.DEMAND.I_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the I (invalid) state, i.e., a cache miss. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	02H	L2_DATA_RQSTS.DEMAND.S_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the S (shared) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	04H	L2_DATA_RQSTS.DEMAND.E_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the E (exclusive) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	08H	L2_DATA_RQSTS.DEMAND.M_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the M (modified) state. L2 demand loads are both L1D demand misses and L1D prefetches.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
26H	0FH	L2_DATA_RQSTS.DEMAND.MESI	Counts all L2 data demand requests. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	10H	L2_DATA_RQSTS.PREFETCH.I_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the I (invalid) state, i.e., a cache miss.	
26H	20H	L2_DATA_RQSTS.PREFETCH.S_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the S (shared) state. A prefetch RFO will miss on an S state line, while a prefetch read will hit on an S state line.	
26H	40H	L2_DATA_RQSTS.PREFETCH.E_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the E (exclusive) state.	
26H	80H	L2_DATA_RQSTS.PREFETCH.M_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the M (modified) state.	
26H	F0H	L2_DATA_RQSTS.PREFETCH.MESI	Counts all L2 prefetch requests.	
26H	FFH	L2_DATA_RQSTS.ANY	Counts all L2 data requests.	
27H	01H	L2_WRITE.RFO.I_STATE	Counts number of L2 demand store RFO requests where the cache line to be loaded is in the I (invalid) state, i.e., a cache miss. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	02H	L2_WRITE.RFO.S_STATE	Counts number of L2 store RFO requests where the cache line to be loaded is in the S (shared) state. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	08H	L2_WRITE.RFO.M_STATE	Counts number of L2 store RFO requests where the cache line to be loaded is in the M (modified) state. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	0EH	L2_WRITE.RFO.HIT	Counts number of L2 store RFO requests where the cache line to be loaded is in either the S, E or M states. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	0FH	L2_WRITE.RFO.MESI	Counts all L2 store RFO requests. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request.
27H	10H	L2_WRITE.LOCK.I_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the I (invalid) state, i.e., a cache miss.	
27H	20H	L2_WRITE.LOCK.S_STATE	Counts number of L2 lock RFO requests where the cache line to be loaded is in the S (shared) state.	
27H	40H	L2_WRITE.LOCK.E_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the E (exclusive) state.	
27H	80H	L2_WRITE.LOCK.M_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the M (modified) state.	
27H	E0H	L2_WRITE.LOCK.HIT	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in either the S, E, or M state.	



**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
27H	F0H	L2_WRITE.LOCK.MESI	Counts all L2 demand lock RFO requests.	
28H	01H	L1D_WB_L2.I_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the I (invalid) state, i.e., a cache miss.	
28H	02H	L1D_WB_L2.S_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the S state.	
28H	04H	L1D_WB_L2.E_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the E (exclusive) state.	
28H	08H	L1D_WB_L2.M_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the M (modified) state.	
28H	0FH	L1D_WB_L2.MESI	Counts all L1 writebacks to the L2 .	
2EH	41H	L3_LAT_CACHE.MISS	Counts uncore Last Level Cache misses. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	See Table 19-1.
2EH	4FH	L3_LAT_CACHE.REFERENCE	Counts uncore Last Level Cache references. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	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_UNHALTED.REF_P	Increments at the frequency of TSC when not halted.	See Table 19-1.
49H	01H	DTLB_MISSES.ANY	Counts the number of misses in the STLB which causes a page walk.	
49H	02H	DTLB_MISSES.WALK_COMPLETED	Counts number of misses in the STLB which resulted in a completed page walk.	
49H	04H	DTLB_MISSES.WALK_CYCLES	Counts cycles of page walk due to misses in the STLB.	
49H	10H	DTLB_MISSES.STLB_HIT	Counts the number of DTLB first level misses that hit in the second level TLB. This event is only relevant if the core contains multiple DTLB levels.	
49H	20H	DTLB_MISSES.PDE_MISS	Number of DTLB misses caused by low part of address, includes references to 2M pages because 2M pages do not use the PDE.	
49H	80H	DTLB_MISSES.LARGE_WALK_COMPLETED	Counts number of completed large page walks due to misses in the STLB.	
4CH	01H	LOAD_HIT_PRE	Counts load operations sent to the L1 data cache while a previous SSE prefetch instruction to the same cache line has started prefetching but has not yet finished.	Counter 0, 1 only.



**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
4EH	01H	L1D_PREFETCH.REQUESTS	Counts number of hardware prefetch requests dispatched out of the prefetch FIFO.	Counter 0, 1 only.
4EH	02H	L1D_PREFETCH.MISS	Counts number of hardware prefetch requests that miss the L1D. There are two prefetchers in the L1D. A streamer, which predicts lines sequentially after this one should be fetched, and the IP prefetcher that remembers access patterns for the current instruction. The streamer prefetcher stops on an L1D hit, while the IP prefetcher does not.	Counter 0, 1 only.
4EH	04H	L1D_PREFETCH.TRIGGERS	Counts number of prefetch requests triggered by the Finite State Machine and pushed into the prefetch FIFO. Some of the prefetch requests are dropped due to overwrites or competition between the IP index prefetcher and streamer prefetcher. The prefetch FIFO contains 4 entries.	Counter 0, 1 only.
4FH	10H	EPT.WALK_CYCLES	Counts Extended Page walk cycles.	
51H	01H	L1D.REPL	Counts the number of lines brought into the L1 data cache.	Counter 0, 1 only.
51H	02H	L1D.M_REPL	Counts the number of modified lines brought into the L1 data cache.	Counter 0, 1 only.
51H	04H	L1D.M_EVICT	Counts the number of modified lines evicted from the L1 data cache due to replacement.	Counter 0, 1 only.
51H	08H	L1D.M_SNOOP_EVICT	Counts the number of modified lines evicted from the L1 data cache due to snoop HITM intervention.	Counter 0, 1 only.
52H	01H	L1D_CACHE_PREFETCH_LOCK_FB_HIT	Counts the number of cacheable load lock speculated instructions accepted into the fill buffer.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_DATA	Counts weighted cycles of offcore demand data read requests. Does not include L2 prefetch requests.	Counter 0.
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_CODE	Counts weighted cycles of offcore demand code read requests. Does not include L2 prefetch requests.	Counter 0.
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND.RFO	Counts weighted cycles of offcore demand RFO requests. Does not include L2 prefetch requests.	Counter 0.
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ANY.READ	Counts weighted cycles of offcore read requests of any kind. Include L2 prefetch requests.	Counter 0.
63H	01H	CACHE_LOCK_CYCLES.L1D_L2	Cycle count during which the L1D and L2 are locked. A lock is asserted when there is a locked memory access, due to uncacheable memory, a locked operation that spans two cache lines, or a page walk from an uncacheable page table. This event does not cause locks, it merely detects them.	Counter 0, 1 only. L1D and L2 locks have a very high performance penalty and it is highly recommended to avoid such accesses.
63H	02H	CACHE_LOCK_CYCLES.L1D	Counts the number of cycles that cacheline in the L1 data cache unit is locked.	Counter 0, 1 only.
6CH	01H	IO_TRANSACTIONS	Counts the number of completed I/O transactions.	
80H	01H	L1I.HITS	Counts all instruction fetches that hit the L1 instruction cache.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
80H	02H	L1I.MISSES	Counts all instruction fetches that miss the L1I cache. This includes instruction cache misses, streaming buffer misses, victim cache misses and uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.	
80H	03H	L1I.READS	Counts all instruction fetches, including uncacheable fetches that bypass the L1I.	
80H	04H	L1I.CYCLES_STALLED	Cycle counts for which an instruction fetch stalls due to a L1I cache miss, ITLB miss or ITLB fault.	
82H	01H	LARGE_ITLB.HIT	Counts number of large ITLB hits.	
85H	01H	ITLB_MISSES.ANY	Counts the number of misses in all levels of the ITLB which causes a page walk.	
85H	02H	ITLB_MISSES.WALK_COMPLETED	Counts number of misses in all levels of the ITLB which resulted in a completed page walk.	
85H	04H	ITLB_MISSES.WALK_CYCLES	Counts ITLB miss page walk cycles.	
85H	10H	ITLB_MISSES.STLB_HIT	Counts number of ITLB first level miss but second level hits.	
85H	80H	ITLB_MISSES.LARGE_WALK_COMPLETED	Counts number of completed large page walks due to misses in the STLB.	
87H	01H	ILD_STALL.LCP	Cycles Instruction Length Decoder stalls due to length changing prefixes: 66, 67 or REX.W (for Intel 64) instructions which change the length of the decoded instruction.	
87H	02H	ILD_STALL.MRU	Instruction Length Decoder stall cycles due to Branch Prediction Unit (PBU) Most Recently Used (MRU) bypass.	
87H	04H	ILD_STALL.IQ_FULL	Stall cycles due to a full instruction queue.	
87H	08H	ILD_STALL.REGEN	Counts the number of regen stalls.	
87H	0FH	ILD_STALL.ANY	Counts any cycles the Instruction Length Decoder is stalled.	
88H	01H	BR_INST_EXEC.COND	Counts the number of conditional near branch instructions executed, but not necessarily retired.	
88H	02H	BR_INST_EXEC.DIRECT	Counts all unconditional near branch instructions excluding calls and indirect branches.	
88H	04H	BR_INST_EXEC.INDIRECT_NON_CALL	Counts the number of executed indirect near branch instructions that are not calls.	
88H	07H	BR_INST_EXEC.NON_CALLS	Counts all non-call near branch instructions executed, but not necessarily retired.	
88H	08H	BR_INST_EXEC.RETURN_NEAR	Counts indirect near branches that have a return mnemonic.	
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Counts unconditional near call branch instructions, excluding non-call branch, executed.	
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Counts indirect near calls, including both register and memory indirect, executed.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
88H	30H	BR_INST_EXEC.NEAR_CALLS	Counts all near call branches executed, but not necessarily retired.	
88H	40H	BR_INST_EXEC.TAKEN	Counts taken near branches executed, but not necessarily retired.	
88H	7FH	BR_INST_EXEC.ANY	Counts all near executed branches (not necessarily retired). This includes only instructions and not micro-op branches. Frequent branching is not necessarily a major performance issue. However frequent branch mispredictions may be a problem.	
89H	01H	BR_MISP_EXEC.COND	Counts the number of mispredicted conditional near branch instructions executed, but not necessarily retired.	
89H	02H	BR_MISP_EXEC.DIRECT	Counts mispredicted macro unconditional near branch instructions, excluding calls and indirect branches (should always be 0).	
89H	04H	BR_MISP_EXEC.INDIRECT_NON_CALL	Counts the number of executed mispredicted indirect near branch instructions that are not calls.	
89H	07H	BR_MISP_EXEC.NON_CALLS	Counts mispredicted non-call near branches executed, but not necessarily retired.	
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Counts mispredicted indirect branches that have a rear return mnemonic.	
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Counts mispredicted non-indirect near calls executed, (should always be 0).	
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Counts mispredicted indirect near calls executed, including both register and memory indirect.	
89H	30H	BR_MISP_EXEC.NEAR_CALLS	Counts all mispredicted near call branches executed, but not necessarily retired.	
89H	40H	BR_MISP_EXEC.TAKEN	Counts executed mispredicted near branches that are taken, but not necessarily retired.	
89H	7FH	BR_MISP_EXEC.ANY	Counts the number of mispredicted near branch instructions that were executed, but not necessarily retired.	
A2H	01H	RESOURCE_STALLS.ANY	Counts the number of Allocator resource related stalls. Includes register renaming buffer entries, memory buffer entries. In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, such as if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	Does not include stalls due to SuperQ (off core) queue full, too many cache misses, etc.
A2H	02H	RESOURCE_STALLS.LOAD	Counts the cycles of stall due to lack of load buffer for load operation.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A2H	04H	RESOURCE_STALLS.RS_FULL	This event counts the number of cycles when the number of instructions in the pipeline waiting for execution reaches the limit the processor can handle. A high count of this event indicates that there are long latency operations in the pipe (possibly load and store operations that miss the L2 cache, or instructions dependent upon instructions further down the pipeline that have yet to retire.	When RS is full, new instructions cannot enter the reservation station and start execution.
A2H	08H	RESOURCE_STALLS.STORE	This event counts the number of cycles that a resource related stall will occur due to the number of store instructions reaching the limit of the pipeline, (i.e. all store buffers are used). The stall ends when a store instruction commits its data to the cache or memory.	
A2H	10H	RESOURCE_STALLS.ROB_FULL	Counts the cycles of stall due to re-order buffer full.	
A2H	20H	RESOURCE_STALLS.FPCW	Counts the number of cycles while execution was stalled due to writing the floating-point unit (FPU) control word.	
A2H	40H	RESOURCE_STALLS.MXCSR	Stalls due to the MXCSR register rename occurring to close to a previous MXCSR rename. The MXCSR provides control and status for the MMX registers.	
A2H	80H	RESOURCE_STALLS.OTHER	Counts the number of cycles while execution was stalled due to other resource issues.	
A6H	01H	MACRO_INSTS.FUSIONS_DECODED	Counts the number of instructions decoded that are macro-fused but not necessarily executed or retired.	
A7H	01H	BACLEAR_FORCE_IQ	Counts number of times a BACLEAR was forced by the Instruction Queue. The IQ is also responsible for providing conditional branch prediction direction based on a static scheme and dynamic data provided by the L2 Branch Prediction Unit. If the conditional branch target is not found in the Target Array and the IQ predicts that the branch is taken, then the IQ will force the Branch Address Calculator to issue a BACLEAR. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline.	
A8H	01H	LSD.UOPS	Counts the number of micro-ops delivered by loop stream detector.	Use cmask=1 and invert to count cycles.
AEH	01H	ITLB_FLUSH	Counts the number of ITLB flushes.	
BOH	01H	OFFCORE_REQUESTS.DEMAND.READ_DATA	Counts number of offcore demand data read requests. Does not count L2 prefetch requests.	
BOH	02H	OFFCORE_REQUESTS.DEMAND.READ_CODE	Counts number of offcore demand code read requests. Does not count L2 prefetch requests.	
BOH	04H	OFFCORE_REQUESTS.DEMAND.RFO	Counts number of offcore demand RFO requests. Does not count L2 prefetch requests.	
BOH	08H	OFFCORE_REQUESTS.ANY.READ	Counts number of offcore read requests. Includes L2 prefetch requests.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B0H	10H	OFFCORE_REQUESTS.ANY.RFO	Counts number of offcore RFO requests. Includes L2 prefetch requests.	
B0H	40H	OFFCORE_REQUESTS.L1D_WRITEBACK	Counts number of L1D writebacks to the uncore.	
B0H	80H	OFFCORE_REQUESTS.ANY	Counts all offcore requests.	
B1H	01H	UOPS_EXECUTED.PORT0	Counts number of uops executed that were issued on port 0. Port 0 handles integer arithmetic, SIMD and FP add uops.	
B1H	02H	UOPS_EXECUTED.PORT1	Counts number of uops executed that were issued on port 1. Port 1 handles integer arithmetic, SIMD, integer shift, FP multiply and FP divide uops.	
B1H	04H	UOPS_EXECUTED.PORT2_CORE	Counts number of uops executed that were issued on port 2. Port 2 handles the load uops. This is a core count only and cannot be collected per thread.	
B1H	08H	UOPS_EXECUTED.PORT3_CORE	Counts number of uops executed that were issued on port 3. Port 3 handles store uops. This is a core count only and cannot be collected per thread.	
B1H	10H	UOPS_EXECUTED.PORT4_CORE	Counts number of uops executed that where issued on port 4. Port 4 handles the value to be stored for the store uops issued on port 3. This is a core count only and cannot be collected per thread.	
B1H	1FH	UOPS_EXECUTED.CORE_ACTIVE_CYCLES_NO_PORT5	Counts number of cycles there are one or more uops being executed and were issued on ports 0-4. This is a core count only and cannot be collected per thread.	
B1H	20H	UOPS_EXECUTED.PORT5	Counts number of uops executed that where issued on port 5.	
B1H	3FH	UOPS_EXECUTED.CORE_ACTIVE_CYCLES	Counts number of cycles there are one or more uops being executed on any ports. This is a core count only and cannot be collected per thread.	
B1H	40H	UOPS_EXECUTED.PORT015	Counts number of uops executed that where issued on port 0, 1, or 5.	Use cmask=1, invert=1 to count stall cycles.
B1H	80H	UOPS_EXECUTED.PORT234	Counts number of uops executed that where issued on port 2, 3, or 4.	
B2H	01H	OFFCORE_REQUESTS_SQ_FULL	Counts number of cycles the SQ is full to handle off-core requests.	
B3H	01H	SNOOPQ_REQUESTS_OUTSTANDING.DATA	Counts weighted cycles of snoopq requests for data. Counter 0 only.	Use cmask=1 to count cycles not empty.
B3H	02H	SNOOPQ_REQUESTS_OUTSTANDING.INVALIDATE	Counts weighted cycles of snoopq invalidate requests. Counter 0 only.	Use cmask=1 to count cycles not empty.
B3H	04H	SNOOPQ_REQUESTS_OUTSTANDING.CODE	Counts weighted cycles of snoopq requests for code. Counter 0 only.	Use cmask=1 to count cycles not empty.
B4H	01H	SNOOPQ_REQUESTS.CODE	Counts the number of snoop code requests.	
B4H	02H	SNOOPQ_REQUESTS.DATA	Counts the number of snoop data requests.	
B4H	04H	SNOOPQ_REQUESTS.INVALIDATE	Counts the number of snoop invalidate requests.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B7H	01H	OFF_CORE_RESPONSE_0	See Section 18.3.1.1.3, "Off-core Response Performance Monitoring in the Processor Core".	Requires programming MSR 01A6H.
B8H	01H	SNOOP_RESPONSE.HIT	Counts HIT snoop response sent by this thread in response to a snoop request.	
B8H	02H	SNOOP_RESPONSE.HITE	Counts HIT E snoop response sent by this thread in response to a snoop request.	
B8H	04H	SNOOP_RESPONSE.HITM	Counts HIT M snoop response sent by this thread in response to a snoop request.	
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.3.1.1.3, "Off-core Response Performance Monitoring in the Processor Core".	Use MSR 01A7H.
COH	00H	INST_RETIRED.ANY_P	See Table 19-1. Notes: INST_RETIRED.ANY is counted by a designated fixed counter. INST_RETIRED.ANY_P is counted by a programmable counter and is an architectural performance event. Event is supported if CPUID.A.EBX[1] = 0.	Counting: Faulting executions of GETSEC/VM entry/VM Exit/MWait will not count as retired instructions.
COH	02H	INST_RETIRED.X87	Counts the number of floating point computational operations retired: floating point computational operations executed by the assist handler and sub-operations of complex floating point instructions like transcendental instructions.	
COH	04H	INST_RETIRED.MMX	Counts the number of retired: MMX instructions.	
C2H	01H	UOPS_RETIRED.ANY	Counts the number of micro-ops retired, (macro-fused=1, micro-fused=2, others=1; maximum count of 8 per cycle). Most instructions are composed of one or two micro-ops. Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists.	Use cmask=1 and invert to count active cycles or stalled cycles.
C2H	02H	UOPS_RETIRED.RETIRE_SLOT_S	Counts the number of retirement slots used each cycle.	
C2H	04H	UOPS_RETIRED.MACRO_FUSED	Counts number of macro-fused uops retired.	
C3H	01H	MACHINE_CLEARS.CYCLES	Counts the cycles machine clear is asserted.	
C3H	02H	MACHINE_CLEARS.MEM_ORDER	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEARS.SMC	Counts the number of times that a program writes to a code section. Self-modifying code causes a severe penalty in all Intel 64 and IA-32 processors. The modified cache line is written back to the L2 and L3caches.	
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.	
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Counts the number of direct & indirect near unconditional calls retired.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C5H	00H	BR_MISP_RETIRE	Mispredicted branch instructions at retirement.	See Table 19-1.
C5H	01H	BR_MISP_RETIRE.CONDIT	Counts mispredicted conditional retired calls.	
C5H	02H	BR_MISP_RETIRE.NEAR_C	Counts mispredicted direct & indirect near unconditional retired calls.	
C5H	04H	BR_MISP_RETIRE.ALL_B	Counts all mispredicted retired calls.	
C7H	01H	SSEX_UOPS_RETIRE.PACK	Counts SIMD packed single-precision floating-point uops retired.	
C7H	02H	SSEX_UOPS_RETIRE.SCAL	Counts SIMD scalar single-precision floating-point uops retired.	
C7H	04H	SSEX_UOPS_RETIRE.PACK	Counts SIMD packed double-precision floating-point uops retired.	
C7H	08H	SSEX_UOPS_RETIRE.SCAL	Counts SIMD scalar double-precision floating-point uops retired.	
C7H	10H	SSEX_UOPS_RETIRE.VECT	Counts 128-bit SIMD vector integer uops retired.	
C8H	20H	ITLB_MISS_RETIRE	Counts the number of retired instructions that missed the ITLB when the instruction was fetched.	
CBH	01H	MEM_LOAD_RETIRE.L1D_H	Counts number of retired loads that hit the L1 data cache.	
CBH	02H	MEM_LOAD_RETIRE.L2_H	Counts number of retired loads that hit the L2 data cache.	
CBH	04H	MEM_LOAD_RETIRE.L3_UN	Counts number of retired loads that hit their own, unshared lines in the L3 cache.	
CBH	08H	MEM_LOAD_RETIRE.OTHER	Counts number of retired loads that hit in a sibling core's L2 (on die core). Since the L3 is inclusive of all cores on the package, this is an L3 hit. This counts both clean and modified hits.	
CBH	10H	MEM_LOAD_RETIRE.L3_MI	Counts number of retired loads that miss the L3 cache. The load was satisfied by a remote socket, local memory or an IOH.	
CBH	40H	MEM_LOAD_RETIRE.HIT_L	Counts number of retired loads that miss the L1D and the address is located in an allocated line fill buffer and will soon be committed to cache. This is counting secondary L1D misses.	
CBH	80H	MEM_LOAD_RETIRE.DTLB	Counts the number of retired loads that missed the DTLB. The DTLB miss is not counted if the load operation causes a fault. This event counts loads from cacheable memory only. The event does not count loads by software prefetches. Counts both primary and secondary misses to the TLB.	
CCH	01H	FP_MMX_TRANS.TO_FP	Counts the first floating-point instruction following any MMX instruction. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	



**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
CCH	02H	FP_MMX_TRANS.TO_MMX	Counts the first MMX instruction following a floating-point instruction. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
CCH	03H	FP_MMX_TRANS.ANY	Counts all transitions from floating point to MMX instructions and from MMX instructions to floating point instructions. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
DOH	01H	MACRO_INSTS.DECODED	Counts the number of instructions decoded, (but not necessarily executed or retired).	
D1H	01H	UOPS_DECODED.STALL_CYCLE S	Counts the cycles of decoder stalls. INV=1, Cmask=1.	
D1H	02H	UOPS_DECODED.MS	Counts the number of Uops decoded by the Microcode Sequencer, MS. The MS delivers uops when the instruction is more than 4 uops long or a microcode assist is occurring.	
D1H	04H	UOPS_DECODED.ESP_FOLDIN G	Counts number of stack pointer (ESP) instructions decoded: push, pop, call, ret, etc. ESP instructions do not generate a Uop to increment or decrement ESP. Instead, they update an ESP_Offset register that keeps track of the delta to the current value of the ESP register.	
D1H	08H	UOPS_DECODED.ESP_SYNC	Counts number of stack pointer (ESP) sync operations where an ESP instruction is corrected by adding the ESP offset register to the current value of the ESP register.	
D2H	01H	RAT_STALLS.FLAGS	Counts the number of cycles during which execution stalled due to several reasons, one of which is a partial flag register stall. A partial register stall may occur when two conditions are met: 1) an instruction modifies some, but not all, of the flags in the flag register and 2) the next instruction, which depends on flags, depends on flags that were not modified by this instruction.	
D2H	02H	RAT_STALLS.REGISTERS	This event counts the number of cycles instruction execution latency became longer than the defined latency because the instruction used a register that was partially written by previous instruction.	
D2H	04H	RAT_STALLS.ROB_READ_POR T	Counts the number of cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the out-of-order pipeline. Note that, at this stage in the pipeline, additional stalls may occur at the same cycle and prevent the stalled micro-ops from entering the pipe. In such a case, micro-ops retry entering the execution pipe in the next cycle and the ROB-read port stall is counted again.	
D2H	08H	RAT_STALLS.SCOREBOARD	Counts the cycles where we stall due to microarchitecturally required serialization. Microcode scoreboarding stalls.	



**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D2H	0FH	RAT_STALLS.ANY	Counts all Register Allocation Table stall cycles due to: Cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the execution pipe, Cycles when partial register stalls occurred, Cycles when flag stalls occurred, Cycles floating-point unit (FPU) status word stalls occurred. To count each of these conditions separately use the events: RAT_STALLS.ROB_READ_PORT, RAT_STALLS.PARTIAL, RAT_STALLS.FLAGS, and RAT_STALLS.FPSW.	
D4H	01H	SEG_RENAME_STALLS	Counts the number of stall cycles due to the lack of renaming resources for the ES, DS, FS, and GS segment registers. If a segment is renamed but not retired and a second update to the same segment occurs, a stall occurs in the front end of the pipeline until the renamed segment retires.	
D5H	01H	ES_REG_RENAMES	Counts the number of times the ES segment register is renamed.	
DBH	01H	UOP_UNFUSION	Counts unfusion events due to floating point exception to a fused uop.	
E0H	01H	BR_INST_DECODED	Counts the number of branch instructions decoded.	
E5H	01H	BPU_MISSED_CALL_RET	Counts number of times the Branch Prediction Unit missed predicting a call or return branch.	
E6H	01H	BACLEAR.CLEAR	Counts the number of times the front end is resteeered, mainly when the Branch Prediction Unit cannot provide a correct prediction and this is corrected by the Branch Address Calculator at the front end. This can occur if the code has many branches such that they cannot be consumed by the BPU. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline. The effect on total execution time depends on the surrounding code.	
E6H	02H	BACLEAR.BAD_TARGET	Counts number of Branch Address Calculator clears (BACLEAR) asserted due to conditional branch instructions in which there was a target hit but the direction was wrong. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline.	
E8H	01H	BPU_CLEARS.EARLY	Counts early (normal) Branch Prediction Unit clears: BPU predicted a taken branch after incorrectly assuming that it was not taken.	The BPU clear leads to 2 cycle bubble in the front end.
E8H	02H	BPU_CLEARS.LATE	Counts late Branch Prediction Unit clears due to Most Recently Used conflicts. The PBU clear leads to a 3 cycle bubble in the front end.	
ECH	01H	THREAD_ACTIVE	Counts cycles threads are active.	
FOH	01H	L2_TRANSACTIONS.LOAD	Counts L2 load operations due to HW prefetch or demand loads.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
F0H	02H	L2_TRANSACTIONS.RFO	Counts L2 RFO operations due to HW prefetch or demand RFOs.	
F0H	04H	L2_TRANSACTIONS.IFETCH	Counts L2 instruction fetch operations due to HW prefetch or demand ifetch.	
F0H	08H	L2_TRANSACTIONS.PREFETCH	Counts L2 prefetch operations.	
F0H	10H	L2_TRANSACTIONS.L1D_WB	Counts L1D writeback operations to the L2.	
F0H	20H	L2_TRANSACTIONS.FILL	Counts L2 cache line fill operations due to load, RFO, L1D writeback or prefetch.	
F0H	40H	L2_TRANSACTIONS.WB	Counts L2 writeback operations to the L3.	
F0H	80H	L2_TRANSACTIONS.ANY	Counts all L2 cache operations.	
F1H	02H	L2_LINES_IN.S_STATE	Counts the number of cache lines allocated in the L2 cache in the S (shared) state.	
F1H	04H	L2_LINES_IN.E_STATE	Counts the number of cache lines allocated in the L2 cache in the E (exclusive) state.	
F1H	07H	L2_LINES_IN.ANY	Counts the number of cache lines allocated in the L2 cache.	
F2H	01H	L2_LINES_OUT.DEMAND_CLEAN	Counts L2 clean cache lines evicted by a demand request.	
F2H	02H	L2_LINES_OUT.DEMAND_DIRTY	Counts L2 dirty (modified) cache lines evicted by a demand request.	
F2H	04H	L2_LINES_OUT.PREFETCH_CLEAN	Counts L2 clean cache line evicted by a prefetch request.	
F2H	08H	L2_LINES_OUT.PREFETCH_DIRTY	Counts L2 modified cache line evicted by a prefetch request.	
F2H	0FH	L2_LINES_OUT.ANY	Counts all L2 cache lines evicted for any reason.	
F4H	04H	SQ_MISC.LRU_HINTS	Counts number of Super Queue LRU hints sent to L3.	
F4H	10H	SQ_MISC.SPLIT_LOCK	Counts the number of SQ lock splits across a cache line.	
F6H	01H	SQ_FULL_STALL_CYCLES	Counts cycles the Super Queue is full. Neither of the threads on this core will be able to access the uncore.	
F7H	01H	FP_ASSIST.ALL	Counts the number of floating point operations executed that required micro-code assist intervention. Assists are required in the following cases: SSE instructions, (Denormal input when the DAZ flag is off or Underflow result when the FTZ flag is off); x87 instructions, (NaN or denormal are loaded to a register or used as input from memory, Division by 0 or Underflow output).	
F7H	02H	FP_ASSIST.OUTPUT	Counts number of floating point micro-code assist when the output value (destination register) is invalid.	
F7H	04H	FP_ASSIST.INPUT	Counts number of floating point micro-code assist when the input value (one of the source operands to an FP instruction) is invalid.	

**Table 19-21. Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
FDH	01H	SIMD_INT_64.PACKED_MPY	Counts number of SIMD integer 64 bit packed multiply operations.	
FDH	02H	SIMD_INT_64.PACKED_SHIFT	Counts number of SIMD integer 64 bit packed shift operations.	
FDH	04H	SIMD_INT_64.PACK	Counts number of SIMD integer 64 bit pack operations.	
FDH	08H	SIMD_INT_64.UNPACK	Counts number of SIMD integer 64 bit unpack operations.	
FDH	10H	SIMD_INT_64.PACKED_LOGICAL	Counts number of SIMD integer 64 bit logical operations.	
FDH	20H	SIMD_INT_64.PACKED_ARITH	Counts number of SIMD integer 64 bit arithmetic operations.	
FDH	40H	SIMD_INT_64.SHUFFLE_MOVE	Counts number of SIMD integer 64 bit shift or move operations.	

Model-specific performance monitoring events of the uncore sub-system for processors with CPUID signature of DisplayFamily\_DisplayModel 06\_25H, 06\_2CH, and 06\_1FH support performance events listed in Table 19-22.

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
00H	01H	UNC_GQ_CYCLES_FULL.READ_TRACKER	Uncore cycles Global Queue read tracker is full.	
00H	02H	UNC_GQ_CYCLES_FULL.WRITE_TRACKER	Uncore cycles Global Queue write tracker is full.	
00H	04H	UNC_GQ_CYCLES_FULL.PEER_PROBE_TRACKER	Uncore cycles Global Queue peer probe tracker is full. The peer probe tracker queue tracks snoops from the IOH and remote sockets.	
01H	01H	UNC_GQ_CYCLES_NOT_EMPTY.READ_TRACKER	Uncore cycles were Global Queue read tracker has at least one valid entry.	
01H	02H	UNC_GQ_CYCLES_NOT_EMPTY.WRITE_TRACKER	Uncore cycles were Global Queue write tracker has at least one valid entry.	
01H	04H	UNC_GQ_CYCLES_NOT_EMPTY.PEER_PROBE_TRACKER	Uncore cycles were Global Queue peer probe tracker has at least one valid entry. The peer probe tracker queue tracks IOH and remote socket snoops.	
02H	01H	UNC_GQ_OCCUPANCY.READ_TRACKER	Increments the number of queue entries (code read, data read, and RFOs) in the tread tracker. The GQ read tracker allocate to deallocate occupancy count is divided by the count to obtain the average read tracker latency.	
03H	01H	UNC_GQ_ALLOC.READ_TRACKER	Counts the number of tread tracker allocate to deallocate entries. The GQ read tracker allocate to deallocate occupancy count is divided by the count to obtain the average read tracker latency.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	UNC_GQ_ALLOC.RT_L3_MISS	Counts the number GQ read tracker entries for which a full cache line read has missed the L3. The GQ read tracker L3 miss to fill occupancy count is divided by this count to obtain the average cache line read L3 miss latency. The latency represents the time after which the L3 has determined that the cache line has missed. The time between a GQ read tracker allocation and the L3 determining that the cache line has missed is the average L3 hit latency. The total L3 cache line read miss latency is the hit latency + L3 miss latency.	
03H	04H	UNC_GQ_ALLOC.RT_TO_L3_RESP	Counts the number of GQ read tracker entries that are allocated in the read tracker queue that hit or miss the L3. The GQ read tracker L3 hit occupancy count is divided by this count to obtain the average L3 hit latency.	
03H	08H	UNC_GQ_ALLOC.RT_TO_RTID_ACQUIRED	Counts the number of GQ read tracker entries that are allocated in the read tracker, have missed in the L3 and have not acquired a Request Transaction ID. The GQ read tracker L3 miss to RTID acquired occupancy count is divided by this count to obtain the average latency for a read L3 miss to acquire an RTID.	
03H	10H	UNC_GQ_ALLOC.WT_TO_RTID_ACQUIRED	Counts the number of GQ write tracker entries that are allocated in the write tracker, have missed in the L3 and have not acquired a Request Transaction ID. The GQ write tracker L3 miss to RTID occupancy count is divided by this count to obtain the average latency for a write L3 miss to acquire an RTID.	
03H	20H	UNC_GQ_ALLOC.WRITE_TRACKER	Counts the number of GQ write tracker entries that are allocated in the write tracker queue that miss the L3. The GQ write tracker occupancy count is divided by this count to obtain the average L3 write miss latency.	
03H	40H	UNC_GQ_ALLOC.PEER_PROBE_TRACKER	Counts the number of GQ peer probe tracker (snoop) entries that are allocated in the peer probe tracker queue that miss the L3. The GQ peer probe occupancy count is divided by this count to obtain the average L3 peer probe miss latency.	
04H	01H	UNC_GQ_DATA.FROM_QPI	Cycles Global Queue Quickpath Interface input data port is busy importing data from the Quickpath Interface. Each cycle the input port can transfer 8 or 16 bytes of data.	
04H	02H	UNC_GQ_DATA.FROM_QMC	Cycles Global Queue Quickpath Memory Interface input data port is busy importing data from the Quickpath Memory Interface. Each cycle the input port can transfer 8 or 16 bytes of data.	
04H	04H	UNC_GQ_DATA.FROM_L3	Cycles GQ L3 input data port is busy importing data from the Last Level Cache. Each cycle the input port can transfer 32 bytes of data.	
04H	08H	UNC_GQ_DATA.FROM_CORES_02	Cycles GQ Core 0 and 2 input data port is busy importing data from processor cores 0 and 2. Each cycle the input port can transfer 32 bytes of data.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
04H	10H	UNC_GQ_DATA.FROM_CORES_13	Cycles GQ Core 1 and 3 input data port is busy importing data from processor cores 1 and 3. Each cycle the input port can transfer 32 bytes of data.	
05H	01H	UNC_GQ_DATA.TO_QPI_QMC	Cycles GQ QPI and QMC output data port is busy sending data to the Quickpath Interface or Quickpath Memory Interface. Each cycle the output port can transfer 32 bytes of data.	
05H	02H	UNC_GQ_DATA.TO_L3	Cycles GQ L3 output data port is busy sending data to the Last Level Cache. Each cycle the output port can transfer 32 bytes of data.	
05H	04H	UNC_GQ_DATA.TO_CORES	Cycles GQ Core output data port is busy sending data to the Cores. Each cycle the output port can transfer 32 bytes of data.	
06H	01H	UNC_SNP_RESP_TO_LOCAL_HOME.I_STATE	Number of snoop responses to the local home that L3 does not have the referenced cache line.	
06H	02H	UNC_SNP_RESP_TO_LOCAL_HOME.S_STATE	Number of snoop responses to the local home that L3 has the referenced line cached in the S state.	
06H	04H	UNC_SNP_RESP_TO_LOCAL_HOME.FWD_S_STATE	Number of responses to code or data read snoops to the local home that the L3 has the referenced cache line in the E state. The L3 cache line state is changed to the S state and the line is forwarded to the local home in the S state.	
06H	08H	UNC_SNP_RESP_TO_LOCAL_HOME.FWD_I_STATE	Number of responses to read invalidate snoops to the local home that the L3 has the referenced cache line in the M state. The L3 cache line state is invalidated and the line is forwarded to the local home in the M state.	
06H	10H	UNC_SNP_RESP_TO_LOCAL_HOME.CONFLICT	Number of conflict snoop responses sent to the local home.	
06H	20H	UNC_SNP_RESP_TO_LOCAL_HOME.WB	Number of responses to code or data read snoops to the local home that the L3 has the referenced line cached in the M state.	
07H	01H	UNC_SNP_RESP_TO_REMOTE_HOME.I_STATE	Number of snoop responses to a remote home that L3 does not have the referenced cache line.	
07H	02H	UNC_SNP_RESP_TO_REMOTE_HOME.S_STATE	Number of snoop responses to a remote home that L3 has the referenced line cached in the S state.	
07H	04H	UNC_SNP_RESP_TO_REMOTE_HOME.FWD_S_STATE	Number of responses to code or data read snoops to a remote home that the L3 has the referenced cache line in the E state. The L3 cache line state is changed to the S state and the line is forwarded to the remote home in the S state.	
07H	08H	UNC_SNP_RESP_TO_REMOTE_HOME.FWD_I_STATE	Number of responses to read invalidate snoops to a remote home that the L3 has the referenced cache line in the M state. The L3 cache line state is invalidated and the line is forwarded to the remote home in the M state.	
07H	10H	UNC_SNP_RESP_TO_REMOTE_HOME.CONFLICT	Number of conflict snoop responses sent to the local home.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
07H	20H	UNC_SNP_RESP_TO_REMOTE_HOME.WB	Number of responses to code or data read snoops to a remote home that the L3 has the referenced line cached in the M state.	
07H	24H	UNC_SNP_RESP_TO_REMOTE_HOME.HITM	Number of HITM snoop responses to a remote home.	
08H	01H	UNC_L3_HITS.READ	Number of code read, data read and RFO requests that hit in the L3.	
08H	02H	UNC_L3_HITS.WRITE	Number of writeback requests that hit in the L3. Writebacks from the cores will always result in L3 hits due to the inclusive property of the L3.	
08H	04H	UNC_L3_HITS.PROBE	Number of snoops from IOH or remote sockets that hit in the L3.	
08H	03H	UNC_L3_HITS.ANY	Number of reads and writes that hit the L3.	
09H	01H	UNC_L3_MISS.READ	Number of code read, data read and RFO requests that miss the L3.	
09H	02H	UNC_L3_MISS.WRITE	Number of writeback requests that miss the L3. Should always be zero as writebacks from the cores will always result in L3 hits due to the inclusive property of the L3.	
09H	04H	UNC_L3_MISS.PROBE	Number of snoops from IOH or remote sockets that miss the L3.	
09H	03H	UNC_L3_MISS.ANY	Number of reads and writes that miss the L3.	
0AH	01H	UNC_L3_LINES_IN.M_STATE	Counts the number of L3 lines allocated in M state. The only time a cache line is allocated in the M state is when the line was forwarded in M state is forwarded due to a Snoop Read Invalidate Own request.	
0AH	02H	UNC_L3_LINES_IN.E_STATE	Counts the number of L3 lines allocated in E state.	
0AH	04H	UNC_L3_LINES_IN.S_STATE	Counts the number of L3 lines allocated in S state.	
0AH	08H	UNC_L3_LINES_IN.F_STATE	Counts the number of L3 lines allocated in F state.	
0AH	0FH	UNC_L3_LINES_IN.ANY	Counts the number of L3 lines allocated in any state.	
0BH	01H	UNC_L3_LINES_OUT.M_STATE	Counts the number of L3 lines victimized that were in the M state. When the victim cache line is in M state, the line is written to its home cache agent which can be either local or remote.	
0BH	02H	UNC_L3_LINES_OUT.E_STATE	Counts the number of L3 lines victimized that were in the E state.	
0BH	04H	UNC_L3_LINES_OUT.S_STATE	Counts the number of L3 lines victimized that were in the S state.	
0BH	08H	UNC_L3_LINES_OUT.I_STATE	Counts the number of L3 lines victimized that were in the I state.	
0BH	10H	UNC_L3_LINES_OUT.F_STATE	Counts the number of L3 lines victimized that were in the F state.	
0BH	1FH	UNC_L3_LINES_OUT.ANY	Counts the number of L3 lines victimized in any state.	
0CH	01H	UNC_GQ_SNOOP.GOTO_S	Counts the number of remote snoops that have requested a cache line be set to the S state.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
0CH	02H	UNC_GQ_SNOOP.GOTO_I	Counts the number of remote snoops that have requested a cache line be set to the I state.	
0CH	04H	UNC_GQ_SNOOP.GOTO_S_HIT_E	Counts the number of remote snoops that have requested a cache line be set to the S state from E state.	Requires writing MSR 301H with mask = 2H.
0CH	04H	UNC_GQ_SNOOP.GOTO_S_HIT_F	Counts the number of remote snoops that have requested a cache line be set to the S state from F (forward) state.	Requires writing MSR 301H with mask = 8H.
0CH	04H	UNC_GQ_SNOOP.GOTO_S_HIT_M	Counts the number of remote snoops that have requested a cache line be set to the S state from M state.	Requires writing MSR 301H with mask = 1H.
0CH	04H	UNC_GQ_SNOOP.GOTO_S_HIT_S	Counts the number of remote snoops that have requested a cache line be set to the S state from S state.	Requires writing MSR 301H with mask = 4H.
0CH	08H	UNC_GQ_SNOOP.GOTO_I_HIT_E	Counts the number of remote snoops that have requested a cache line be set to the I state from E state.	Requires writing MSR 301H with mask = 2H.
0CH	08H	UNC_GQ_SNOOP.GOTO_I_HIT_F	Counts the number of remote snoops that have requested a cache line be set to the I state from F (forward) state.	Requires writing MSR 301H with mask = 8H.
0CH	08H	UNC_GQ_SNOOP.GOTO_I_HIT_M	Counts the number of remote snoops that have requested a cache line be set to the I state from M state.	Requires writing MSR 301H with mask = 1H.
0CH	08H	UNC_GQ_SNOOP.GOTO_I_HIT_S	Counts the number of remote snoops that have requested a cache line be set to the I state from S state.	Requires writing MSR 301H with mask = 4H.
20H	01H	UNC_QHL_REQUESTS.IOH_READS	Counts number of Quickpath Home Logic read requests from the IOH.	
20H	02H	UNC_QHL_REQUESTS.IOH_WRITES	Counts number of Quickpath Home Logic write requests from the IOH.	
20H	04H	UNC_QHL_REQUESTS.REMOTE_READS	Counts number of Quickpath Home Logic read requests from a remote socket.	
20H	08H	UNC_QHL_REQUESTS.REMOTE_WRITES	Counts number of Quickpath Home Logic write requests from a remote socket.	
20H	10H	UNC_QHL_REQUESTS.LOCAL_READS	Counts number of Quickpath Home Logic read requests from the local socket.	
20H	20H	UNC_QHL_REQUESTS.LOCAL_WRITES	Counts number of Quickpath Home Logic write requests from the local socket.	
21H	01H	UNC_QHL_CYCLES_FULL.IOH	Counts uclk cycles all entries in the Quickpath Home Logic IOH are full.	
21H	02H	UNC_QHL_CYCLES_FULL.REMOTE	Counts uclk cycles all entries in the Quickpath Home Logic remote tracker are full.	
21H	04H	UNC_QHL_CYCLES_FULL.LOCAL	Counts uclk cycles all entries in the Quickpath Home Logic local tracker are full.	
22H	01H	UNC_QHL_CYCLES_NOT_EMPTY.IOH	Counts uclk cycles all entries in the Quickpath Home Logic IOH is busy.	



**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
22H	02H	UNC_QHL_CYCLES_NOT_EMPTY.REMOTE	Counts uclk cycles all entries in the Quickpath Home Logic remote tracker is busy.	
22H	04H	UNC_QHL_CYCLES_NOT_EMPTY.LOCAL	Counts uclk cycles all entries in the Quickpath Home Logic local tracker is busy.	
23H	01H	UNC_QHL_OCCUPANCY.IOH	QHL IOH tracker allocate to deallocate read occupancy.	
23H	02H	UNC_QHL_OCCUPANCY.REMOTE	QHL remote tracker allocate to deallocate read occupancy.	
23H	04H	UNC_QHL_OCCUPANCY.LOCAL	QHL local tracker allocate to deallocate read occupancy.	
24H	02H	UNC_QHL_ADDRESS_CONFLICTS.2WAY	Counts number of QHL Active Address Table (AAT) entries that saw a max of 2 conflicts. The AAT is a structure that tracks requests that are in conflict. The requests themselves are in the home tracker entries. The count is reported when an AAT entry deallocates.	
24H	04H	UNC_QHL_ADDRESS_CONFLICTS.3WAY	Counts number of QHL Active Address Table (AAT) entries that saw a max of 3 conflicts. The AAT is a structure that tracks requests that are in conflict. The requests themselves are in the home tracker entries. The count is reported when an AAT entry deallocates.	
25H	01H	UNC_QHL_CONFLICT_CYCLES.IOH	Counts cycles the Quickpath Home Logic IOH Tracker contains two or more requests with an address conflict. A max of 3 requests can be in conflict.	
25H	02H	UNC_QHL_CONFLICT_CYCLES.REMOTE	Counts cycles the Quickpath Home Logic Remote Tracker contains two or more requests with an address conflict. A max of 3 requests can be in conflict.	
25H	04H	UNC_QHL_CONFLICT_CYCLES.LOCAL	Counts cycles the Quickpath Home Logic Local Tracker contains two or more requests with an address conflict. A max of 3 requests can be in conflict.	
26H	01H	UNC_QHL_TO_QMC_BYPASS	Counts number or requests to the Quickpath Memory Controller that bypass the Quickpath Home Logic. All local accesses can be bypassed. For remote requests, only read requests can be bypassed.	
28H	01H	UNC_QMC_ISOC_FULL.READ.CH0	Counts cycles all the entries in the DRAM channel 0 high priority queue are occupied with isochronous read requests.	
28H	02H	UNC_QMC_ISOC_FULL.READ.CH1	Counts cycles all the entries in the DRAM channel 1 high priority queue are occupied with isochronous read requests.	
28H	04H	UNC_QMC_ISOC_FULL.READ.CH2	Counts cycles all the entries in the DRAM channel 2 high priority queue are occupied with isochronous read requests.	
28H	08H	UNC_QMC_ISOC_FULL.WRITE.CH0	Counts cycles all the entries in the DRAM channel 0 high priority queue are occupied with isochronous write requests.	
28H	10H	UNC_QMC_ISOC_FULL.WRITE.CH1	Counts cycles all the entries in the DRAM channel 1 high priority queue are occupied with isochronous write requests.	



**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
28H	20H	UNC_QMC_ISOC_FULL.WRITE.CH2	Counts cycles all the entries in the DRAM channel 2 high priority queue are occupied with isochronous write requests.	
29H	01H	UNC_QMC_BUSY.READ.CH0	Counts cycles where Quickpath Memory Controller has at least 1 outstanding read request to DRAM channel 0.	
29H	02H	UNC_QMC_BUSY.READ.CH1	Counts cycles where Quickpath Memory Controller has at least 1 outstanding read request to DRAM channel 1.	
29H	04H	UNC_QMC_BUSY.READ.CH2	Counts cycles where Quickpath Memory Controller has at least 1 outstanding read request to DRAM channel 2.	
29H	08H	UNC_QMC_BUSY.WRITE.CH0	Counts cycles where Quickpath Memory Controller has at least 1 outstanding write request to DRAM channel 0.	
29H	10H	UNC_QMC_BUSY.WRITE.CH1	Counts cycles where Quickpath Memory Controller has at least 1 outstanding write request to DRAM channel 1.	
29H	20H	UNC_QMC_BUSY.WRITE.CH2	Counts cycles where Quickpath Memory Controller has at least 1 outstanding write request to DRAM channel 2.	
2AH	01H	UNC_QMC_OCCUPANCY.CH0	IMC channel 0 normal read request occupancy.	
2AH	02H	UNC_QMC_OCCUPANCY.CH1	IMC channel 1 normal read request occupancy.	
2AH	04H	UNC_QMC_OCCUPANCY.CH2	IMC channel 2 normal read request occupancy.	
2AH	07H	UNC_QMC_OCCUPANCY.ANY	Normal read request occupancy for any channel.	
2BH	01H	UNC_QMC_ISSOC_OCCUPANCY.CH0	IMC channel 0 issoc read request occupancy.	
2BH	02H	UNC_QMC_ISSOC_OCCUPANCY.CH1	IMC channel 1 issoc read request occupancy.	
2BH	04H	UNC_QMC_ISSOC_OCCUPANCY.CH2	IMC channel 2 issoc read request occupancy.	
2BH	07H	UNC_QMC_ISSOC_READS.ANY	IMC issoc read request occupancy.	
2CH	01H	UNC_QMC_NORMAL_READS.CH0	Counts the number of Quickpath Memory Controller channel 0 medium and low priority read requests. The QMC channel 0 normal read occupancy divided by this count provides the average QMC channel 0 read latency.	
2CH	02H	UNC_QMC_NORMAL_READS.CH1	Counts the number of Quickpath Memory Controller channel 1 medium and low priority read requests. The QMC channel 1 normal read occupancy divided by this count provides the average QMC channel 1 read latency.	
2CH	04H	UNC_QMC_NORMAL_READS.CH2	Counts the number of Quickpath Memory Controller channel 2 medium and low priority read requests. The QMC channel 2 normal read occupancy divided by this count provides the average QMC channel 2 read latency.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
2CH	07H	UNC_QMC_NORMAL_READS.ANY	Counts the number of Quickpath Memory Controller medium and low priority read requests. The QMC normal read occupancy divided by this count provides the average QMC read latency.	
2DH	01H	UNC_QMC_HIGH_PRIORITY_READS.CH0	Counts the number of Quickpath Memory Controller channel 0 high priority isochronous read requests.	
2DH	02H	UNC_QMC_HIGH_PRIORITY_READS.CH1	Counts the number of Quickpath Memory Controller channel 1 high priority isochronous read requests.	
2DH	04H	UNC_QMC_HIGH_PRIORITY_READS.CH2	Counts the number of Quickpath Memory Controller channel 2 high priority isochronous read requests.	
2DH	07H	UNC_QMC_HIGH_PRIORITY_READS.ANY	Counts the number of Quickpath Memory Controller high priority isochronous read requests.	
2EH	01H	UNC_QMC_CRITICAL_PRIORITY_READS.CH0	Counts the number of Quickpath Memory Controller channel 0 critical priority isochronous read requests.	
2EH	02H	UNC_QMC_CRITICAL_PRIORITY_READS.CH1	Counts the number of Quickpath Memory Controller channel 1 critical priority isochronous read requests.	
2EH	04H	UNC_QMC_CRITICAL_PRIORITY_READS.CH2	Counts the number of Quickpath Memory Controller channel 2 critical priority isochronous read requests.	
2EH	07H	UNC_QMC_CRITICAL_PRIORITY_READS.ANY	Counts the number of Quickpath Memory Controller critical priority isochronous read requests.	
2FH	01H	UNC_QMC_WRITES.FULL.CH0	Counts number of full cache line writes to DRAM channel 0.	
2FH	02H	UNC_QMC_WRITES.FULL.CH1	Counts number of full cache line writes to DRAM channel 1.	
2FH	04H	UNC_QMC_WRITES.FULL.CH2	Counts number of full cache line writes to DRAM channel 2.	
2FH	07H	UNC_QMC_WRITES.FULL.ANY	Counts number of full cache line writes to DRAM.	
2FH	08H	UNC_QMC_WRITES.PARTIAL.CH0	Counts number of partial cache line writes to DRAM channel 0.	
2FH	10H	UNC_QMC_WRITES.PARTIAL.CH1	Counts number of partial cache line writes to DRAM channel 1.	
2FH	20H	UNC_QMC_WRITES.PARTIAL.CH2	Counts number of partial cache line writes to DRAM channel 2.	
2FH	38H	UNC_QMC_WRITES.PARTIAL.ANY	Counts number of partial cache line writes to DRAM.	
30H	01H	UNC_QMC_CANCEL.CH0	Counts number of DRAM channel 0 cancel requests.	
30H	02H	UNC_QMC_CANCEL.CH1	Counts number of DRAM channel 1 cancel requests.	
30H	04H	UNC_QMC_CANCEL.CH2	Counts number of DRAM channel 2 cancel requests.	
30H	07H	UNC_QMC_CANCEL.ANY	Counts number of DRAM cancel requests.	
31H	01H	UNC_QMC_PRIORITY_UPDATE.S.CH0	Counts number of DRAM channel 0 priority updates. A priority update occurs when an ISOC high or critical request is received by the QHL and there is a matching request with normal priority that has already been issued to the QMC. In this instance, the QHL will send a priority update to QMC to expedite the request.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
31H	02H	UNC_QMC_PRIORITY_UPDATE.S.CH1	Counts number of DRAM channel 1 priority updates. A priority update occurs when an ISOC high or critical request is received by the QHL and there is a matching request with normal priority that has already been issued to the QMC. In this instance, the QHL will send a priority update to QMC to expedite the request.	
31H	04H	UNC_QMC_PRIORITY_UPDATE.S.CH2	Counts number of DRAM channel 2 priority updates. A priority update occurs when an ISOC high or critical request is received by the QHL and there is a matching request with normal priority that has already been issued to the QMC. In this instance, the QHL will send a priority update to QMC to expedite the request.	
31H	07H	UNC_QMC_PRIORITY_UPDATE.S.ANY	Counts number of DRAM priority updates. A priority update occurs when an ISOC high or critical request is received by the QHL and there is a matching request with normal priority that has already been issued to the QMC. In this instance, the QHL will send a priority update to QMC to expedite the request.	
32H	01H	UNC_IMC_RETRY.CHO	Counts number of IMC DRAM channel 0 retries. DRAM retry only occurs when configured in RAS mode.	
32H	02H	UNC_IMC_RETRY.CH1	Counts number of IMC DRAM channel 1 retries. DRAM retry only occurs when configured in RAS mode.	
32H	04H	UNC_IMC_RETRY.CH2	Counts number of IMC DRAM channel 2 retries. DRAM retry only occurs when configured in RAS mode.	
32H	07H	UNC_IMC_RETRY.ANY	Counts number of IMC DRAM retries from any channel. DRAM retry only occurs when configured in RAS mode.	
33H	01H	UNC_QHL_FRC_ACK_CNFLTS.IOH	Counts number of Force Acknowledge Conflict messages sent by the Quickpath Home Logic to the IOH.	
33H	02H	UNC_QHL_FRC_ACK_CNFLTS.REMOTE	Counts number of Force Acknowledge Conflict messages sent by the Quickpath Home Logic to the remote home.	
33H	04H	UNC_QHL_FRC_ACK_CNFLTS.LOCAL	Counts number of Force Acknowledge Conflict messages sent by the Quickpath Home Logic to the local home.	
33H	07H	UNC_QHL_FRC_ACK_CNFLTS.ANY	Counts number of Force Acknowledge Conflict messages sent by the Quickpath Home Logic.	
34H	01H	UNC_QHL_SLEEPS.IOH_ORDER	Counts number of occurrences a request was put to sleep due to IOH ordering (write after read) conflicts. While in the sleep state, the request is not eligible to be scheduled to the QMC.	
34H	02H	UNC_QHL_SLEEPS.REMOTE_ORDER	Counts number of occurrences a request was put to sleep due to remote socket ordering (write after read) conflicts. While in the sleep state, the request is not eligible to be scheduled to the QMC.	
34H	04H	UNC_QHL_SLEEPS.LOCAL_ORDER	Counts number of occurrences a request was put to sleep due to local socket ordering (write after read) conflicts. While in the sleep state, the request is not eligible to be scheduled to the QMC.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
34H	08H	UNC_QHL_SLEEPS.IOH_CONFLICT	Counts number of occurrences a request was put to sleep due to IOH address conflicts. While in the sleep state, the request is not eligible to be scheduled to the QMC.	
34H	10H	UNC_QHL_SLEEPS.REMOTE_CONFLICT	Counts number of occurrences a request was put to sleep due to remote socket address conflicts. While in the sleep state, the request is not eligible to be scheduled to the QMC.	
34H	20H	UNC_QHL_SLEEPS.LOCAL_CONFLICT	Counts number of occurrences a request was put to sleep due to local socket address conflicts. While in the sleep state, the request is not eligible to be scheduled to the QMC.	
35H	01H	UNC_ADDR_OPCODE_MATCH.IOH	Counts number of requests from the IOH, address/opcode of request is qualified by mask value written to MSR 396H. The following mask values are supported: 0: NONE 40000000_00000000H:RSPFWDI 40001A00_00000000H:RSPFWS 40001D00_00000000H:RSPIWB	Match opcode/address by writing MSR 396H with mask supported mask value.
35H	02H	UNC_ADDR_OPCODE_MATCH.REMOTE	Counts number of requests from the remote socket, address/opcode of request is qualified by mask value written to MSR 396H. The following mask values are supported: 0: NONE 40000000_00000000H:RSPFWDI 40001A00_00000000H:RSPFWS 40001D00_00000000H:RSPIWB	Match opcode/address by writing MSR 396H with mask supported mask value.
35H	04H	UNC_ADDR_OPCODE_MATCH.LOCAL	Counts number of requests from the local socket, address/opcode of request is qualified by mask value written to MSR 396H. The following mask values are supported: 0: NONE 40000000_00000000H:RSPFWDI 40001A00_00000000H:RSPFWS 40001D00_00000000H:RSPIWB	Match opcode/address by writing MSR 396H with mask supported mask value.
40H	01H	UNC_QPI_TX_STALLED_SINGLE_FLIT.HOME.LINK_0	Counts cycles the Quickpath outbound link 0 HOME virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	02H	UNC_QPI_TX_STALLED_SINGLE_FLIT.SNOOP.LINK_0	Counts cycles the Quickpath outbound link 0 SNOOP virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
40H	04H	UNC_QPI_TX_STALLED_SINGL E_FLIT.NDR.LINK_0	Counts cycles the Quickpath outbound link 0 non-data response virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	08H	UNC_QPI_TX_STALLED_SINGL E_FLIT.HOME.LINK_1	Counts cycles the Quickpath outbound link 1 HOME virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	10H	UNC_QPI_TX_STALLED_SINGL E_FLIT.SNOOP.LINK_1	Counts cycles the Quickpath outbound link 1 SNOOP virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	20H	UNC_QPI_TX_STALLED_SINGL E_FLIT.NDR.LINK_1	Counts cycles the Quickpath outbound link 1 non-data response virtual channel is stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	07H	UNC_QPI_TX_STALLED_SINGL E_FLIT.LINK_0	Counts cycles the Quickpath outbound link 0 virtual channels are stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
40H	38H	UNC_QPI_TX_STALLED_SINGL E_FLIT.LINK_1	Counts cycles the Quickpath outbound link 1 virtual channels are stalled due to lack of a VNA and VNO credit. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	01H	UNC_QPI_TX_STALLED_MULT _FLIT.DRS.LINK_0	Counts cycles the Quickpath outbound link 0 Data Response virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	02H	UNC_QPI_TX_STALLED_MULT _FLIT.NCB.LINK_0	Counts cycles the Quickpath outbound link 0 Non-Coherent Bypass virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	04H	UNC_QPI_TX_STALLED_MULT _FLIT.NCS.LINK_0	Counts cycles the Quickpath outbound link 0 Non-Coherent Standard virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
41H	08H	UNC_QPI_TX_STALLED_MULTI_FLIT.DRS.LINK_1	Counts cycles the Quickpath outbound link 1 Data Response virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	10H	UNC_QPI_TX_STALLED_MULTI_FLIT.NCB.LINK_1	Counts cycles the Quickpath outbound link 1 Non-Coherent Bypass virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	20H	UNC_QPI_TX_STALLED_MULTI_FLIT.NCS.LINK_1	Counts cycles the Quickpath outbound link 1 Non-Coherent Standard virtual channel is stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	07H	UNC_QPI_TX_STALLED_MULTI_FLIT.LINK_0	Counts cycles the Quickpath outbound link 0 virtual channels are stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
41H	38H	UNC_QPI_TX_STALLED_MULTI_FLIT.LINK_1	Counts cycles the Quickpath outbound link 1 virtual channels are stalled due to lack of VNA and VNO credits. Note that this event does not filter out when a flit would not have been selected for arbitration because another virtual channel is getting arbitrated.	
42H	01H	UNC_QPI_TX_HEADER.FULL.LINK_0	Number of cycles that the header buffer in the Quickpath Interface outbound link 0 is full.	
42H	02H	UNC_QPI_TX_HEADER.BUSY.LINK_0	Number of cycles that the header buffer in the Quickpath Interface outbound link 0 is busy.	
42H	04H	UNC_QPI_TX_HEADER.FULL.LINK_1	Number of cycles that the header buffer in the Quickpath Interface outbound link 1 is full.	
42H	08H	UNC_QPI_TX_HEADER.BUSY.LINK_1	Number of cycles that the header buffer in the Quickpath Interface outbound link 1 is busy.	
43H	01H	UNC_QPI_RX_NO_PPT_CREDIT.STALLS.LINK_0	Number of cycles that snoop packets incoming to the Quickpath Interface link 0 are stalled and not sent to the GQ because the GQ Peer Probe Tracker (PPT) does not have any available entries.	
43H	02H	UNC_QPI_RX_NO_PPT_CREDIT.STALLS.LINK_1	Number of cycles that snoop packets incoming to the Quickpath Interface link 1 are stalled and not sent to the GQ because the GQ Peer Probe Tracker (PPT) does not have any available entries.	
60H	01H	UNC_DRAM_OPEN.CHO	Counts number of DRAM Channel 0 open commands issued either for read or write. To read or write data, the referenced DRAM page must first be opened.	

**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
60H	02H	UNC_DRAM_OPEN.CH1	Counts number of DRAM Channel 1 open commands issued either for read or write. To read or write data, the referenced DRAM page must first be opened.	
60H	04H	UNC_DRAM_OPEN.CH2	Counts number of DRAM Channel 2 open commands issued either for read or write. To read or write data, the referenced DRAM page must first be opened.	
61H	01H	UNC_DRAM_PAGE_CLOSE.CH0	DRAM channel 0 command issued to CLOSE a page due to page idle timer expiration. Closing a page is done by issuing a precharge.	
61H	02H	UNC_DRAM_PAGE_CLOSE.CH1	DRAM channel 1 command issued to CLOSE a page due to page idle timer expiration. Closing a page is done by issuing a precharge.	
61H	04H	UNC_DRAM_PAGE_CLOSE.CH2	DRAM channel 2 command issued to CLOSE a page due to page idle timer expiration. Closing a page is done by issuing a precharge.	
62H	01H	UNC_DRAM_PAGE_MISS.CH0	Counts the number of precharges (PRE) that were issued to DRAM channel 0 because there was a page miss. A page miss refers to a situation in which a page is currently open and another page from the same bank needs to be opened. The new page experiences a page miss. Closing of the old page is done by issuing a precharge.	
62H	02H	UNC_DRAM_PAGE_MISS.CH1	Counts the number of precharges (PRE) that were issued to DRAM channel 1 because there was a page miss. A page miss refers to a situation in which a page is currently open and another page from the same bank needs to be opened. The new page experiences a page miss. Closing of the old page is done by issuing a precharge.	
62H	04H	UNC_DRAM_PAGE_MISS.CH2	Counts the number of precharges (PRE) that were issued to DRAM channel 2 because there was a page miss. A page miss refers to a situation in which a page is currently open and another page from the same bank needs to be opened. The new page experiences a page miss. Closing of the old page is done by issuing a precharge.	
63H	01H	UNC_DRAM_READ_CAS.CH0	Counts the number of times a read CAS command was issued on DRAM channel 0.	
63H	02H	UNC_DRAM_READ_CAS.AUTO PRE_CH0	Counts the number of times a read CAS command was issued on DRAM channel 0 where the command issued used the auto-precharge (auto page close) mode.	
63H	04H	UNC_DRAM_READ_CAS.CH1	Counts the number of times a read CAS command was issued on DRAM channel 1.	
63H	08H	UNC_DRAM_READ_CAS.AUTO PRE_CH1	Counts the number of times a read CAS command was issued on DRAM channel 1 where the command issued used the auto-precharge (auto page close) mode.	
63H	10H	UNC_DRAM_READ_CAS.CH2	Counts the number of times a read CAS command was issued on DRAM channel 2.	



**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
63H	20H	UNC_DRAM_READ_CAS.AUTO PRE_CH2	Counts the number of times a read CAS command was issued on DRAM channel 2 where the command issued used the auto-precharge (auto page close) mode.	
64H	01H	UNC_DRAM_WRITE_CAS.CHO	Counts the number of times a write CAS command was issued on DRAM channel 0.	
64H	02H	UNC_DRAM_WRITE_CAS.AUTO PRE_CH0	Counts the number of times a write CAS command was issued on DRAM channel 0 where the command issued used the auto-precharge (auto page close) mode.	
64H	04H	UNC_DRAM_WRITE_CAS.CH1	Counts the number of times a write CAS command was issued on DRAM channel 1.	
64H	08H	UNC_DRAM_WRITE_CAS.AUTO PRE_CH1	Counts the number of times a write CAS command was issued on DRAM channel 1 where the command issued used the auto-precharge (auto page close) mode.	
64H	10H	UNC_DRAM_WRITE_CAS.CH2	Counts the number of times a write CAS command was issued on DRAM channel 2.	
64H	20H	UNC_DRAM_WRITE_CAS.AUTO PRE_CH2	Counts the number of times a write CAS command was issued on DRAM channel 2 where the command issued used the auto-precharge (auto page close) mode.	
65H	01H	UNC_DRAM_REFRESH.CHO	Counts number of DRAM channel 0 refresh commands. DRAM loses data content over time. In order to keep correct data content, the data values have to be refreshed periodically.	
65H	02H	UNC_DRAM_REFRESH.CH1	Counts number of DRAM channel 1 refresh commands. DRAM loses data content over time. In order to keep correct data content, the data values have to be refreshed periodically.	
65H	04H	UNC_DRAM_REFRESH.CH2	Counts number of DRAM channel 2 refresh commands. DRAM loses data content over time. In order to keep correct data content, the data values have to be refreshed periodically.	
66H	01H	UNC_DRAM_PRE_ALL.CHO	Counts number of DRAM Channel 0 precharge-all (PREALL) commands that close all open pages in a rank. PREALL is issued when the DRAM needs to be refreshed or needs to go into a power down mode.	
66H	02H	UNC_DRAM_PRE_ALL.CH1	Counts number of DRAM Channel 1 precharge-all (PREALL) commands that close all open pages in a rank. PREALL is issued when the DRAM needs to be refreshed or needs to go into a power down mode.	
66H	04H	UNC_DRAM_PRE_ALL.CH2	Counts number of DRAM Channel 2 precharge-all (PREALL) commands that close all open pages in a rank. PREALL is issued when the DRAM needs to be refreshed or needs to go into a power down mode.	
67H	01H	UNC_DRAM_THERMAL_THROT TLED	Uncore cycles DRAM was throttled due to its temperature being above the thermal throttling threshold.	
80H	01H	UNC_THERMAL_THROTTLING_ TEMP.CORE_0	Cycles that the PCU records that core 0 is above the thermal throttling threshold temperature.	



**Table 19-22. Performance Events In the Processor Uncore for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
80H	02H	UNC_THERMAL_THROTTLING_TEMP.CORE_1	Cycles that the PCU records that core 1 is above the thermal throttling threshold temperature.	
80H	04H	UNC_THERMAL_THROTTLING_TEMP.CORE_2	Cycles that the PCU records that core 2 is above the thermal throttling threshold temperature.	
80H	08H	UNC_THERMAL_THROTTLING_TEMP.CORE_3	Cycles that the PCU records that core 3 is above the thermal throttling threshold temperature.	
81H	01H	UNC_THERMAL_THROTTLED_TEMP.CORE_0	Cycles that the PCU records that core 0 is in the power throttled state due to core's temperature being above the thermal throttling threshold.	
81H	02H	UNC_THERMAL_THROTTLED_TEMP.CORE_1	Cycles that the PCU records that core 1 is in the power throttled state due to core's temperature being above the thermal throttling threshold.	
81H	04H	UNC_THERMAL_THROTTLED_TEMP.CORE_2	Cycles that the PCU records that core 2 is in the power throttled state due to core's temperature being above the thermal throttling threshold.	
81H	08H	UNC_THERMAL_THROTTLED_TEMP.CORE_3	Cycles that the PCU records that core 3 is in the power throttled state due to core's temperature being above the thermal throttling threshold.	
82H	01H	UNC_PROCHOT_ASSERTION	Number of system assertions of PROCHOT indicating the entire processor has exceeded the thermal limit.	
83H	01H	UNC_THERMAL_THROTTLING_PROCHOT.CORE_0	Cycles that the PCU records that core 0 is a low power state due to the system asserting PROCHOT the entire processor has exceeded the thermal limit.	
83H	02H	UNC_THERMAL_THROTTLING_PROCHOT.CORE_1	Cycles that the PCU records that core 1 is a low power state due to the system asserting PROCHOT the entire processor has exceeded the thermal limit.	
83H	04H	UNC_THERMAL_THROTTLING_PROCHOT.CORE_2	Cycles that the PCU records that core 2 is a low power state due to the system asserting PROCHOT the entire processor has exceeded the thermal limit.	
83H	08H	UNC_THERMAL_THROTTLING_PROCHOT.CORE_3	Cycles that the PCU records that core 3 is a low power state due to the system asserting PROCHOT the entire processor has exceeded the thermal limit.	
84H	01H	UNC_TURBO_MODE.CORE_0	Uncore cycles that core 0 is operating in turbo mode.	
84H	02H	UNC_TURBO_MODE.CORE_1	Uncore cycles that core 1 is operating in turbo mode.	
84H	04H	UNC_TURBO_MODE.CORE_2	Uncore cycles that core 2 is operating in turbo mode.	
84H	08H	UNC_TURBO_MODE.CORE_3	Uncore cycles that core 3 is operating in turbo mode.	
85H	02H	UNC_CYCLES_UNHALTED_L3_FLL_ENABLE	Uncore cycles that at least one core is unhalted and all L3 ways are enabled.	
86H	01H	UNC_CYCLES_UNHALTED_L3_FLL_DISABLE	Uncore cycles that at least one core is unhalted and all L3 ways are disabled.	

## 19.11 PERFORMANCE MONITORING EVENTS FOR INTEL® XEON® PROCESSOR 5200, 5400 SERIES AND INTEL® CORE™ 2 EXTREME PROCESSORS QX 9000 SERIES

Processors based on the Enhanced Intel Core microarchitecture support the architectural and model-specific performance monitoring events listed in Table 19-1 and Table 19-25. In addition, they also support the following model-specific performance monitoring events listed in Table 19-23. Fixed counters support the architecture events defined in Table 19-24.

**Table 19-23. Performance Events for Processors Based on Enhanced Intel Core Microarchitecture**

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
COH	08H	INST_RETIRED.VM_HOST	Instruction retired while in VMX root operations.	
D2H	10H	RAT_STAALS.OTHER_SERIALI ZATION_STALLS	This event counts the number of stalls due to other RAT resource serialization not counted by Umask value 0FH.	

## 19.12 PERFORMANCE MONITORING EVENTS FOR INTEL® XEON® PROCESSOR 3000, 3200, 5100, 5300 SERIES AND INTEL® CORE™ 2 DUO PROCESSORS

Processors based on the Intel® Core™ microarchitecture support architectural and model-specific performance monitoring events.

Fixed-function performance counters are introduced first on processors based on Intel Core microarchitecture. Table 19-24 lists pre-defined performance events that can be counted using fixed-function performance counters.

**Table 19-24. Fixed-Function Performance Counter and Pre-defined Performance Events**

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
MSR_PERF_FIXED_CTR0/IA32_PERF_FIXED_CTR0	309H	Inst_Retired.Any	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers.
MSR_PERF_FIXED_CTR1/IA32_PERF_FIXED_CTR1	30AH	CPU_CLK_UNHALTED.CORE	This event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. This event is a component in many key event ratios.  The core frequency may change from time to time due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason this event may have a changing ratio with regards to time. When the core frequency is constant, this event can approximate elapsed time while the core was not in halt state.
MSR_PERF_FIXED_CTR2/IA32_PERF_FIXED_CTR2	30BH	CPU_CLK_UNHALTED.REF	This event counts the number of reference cycles when the core is not in a halt state and not in a TM stop-clock state. The core enters the halt state when it is running the HLT instruction or the MWAIT instruction.

**Table 19-24. Fixed-Function Performance Counter and Pre-defined Performance Events (Contd.)**

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
			<p>This event is not affected by core frequency changes (e.g., P states) but counts at the same frequency as the time stamp counter. This event can approximate elapsed time while the core was not in halt state and not in a TM stop-clock state.</p> <p>This event has a constant ratio with the CPU_CLK_UNHALTED.BUS event.</p>

Table 19-25 lists general-purpose model-specific performance monitoring events supported in processors based on Intel® Core™ microarchitecture. For convenience, Table 19-25 also includes architectural events and describes minor model-specific behavior where applicable. Software must use a general-purpose performance counter to count events listed in Table 19-25.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture**

Event Num	Umask Value	Event Name	Definition	Description and Comment
03H	02H	LOAD_BLOCK.STA	Loads blocked by a preceding store with unknown address.	<p>This event indicates that loads are blocked by preceding stores. A load is blocked when there is a preceding store to an address that is not yet calculated. The number of events is greater or equal to the number of load operations that were blocked.</p> <p>If the load and the store are always to different addresses, check why the memory disambiguation mechanism is not working. To avoid such blocks, increase the distance between the store and the following load so that the store address is known at the time the load is dispatched.</p>
03H	04H	LOAD_BLOCK.STD	Loads blocked by a preceding store with unknown data.	<p>This event indicates that loads are blocked by preceding stores. A load is blocked when there is a preceding store to the same address and the stored data value is not yet known. The number of events is greater or equal to the number of load operations that were blocked.</p> <p>To avoid such blocks, increase the distance between the store and the dependent load, so that the store data is known at the time the load is dispatched.</p>
03H	08H	LOAD_BLOCK.OVERLAP_STORE	Loads that partially overlap an earlier store, or 4-Kbyte aliased with a previous store.	<p>This event indicates that loads are blocked due to a variety of reasons. Some of the triggers for this event are when a load is blocked by a preceding store, in one of the following:</p> <ul style="list-style-type: none"> <li>▪ Some of the loaded byte locations are written by the preceding store and some are not.</li> <li>▪ The load is from bytes written by the preceding store, the store is aligned to its size and either:                             <ul style="list-style-type: none"> <li>▪ The load's data size is one or two bytes and it is not aligned to the store.</li> <li>▪ The load's data size is of four or eight bytes and the load is misaligned.</li> </ul> </li> </ul>

Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
				<ul style="list-style-type: none"> <li>The load is from bytes written by the preceding store, the store is misaligned and the load is not aligned on the beginning of the store.</li> <li>The load is split over an eight byte boundary (excluding 16-byte loads).</li> <li>The load and store have the same offset relative to the beginning of different 4-KByte pages. This case is also called 4-KByte aliasing.</li> <li>In all these cases the load is blocked until after the blocking store retires and the stored data is committed to the cache hierarchy.</li> </ul>
03H	10H	LOAD_BLOCK.UNTIL_RETIRE	Loads blocked until retirement.	This event indicates that load operations were blocked until retirement. The number of events is greater or equal to the number of load operations that were blocked. This includes mainly uncacheable loads and split loads (loads that cross the cache line boundary) but may include other cases where loads are blocked until retirement.
03H	20H	LOAD_BLOCK.L1D	Loads blocked by the L1 data cache.	This event indicates that loads are blocked due to one or more reasons. Some triggers for this event are: <ul style="list-style-type: none"> <li>The number of L1 data cache misses exceeds the maximum number of outstanding misses supported by the processor. This includes misses generated as result of demand fetches, software prefetches or hardware prefetches.</li> <li>Cache line split loads.</li> <li>Partial reads, such as reads to un-cacheable memory, I/O instructions and more.</li> <li>A locked load operation is in progress. The number of events is greater or equal to the number of load operations that were blocked.</li> </ul>
04H	01H	SB_DRAIN_CYCLES	Cycles while stores are blocked due to store buffer drain.	This event counts every cycle during which the store buffer is draining. This includes: <ul style="list-style-type: none"> <li>Serializing operations such as CPUID</li> <li>Synchronizing operations such as XCHG</li> <li>Interrupt acknowledgment</li> <li>Other conditions, such as cache flushing</li> </ul>
04H	02H	STORE_BLOCK.ORDER	Cycles while store is waiting for a preceding store to be globally observed.	This event counts the total duration, in number of cycles, which stores are waiting for a preceding stored cache line to be observed by other cores. This situation happens as a result of the strong store ordering behavior, as defined in "Memory Ordering," Chapter 8, <i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A</i> .  The stall may occur and be noticeable if there are many cases when a store either misses the L1 data cache or hits a cache line in the Shared state. If the store requires a bus transaction to read the cache line then the stall ends when snoop response for the bus transaction arrives.
04H	08H	STORE_BLOCK.SNOOP	A store is blocked due to a conflict with an external or internal snoop.	This event counts the number of cycles the store port was used for snooping the L1 data cache and a store was stalled by the snoop. The store is typically resubmitted one cycle later.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
06H	00H	SEGMENT_REG_LOADS	Number of segment register loads.	<p>This event counts the number of segment register load operations. Instructions that load new values into segment registers cause a penalty.</p> <p>This event indicates performance issues in 16-bit code. If this event occurs frequently, it may be useful to calculate the number of instructions retired per segment register load. If the resulting calculation is low (on average a small number of instructions are executed between segment register loads), then the code's segment register usage should be optimized.</p> <p>As a result of branch misprediction, this event is speculative and may include segment register loads that do not actually occur. However, most segment register loads are internally serialized and such speculative effects are minimized.</p>
07H	00H	SSE_PRE_EXEC.NTA	Streaming SIMD Extensions (SSE) Prefetch NTA instructions executed.	<p>This event counts the number of times the SSE instruction prefetchNTA is executed.</p> <p>This instruction prefetches the data to the L1 data cache.</p>
07H	01H	SSE_PRE_EXEC.L1	Streaming SIMD Extensions (SSE) PrefetchT0 instructions executed.	This event counts the number of times the SSE instruction prefetchT0 is executed. This instruction prefetches the data to the L1 data cache and L2 cache.
07H	02H	SSE_PRE_EXEC.L2	Streaming SIMD Extensions (SSE) PrefetchT1 and PrefetchT2 instructions executed.	This event counts the number of times the SSE instructions prefetchT1 and prefetchT2 are executed. These instructions prefetch the data to the L2 cache.
07H	03H	SSE_PRE_EXEC.STORES	Streaming SIMD Extensions (SSE) Weakly-ordered store instructions executed.	This event counts the number of times SSE non-temporal store instructions are executed.
08H	01H	DTLB_MISSES.ANY	Memory accesses that missed the DTLB.	<p>This event counts the number of Data Table Lookaside Buffer (DTLB) misses. The count includes misses detected as a result of speculative accesses.</p> <p>Typically a high count for this event indicates that the code accesses a large number of data pages.</p>
08H	02H	DTLB_MISSES.MISS_LD	DTLB misses due to load operations.	<p>This event counts the number of Data Table Lookaside Buffer (DTLB) misses due to load operations.</p> <p>This count includes misses detected as a result of speculative accesses.</p>
08H	04H	DTLB_MISSES.LO_MISS_LD	LO DTLB misses due to load operations.	<p>This event counts the number of level 0 Data Table Lookaside Buffer (DTLB0) misses due to load operations.</p> <p>This count includes misses detected as a result of speculative accesses. Loads that miss that DTLB0 and hit the DTLB1 can incur two-cycle penalty.</p>

Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
08H	08H	DTLB_MISSES. MISS_ST	TLB misses due to store operations.	This event counts the number of Data Table Lookaside Buffer (DTLB) misses due to store operations.  This count includes misses detected as a result of speculative accesses. Address translation for store operations is performed in the DTLB1.
09H	01H	MEMORY_ DISAMBIGUATION.RESET	Memory disambiguation reset cycles.	This event counts the number of cycles during which memory disambiguation misprediction occurs. As a result the execution pipeline is cleaned and execution of the mispredicted load instruction and all succeeding instructions restarts.  This event occurs when the data address accessed by a load instruction, collides infrequently with preceding stores, but usually there is no collision. It happens rarely, and may have a penalty of about 20 cycles.
09H	02H	MEMORY_DISAMBIGUATIO N.SUCCESS	Number of loads successfully disambiguated.	This event counts the number of load operations that were successfully disambiguated. Loads are preceded by a store with an unknown address, but they are not blocked.
0CH	01H	PAGE_WALKS. .COUNT	Number of page-walks executed.	This event counts the number of page-walks executed due to either a DTLB or ITLB miss.  The page walk duration, PAGE_WALKS.CYCLES, divided by number of page walks is the average duration of a page walk. The average can hint whether most of the page-walks are satisfied by the caches or cause an L2 cache miss.
0CH	02H	PAGE_WALKS. CYCLES	Duration of page-walks in core cycles.	This event counts the duration of page-walks in core cycles. The paging mode in use typically affects the duration of page walks.  Page walk duration divided by number of page walks is the average duration of page-walks. The average can hint at whether most of the page-walks are satisfied by the caches or cause an L2 cache miss.
10H	00H	FP_COMP_OPS _EXE	Floating point computational micro-ops executed.	This event counts the number of floating point computational micro-ops executed. Use IA32_PMC0 only.
11H	00H	FP_ASSIST	Floating point assists.	This event counts the number of floating point operations executed that required micro-code assist intervention. Assists are required in the following cases: <ul style="list-style-type: none"> <li>▪ Streaming SIMD Extensions (SSE) instructions:</li> <li>▪ Denormal input when the DAZ (Denormals Are Zeros) flag is off</li> <li>▪ Underflow result when the FTZ (Flush To Zero) flag is off</li> <li>▪ X87 instructions:</li> <li>▪ NaN or denormal are loaded to a register or used as input from memory</li> <li>▪ Division by 0</li> <li>▪ Underflow output</li> </ul> Use IA32_PMC1 only.
12H	00H	MUL	Multiply operations executed.	This event counts the number of multiply operations executed. This includes integer as well as floating point multiply operations. Use IA32_PMC1 only.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
13H	00H	DIV	Divide operations executed.	This event counts the number of divide operations executed. This includes integer divides, floating point divides and square-root operations executed. Use IA32_PMC1 only.
14H	00H	CYCLES_DIV_BUSY	Cycles the divider busy.	This event counts the number of cycles the divider is busy executing divide or square root operations. The divide can be integer, X87 or Streaming SIMD Extensions (SSE). The square root operation can be either X87 or SSE. Use IA32_PMC0 only.
18H	00H	IDLE_DURING_DIV	Cycles the divider is busy and all other execution units are idle.	This event counts the number of cycles the divider is busy (with a divide or a square root operation) and no other execution unit or load operation is in progress. Load operations are assumed to hit the L1 data cache. This event considers only micro-ops dispatched after the divider started operating. Use IA32_PMC0 only.
19H	00H	DELAYED_BYPASS.FP	Delayed bypass to FP operation.	This event counts the number of times floating point operations use data immediately after the data was generated by a non-floating point execution unit. Such cases result in one penalty cycle due to data bypass between the units. Use IA32_PMC1 only.
19H	01H	DELAYED_BYPASS.SIMD	Delayed bypass to SIMD operation.	This event counts the number of times SIMD operations use data immediately after the data was generated by a non-SIMD execution unit. Such cases result in one penalty cycle due to data bypass between the units. Use IA32_PMC1 only.
19H	02H	DELAYED_BYPASS.LOAD	Delayed bypass to load operation.	This event counts the number of delayed bypass penalty cycles that a load operation incurred. When load operations use data immediately after the data was generated by an integer execution unit, they may (pending on certain dynamic internal conditions) incur one penalty cycle due to delayed data bypass between the units. Use IA32_PMC1 only.
21H	See Table 18-61	L2_ADS.(Core)	Cycles L2 address bus is in use.	This event counts the number of cycles the L2 address bus is being used for accesses to the L2 cache or bus queue. It can count occurrences for this core or both cores.
23H	See Table 18-61	L2_DBUS_BUSY_RD.(Core)	Cycles the L2 transfers data to the core.	This event counts the number of cycles during which the L2 data bus is busy transferring data from the L2 cache to the core. It counts for all L1 cache misses (data and instruction) that hit the L2 cache. This event can count occurrences for this core or both cores.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Unmask Value	Event Name	Definition	Description and Comment
24H	Combined mask from Table 18-61 and Table 18-63	L2_LINES_IN. (Core, Prefetch)	L2 cache misses.	This event counts the number of cache lines allocated in the L2 cache. Cache lines are allocated in the L2 cache as a result of requests from the L1 data and instruction caches and the L2 hardware prefetchers to cache lines that are missing in the L2 cache.  This event can count occurrences for this core or both cores. It can also count demand requests and L2 hardware prefetch requests together or separately.
25H	See Table 18-61	L2_M_LINES_IN. (Core)	L2 cache line modifications.	This event counts whenever a modified cache line is written back from the L1 data cache to the L2 cache.  This event can count occurrences for this core or both cores.
26H	See Table 18-61 and Table 18-63	L2_LINES_OUT. (Core, Prefetch)	L2 cache lines evicted.	This event counts the number of L2 cache lines evicted.  This event can count occurrences for this core or both cores. It can also count evictions due to demand requests and L2 hardware prefetch requests together or separately.
27H	See Table 18-61 and Table 18-63	L2_M_LINES_OUT.(Core, Prefetch)	Modified lines evicted from the L2 cache.	This event counts the number of L2 modified cache lines evicted. These lines are written back to memory unless they also exist in a modified-state in one of the L1 data caches.  This event can count occurrences for this core or both cores. It can also count evictions due to demand requests and L2 hardware prefetch requests together or separately.
28H	Combined mask from Table 18-61 and Table 18-64	L2_IFETCH.(Core, Cache Line State)	L2 cacheable instruction fetch requests.	This event counts the number of instruction cache line requests from the IFU. It does not include fetch requests from uncacheable memory. It does not include ITLB miss accesses.  This event can count occurrences for this core or both cores. It can also count accesses to cache lines at different MESI states.
29H	Combined mask from Table 18-61, Table 18-63, and Table 18-64	L2_LD.(Core, Prefetch, Cache Line State)	L2 cache reads.	This event counts L2 cache read requests coming from the L1 data cache and L2 prefetchers.  The event can count occurrences: <ul style="list-style-type: none"> <li>▪ For this core or both cores.</li> <li>▪ Due to demand requests and L2 hardware prefetch requests together or separately.</li> <li>▪ Of accesses to cache lines at different MESI states.</li> </ul>
2AH	See Table 18-61 and Table 18-64	L2_ST.(Core, Cache Line State)	L2 store requests.	This event counts all store operations that miss the L1 data cache and request the data from the L2 cache.  The event can count occurrences for this core or both cores. It can also count accesses to cache lines at different MESI states.



**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
2BH	See Table 18-61 and Table 18-64	L2_LOCK.(Core, Cache Line State)	L2 locked accesses.	This event counts all locked accesses to cache lines that miss the L1 data cache.  The event can count occurrences for this core or both cores. It can also count accesses to cache lines at different MESI states.
2EH	See Table 18-61, Table 18-63, and Table 18-64	L2_RQSTS.(Core, Prefetch, Cache Line State)	L2 cache requests.	This event counts all completed L2 cache requests. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, instruction fetches, and all L2 hardware prefetch requests.  This event can count occurrences: <ul style="list-style-type: none"> <li>▪ For this core or both cores.</li> <li>▪ Due to demand requests and L2 hardware prefetch requests together, or separately.</li> <li>▪ Of accesses to cache lines at different MESI states.</li> </ul>
2EH	41H	L2_RQSTS.SELF.DEMAND.I_STATE	L2 cache demand requests from this core that missed the L2.	This event counts all completed L2 cache demand requests from this core that miss the L2 cache. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, and instruction fetches.  This is an architectural performance event.
2EH	4FH	L2_RQSTS.SELF.DEMAND.MESI	L2 cache demand requests from this core.	This event counts all completed L2 cache demand requests from this core. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, and instruction fetches.  This is an architectural performance event.
30H	See Table 18-61, Table 18-63, and Table 18-64	L2_REJECT_BUSQ.(Core, Prefetch, Cache Line State)	Rejected L2 cache requests.	This event indicates that a pending L2 cache request that requires a bus transaction is delayed from moving to the bus queue. Some of the reasons for this event are: <ul style="list-style-type: none"> <li>▪ The bus queue is full.</li> <li>▪ The bus queue already holds an entry for a cache line in the same set.</li> </ul> The number of events is greater or equal to the number of requests that were rejected. <ul style="list-style-type: none"> <li>▪ For this core or both cores.</li> <li>▪ Due to demand requests and L2 hardware prefetch requests together, or separately.</li> <li>▪ Of accesses to cache lines at different MESI states.</li> </ul>
32H	See Table 18-61	L2_NO_REQ.(Core)	Cycles no L2 cache requests are pending.	This event counts the number of cycles that no L2 cache requests were pending from a core. When using the BOTH_CORE modifier, the event counts only if none of the cores have a pending request. The event counts also when one core is halted and the other is not halted.  The event can count occurrences for this core or both cores.
3AH	00H	EIST_TRANS	Number of Enhanced Intel SpeedStep Technology (EIST) transitions.	This event counts the number of transitions that include a frequency change, either with or without voltage change. This includes Enhanced Intel SpeedStep Technology (EIST) and TM2 transitions.  The event is incremented only while the counting core is in C0 state. Since transitions to higher-numbered CxE states and TM2 transitions include a frequency change or voltage transition, the event is incremented accordingly.

Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
3BH	C0H	THERMAL_TRIP	Number of thermal trips.	This event counts the number of thermal trips. A thermal trip occurs whenever the processor temperature exceeds the thermal trip threshold temperature.  Following a thermal trip, the processor automatically reduces frequency and voltage. The processor checks the temperature every millisecond and returns to normal when the temperature falls below the thermal trip threshold temperature.
3CH	00H	CPU_CLK_UNHALTED.CORE_P	Core cycles when core is not halted.	This event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. This event is a component in many key event ratios.  The core frequency may change due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason, this event may have a changing ratio in regard to time.  When the core frequency is constant, this event can give approximate elapsed time while the core not in halt state. This is an architectural performance event.
3CH	01H	CPU_CLK_UNHALTED.BUS	Bus cycles when core is not halted.	This event counts the number of bus cycles while the core is not in the halt state. This event can give a measurement of the elapsed time while the core was not in the halt state. The core enters the halt state when it is running the HLT instruction.  The event also has a constant ratio with CPU_CLK_UNHALTED.REF event, which is the maximum bus to processor frequency ratio.  Non-halted bus cycles are a component in many key event ratios.
3CH	02H	CPU_CLK_UNHALTED.NO_OTHER	Bus cycles when core is active and the other is halted.	This event counts the number of bus cycles during which the core remains non-halted and the other core on the processor is halted.  This event can be used to determine the amount of parallelism exploited by an application or a system. Divide this event count by the bus frequency to determine the amount of time that only one core was in use.
40H	See Table 18-64	L1D_CACHE_LD.(Cache Line State)	L1 cacheable data reads.	This event counts the number of data reads from cacheable memory. Locked reads are not counted.
41H	See Table 18-64	L1D_CACHE_ST.(Cache Line State)	L1 cacheable data writes.	This event counts the number of data writes to cacheable memory. Locked writes are not counted.
42H	See Table 18-64	L1D_CACHE_LOCK.(Cache Line State)	L1 data cacheable locked reads.	This event counts the number of locked data reads from cacheable memory.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
42H	10H	L1D_CACHE_LOCK_DURATION	Duration of L1 data cacheable locked operation.	This event counts the number of cycles during which any cache line is locked by any locking instruction. Locking happens at retirement and therefore the event does not occur for instructions that are speculatively executed. Locking duration is shorter than locked instruction execution duration.
43H	01H	L1D_ALL_REF	All references to the L1 data cache.	This event counts all references to the L1 data cache, including all loads and stores with any memory types. The event counts memory accesses only when they are actually performed. For example, a load blocked by unknown store address and later performed is only counted once. The event includes non-cacheable accesses, such as I/O accesses.
43H	02H	L1D_ALL_CACHE_REF	L1 Data cacheable reads and writes.	This event counts the number of data reads and writes from cacheable memory, including locked operations. This event is a sum of: <ul style="list-style-type: none"> <li>▪ L1D_CACHE_LD.MESI</li> <li>▪ L1D_CACHE_ST.MESI</li> <li>▪ L1D_CACHE_LOCK.MESI</li> </ul>
45H	0FH	L1D_REPL	Cache lines allocated in the L1 data cache.	This event counts the number of lines brought into the L1 data cache.
46H	00H	L1D_M_REPL	Modified cache lines allocated in the L1 data cache.	This event counts the number of modified lines brought into the L1 data cache.
47H	00H	L1D_M_EVICT	Modified cache lines evicted from the L1 data cache.	This event counts the number of modified lines evicted from the L1 data cache, whether due to replacement or by snoop HITM intervention.
48H	00H	L1D_PEND_MISS	Total number of outstanding L1 data cache misses at any cycle.	This event counts the number of outstanding L1 data cache misses at any cycle. An L1 data cache miss is outstanding from the cycle on which the miss is determined until the first chunk of data is available. This event counts: <ul style="list-style-type: none"> <li>▪ All cacheable demand requests.</li> <li>▪ L1 data cache hardware prefetch requests.</li> <li>▪ Requests to write through memory.</li> <li>▪ Requests to write combine memory.</li> </ul> Uncacheable requests are not counted. The count of this event divided by the number of L1 data cache misses, L1D_REPL, is the average duration in core cycles of an L1 data cache miss.
49H	01H	L1D_SPLIT.LOADS	Cache line split loads from the L1 data cache.	This event counts the number of load operations that span two cache lines. Such load operations are also called split loads. Split load operations are executed at retirement.
49H	02H	L1D_SPLIT.STORES	Cache line split stores to the L1 data cache.	This event counts the number of store operations that span two cache lines.
4BH	00H	SSE_PRE_MISS.NTA	Streaming SIMD Extensions (SSE) Prefetch NTA instructions missing all cache levels.	This event counts the number of times the SSE instructions prefetchNTA were executed and missed all cache levels. Due to speculation an executed instruction might not retire. This instruction prefetches the data to the L1 data cache.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
4BH	01H	SSE_PRE_MISS.L1	Streaming SIMD Extensions (SSE) PrefetchT0 instructions missing all cache levels.	This event counts the number of times the SSE instructions prefetchT0 were executed and missed all cache levels. Due to speculation executed instruction might not retire. The prefetchT0 instruction prefetches data to the L2 cache and L1 data cache.
4BH	02H	SSE_PRE_MISS.L2	Streaming SIMD Extensions (SSE) PrefetchT1 and PrefetchT2 instructions missing all cache levels.	This event counts the number of times the SSE instructions prefetchT1 and prefetchT2 were executed and missed all cache levels. Due to speculation, an executed instruction might not retire. The prefetchT1 and PrefetchNT2 instructions prefetch data to the L2 cache.
4CH	00H	LOAD_HIT_PRE	Load operations conflicting with a software prefetch to the same address.	This event counts load operations sent to the L1 data cache while a previous Streaming SIMD Extensions (SSE) prefetch instruction to the same cache line has started prefetching but has not yet finished.
4EH	10H	L1D_PREFETCH_REQUESTS	L1 data cache prefetch requests.	This event counts the number of times the L1 data cache requested to prefetch a data cache line. Requests can be rejected when the L2 cache is busy and resubmitted later or lost. All requests are counted, including those that are rejected.
60H	See Table 18-61 and Table 18-62.	BUS_REQUEST_OUTSTANDING. (Core and Bus Agents)	Outstanding cacheable data read bus requests duration.	This event counts the number of pending full cache line read transactions on the bus occurring in each cycle. A read transaction is pending from the cycle it is sent on the bus until the full cache line is received by the processor. The event counts only full-line cacheable read requests from either the L1 data cache or the L2 prefetchers. It does not count Read for Ownership transactions, instruction byte fetch transactions, or any other bus transaction.
61H	See Table 18-62.	BUS_BNR_DRV. (Bus Agents)	Number of Bus Not Ready signals asserted.	This event counts the number of Bus Not Ready (BNR) signals that the processor asserts on the bus to suspend additional bus requests by other bus agents. A bus agent asserts the BNR signal when the number of data and snoop transactions is close to the maximum that the bus can handle. To obtain the number of bus cycles during which the BNR signal is asserted, multiply the event count by two. While this signal is asserted, new transactions cannot be submitted on the bus. As a result, transaction latency may have higher impact on program performance.
62H	See Table 18-62.	BUS_DRDY_CLOCKS. (Bus Agents)	Bus cycles when data is sent on the bus.	This event counts the number of bus cycles during which the DRDY (Data Ready) signal is asserted on the bus. The DRDY signal is asserted when data is sent on the bus. With the 'THIS_AGENT' mask this event counts the number of bus cycles during which this agent (the processor) writes data on the bus back to memory or to other bus agents. This includes all explicit and implicit data writebacks, as well as partial writes. With the 'ALL_AGENTS' mask, this event counts the number of bus cycles during which any bus agent sends data on the bus. This includes all data reads and writes on the bus.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
63H	See Table 18-61 and Table 18-62.	BUS_LOCK_CLOCKS.(Core and Bus Agents)	Bus cycles when a LOCK signal asserted.	This event counts the number of bus cycles, during which the LOCK signal is asserted on the bus. A LOCK signal is asserted when there is a locked memory access, due to: <ul style="list-style-type: none"> <li>▪ Uncacheable memory.</li> <li>▪ Locked operation that spans two cache lines.</li> <li>▪ Page-walk from an uncacheable page table.</li> </ul> Bus locks have a very high performance penalty and it is highly recommended to avoid such accesses.
64H	See Table 18-61.	BUS_DATA_RCV.(Core)	Bus cycles while processor receives data.	This event counts the number of bus cycles during which the processor is busy receiving data.
65H	See Table 18-61 and Table 18-62.	BUS_TRANS_BRD.(Core and Bus Agents)	Burst read bus transactions.	This event counts the number of burst read transactions including: <ul style="list-style-type: none"> <li>▪ L1 data cache read misses (and L1 data cache hardware prefetches).</li> <li>▪ L2 hardware prefetches by the DPL and L2 streamer.</li> <li>▪ IFU read misses of cacheable lines.</li> </ul> It does not include RFO transactions.
66H	See Table 18-61 and Table 18-62.	BUS_TRANS_RFO.(Core and Bus Agents)	RFO bus transactions.	This event counts the number of Read For Ownership (RFO) bus transactions, due to store operations that miss the L1 data cache and the L2 cache. It also counts RFO bus transactions due to locked operations.
67H	See Table 18-61 and Table 18-62.	BUS_TRANS_WB.(Core and Bus Agents)	Explicit writeback bus transactions.	This event counts all explicit writeback bus transactions due to dirty line evictions. It does not count implicit writebacks due to invalidation by a snoop request.
68H	See Table 18-61 and Table 18-62.	BUS_TRANS_IFETCH.(Core and Bus Agents)	Instruction-fetch bus transactions.	This event counts all instruction fetch full cache line bus transactions.
69H	See Table 18-61 and Table 18-62.	BUS_TRANS_INVALID.(Core and Bus Agents)	Invalidate bus transactions.	This event counts all invalidate transactions. Invalidate transactions are generated when: <ul style="list-style-type: none"> <li>▪ A store operation hits a shared line in the L2 cache.</li> <li>▪ A full cache line write misses the L2 cache or hits a shared line in the L2 cache.</li> </ul>
6AH	See Table 18-61 and Table 18-62.	BUS_TRANS_PWR.(Core and Bus Agents)	Partial write bus transaction.	This event counts partial write bus transactions.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
6BH	See Table 18-61 and Table 18-62.	BUS_TRANS_P.(Core and Bus Agents)	Partial bus transactions.	This event counts all (read and write) partial bus transactions.
6CH	See Table 18-61 and Table 18-62.	BUS_TRANS_IO.(Core and Bus Agents)	IO bus transactions.	This event counts the number of completed I/O bus transactions as a result of IN and OUT instructions. The count does not include memory mapped IO.
6DH	See Table 18-61 and Table 18-62.	BUS_TRANS_DEF.(Core and Bus Agents)	Deferred bus transactions.	This event counts the number of deferred transactions.
6EH	See Table 18-61 and Table 18-62.	BUS_TRANS_BURST.(Core and Bus Agents)	Burst (full cache-line) bus transactions.	This event counts burst (full cache line) transactions including: <ul style="list-style-type: none"> <li>▪ Burst reads.</li> <li>▪ RFOs.</li> <li>▪ Explicit writebacks.</li> <li>▪ Write combine lines.</li> </ul>
6FH	See Table 18-61 and Table 18-62.	BUS_TRANS_MEM.(Core and Bus Agents)	Memory bus transactions.	This event counts all memory bus transactions including: <ul style="list-style-type: none"> <li>▪ Burst transactions.</li> <li>▪ Partial reads and writes - invalidate transactions.</li> </ul> The BUS_TRANS_MEM count is the sum of BUS_TRANS_BURST, BUS_TRANS_P and BUS_TRANS_IVAL.
70H	See Table 18-61 and Table 18-62.	BUS_TRANS_ANY.(Core and Bus Agents)	All bus transactions.	This event counts all bus transactions. This includes: <ul style="list-style-type: none"> <li>▪ Memory transactions.</li> <li>▪ IO transactions (non memory-mapped).</li> <li>▪ Deferred transaction completion.</li> <li>▪ Other less frequent transactions, such as interrupts.</li> </ul>
77H	See Table 18-61 and Table 18-65.	EXT_SNOOP.(Bus Agents, Snoop Response)	External snoops.	This event counts the snoop responses to bus transactions. Responses can be counted separately by type and by bus agent.  With the 'THIS_AGENT' mask, the event counts snoop responses from this processor to bus transactions sent by this processor. With the 'ALL_AGENTS' mask the event counts all snoop responses seen on the bus.
78H	See Table 18-61 and Table 18-66.	CMP_SNOOP.(Core, Snoop Type)	L1 data cache snooped by other core.	This event counts the number of times the L1 data cache is snooped for a cache line that is needed by the other core in the same processor. The cache line is either missing in the L1 instruction or data caches of the other core, or is available for reading only and the other core wishes to write the cache line.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
				<p>The snoop operation may change the cache line state. If the other core issued a read request that hit this core in E state, typically the state changes to S state in this core. If the other core issued a read for ownership request (due a write miss or hit to S state) that hits this core’s cache line in E or S state, this typically results in invalidation of the cache line in this core. If the snoop hits a line in M state, the state is changed at a later opportunity.</p> <p>These snoops are performed through the L1 data cache store port. Therefore, frequent snoops may conflict with extensive stores to the L1 data cache, which may increase store latency and impact performance.</p>
7AH	See Table 18-62.	BUS_HIT_DRV. (Bus Agents)	HIT signal asserted.	This event counts the number of bus cycles during which the processor drives the HIT# pin to signal HIT snoop response.
7BH	See Table 18-62.	BUS_HITM_DRV. (Bus Agents)	HITM signal asserted.	This event counts the number of bus cycles during which the processor drives the HITM# pin to signal HITM snoop response.
7DH	See Table 18-61.	BUSQ_EMPTY. (Core)	Bus queue empty.	<p>This event counts the number of cycles during which the core did not have any pending transactions in the bus queue. It also counts when the core is halted and the other core is not halted.</p> <p>This event can count occurrences for this core or both cores.</p>
7EH	See Table 18-61 and Table 18-62.	SNOOP_STALL_DRV. (Core and Bus Agents)	Bus stalled for snoops.	<p>This event counts the number of times that the bus snoop stall signal is asserted. To obtain the number of bus cycles during which snoops on the bus are prohibited, multiply the event count by two.</p> <p>During the snoop stall cycles, no new bus transactions requiring a snoop response can be initiated on the bus. A bus agent asserts a snoop stall signal if it cannot response to a snoop request within three bus cycles.</p>
7FH	See Table 18-61.	BUS_IO_WAIT. (Core)	IO requests waiting in the bus queue.	<p>This event counts the number of core cycles during which IO requests wait in the bus queue. With the SELF modifier this event counts IO requests per core.</p> <p>With the BOTH_CORE modifier, this event increments by one for any cycle for which there is a request from either core.</p>
80H	00H	L1I_READS	Instruction fetches.	This event counts all instruction fetches, including uncacheable fetches that bypass the Instruction Fetch Unit (IFU).
81H	00H	L1I_MISSES	Instruction Fetch Unit misses.	<p>This event counts all instruction fetches that miss the Instruction Fetch Unit (IFU) or produce memory requests. This includes uncacheable fetches.</p> <p>An instruction fetch miss is counted only once and not once for every cycle it is outstanding.</p>
82H	02H	ITLB.SMALL_MISS	ITLB small page misses.	This event counts the number of instruction fetches from small pages that miss the ITLB.
82H	10H	ITLB.LARGE_MISS	ITLB large page misses.	This event counts the number of instruction fetches from large pages that miss the ITLB.

Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
82H	40H	ITLB.FLUSH	ITLB flushes.	This event counts the number of ITLB flushes. This usually happens upon CR3 or CR0 writes, which are executed by the operating system during process switches.
82H	12H	ITLB.MISSES	ITLB misses.	This event counts the number of instruction fetches from either small or large pages that miss the ITLB.
83H	02H	INST_QUEUE.FULL	Cycles during which the instruction queue is full.	This event counts the number of cycles during which the instruction queue is full. In this situation, the core front end stops fetching more instructions. This is an indication of very long stalls in the back-end pipeline stages.
86H	00H	CYCLES_L1L_MEM_STALLED	Cycles during which instruction fetches stalled.	This event counts the number of cycles for which an instruction fetch stalls, including stalls due to any of the following reasons: <ul style="list-style-type: none"> <li>▪ Instruction Fetch Unit cache misses.</li> <li>▪ Instruction TLB misses.</li> <li>▪ Instruction TLB faults.</li> </ul>
87H	00H	ILD_STALL	Instruction Length Decoder stall cycles due to a length changing prefix.	This event counts the number of cycles during which the instruction length decoder uses the slow length decoder. Usually, instruction length decoding is done in one cycle. When the slow decoder is used, instruction decoding requires 6 cycles.  The slow decoder is used in the following cases: <ul style="list-style-type: none"> <li>▪ Operand override prefix (66H) preceding an instruction with immediate data.</li> <li>▪ Address override prefix (67H) preceding an instruction with a modr/m in real, big real, 16-bit protected or 32-bit protected modes.</li> </ul> To avoid instruction length decoding stalls, generate code using imm8 or imm32 values instead of imm16 values. If you must use an imm16 value, store the value in a register using "mov reg, imm32" and use the register format of the instruction.
88H	00H	BR_INST_EXEC	Branch instructions executed.	This event counts all executed branches (not necessarily retired). This includes only instructions and not micro-op branches.  Frequent branching is not necessarily a major performance issue. However frequent branch mispredictions may be a problem.
89H	00H	BR_MISSP_EXEC	Mispredicted branch instructions executed.	This event counts the number of mispredicted branch instructions that were executed.
8AH	00H	BR_BAC_MISSP_EXEC	Branch instructions mispredicted at decoding.	This event counts the number of branch instructions that were mispredicted at decoding.
8BH	00H	BR_CND_EXEC	Conditional branch instructions executed.	This event counts the number of conditional branch instructions executed, but not necessarily retired.
8CH	00H	BR_CND_MISSP_EXEC	Mispredicted conditional branch instructions executed.	This event counts the number of mispredicted conditional branch instructions that were executed.
8DH	00H	BR_IND_EXEC	Indirect branch instructions executed.	This event counts the number of indirect branch instructions that were executed.



**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
8EH	00H	BR_IND_MISSP_EXEC	Mispredicted indirect branch instructions executed.	This event counts the number of mispredicted indirect branch instructions that were executed.
8FH	00H	BR_RET_EXEC	RET instructions executed.	This event counts the number of RET instructions that were executed.
90H	00H	BR_RET_MISSP_EXEC	Mispredicted RET instructions executed.	This event counts the number of mispredicted RET instructions that were executed.
91H	00H	BR_RET_BAC_MISSP_EXEC	RET instructions executed mispredicted at decoding.	This event counts the number of RET instructions that were executed and were mispredicted at decoding.
92H	00H	BR_CALL_EXEC	CALL instructions executed.	This event counts the number of CALL instructions executed.
93H	00H	BR_CALL_MISSP_EXEC	Mispredicted CALL instructions executed.	This event counts the number of mispredicted CALL instructions that were executed.
94H	00H	BR_IND_CALL_EXEC	Indirect CALL instructions executed.	This event counts the number of indirect CALL instructions that were executed.
97H	00H	BR_TKN_BUBBLE_1	Branch predicted taken with bubble 1.	The events BR_TKN_BUBBLE_1 and BR_TKN_BUBBLE_2 together count the number of times a taken branch prediction incurred a one-cycle penalty. The penalty incurs when: <ul style="list-style-type: none"> <li>▪ Too many taken branches are placed together. To avoid this, unroll loops and add a non-taken branch in the middle of the taken sequence.</li> <li>▪ The branch target is unaligned. To avoid this, align the branch target.</li> </ul>
98H	00H	BR_TKN_BUBBLE_2	Branch predicted taken with bubble 2.	The events BR_TKN_BUBBLE_1 and BR_TKN_BUBBLE_2 together count the number of times a taken branch prediction incurred a one-cycle penalty. The penalty incurs when: <ul style="list-style-type: none"> <li>▪ Too many taken branches are placed together. To avoid this, unroll loops and add a non-taken branch in the middle of the taken sequence.</li> <li>▪ The branch target is unaligned. To avoid this, align the branch target.</li> </ul>
A0H	00H	RS_UOPS_DISPATCHED	Micro-ops dispatched for execution.	This event counts the number of micro-ops dispatched for execution. Up to six micro-ops can be dispatched in each cycle.
A1H	01H	RS_UOPS_DISPATCHED.PORT0	Cycles micro-ops dispatched for execution on port 0.	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Issue Ports are described in <i>Intel® 64 and IA-32 Architectures Optimization Reference Manual</i> . Use IA32_PMC0 only.
A1H	02H	RS_UOPS_DISPATCHED.PORT1	Cycles micro-ops dispatched for execution on port 1.	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
A1H	04H	RS_UOPS_DISPATCHED.PORT2	Cycles micro-ops dispatched for execution on port 2.	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
A1H	08H	RS_UOPS_DISPATCHED.PORT3	Cycles micro-ops dispatched for execution on port 3.	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
A1H	10H	RS_UOPS_DISPATCHED.PORT4	Cycles micro-ops dispatched for execution on port 4.	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
A1H	20H	RS_UOPS_DISPATCHED.PORT5	Cycles micro-ops dispatched for execution on port 5.	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
AAH	01H	MACRO_INSTS_DECODED	Instructions decoded.	This event counts the number of instructions decoded (but not necessarily executed or retired).
AAH	08H	MACRO_INSTS_CISC_DECODED	CISC Instructions decoded.	This event counts the number of complex instructions decoded. Complex instructions usually have more than four micro-ops. Only one complex instruction can be decoded at a time.
ABH	01H	ESP.SYNCH	ESP register content synchronization.	This event counts the number of times that the ESP register is explicitly used in the address expression of a load or store operation, after it is implicitly used, for example by a push or a pop instruction.  ESP synch micro-op uses resources from the rename pipe-stage and up to retirement. The expected ratio of this event divided by the number of ESP implicit changes is 0.2. If the ratio is higher, consider rearranging your code to avoid ESP synchronization events.
ABH	02H	ESP.ADDITIONS	ESP register automatic additions.	This event counts the number of ESP additions performed automatically by the decoder. A high count of this event is good, since each automatic addition performed by the decoder saves a micro-op from the execution units.  To maximize the number of ESP additions performed automatically by the decoder, choose instructions that implicitly use the ESP, such as PUSH, POP, CALL, and RET instructions whenever possible.
B0H	00H	SIMD_UOPS_EXEC	SIMD micro-ops executed (excluding stores).	This event counts all the SIMD micro-ops executed. It does not count MOVQ and MOVD stores from register to memory.
B1H	00H	SIMD_SAT_UOP_EXEC	SIMD saturated arithmetic micro-ops executed.	This event counts the number of SIMD saturated arithmetic micro-ops executed.
B3H	01H	SIMD_UOP_TYPE_EXEC.MUL	SIMD packed multiply micro-ops executed.	This event counts the number of SIMD packed multiply micro-ops executed.
B3H	02H	SIMD_UOP_TYPE_EXEC.SHIFT	SIMD packed shift micro-ops executed.	This event counts the number of SIMD packed shift micro-ops executed.
B3H	04H	SIMD_UOP_TYPE_EXEC.PACK	SIMD pack micro-ops executed.	This event counts the number of SIMD pack micro-ops executed.
B3H	08H	SIMD_UOP_TYPE_EXEC.UNPACK	SIMD unpack micro-ops executed.	This event counts the number of SIMD unpack micro-ops executed.
B3H	10H	SIMD_UOP_TYPE_EXEC.LOGICAL	SIMD packed logical micro-ops executed.	This event counts the number of SIMD packed logical micro-ops executed.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
B3H	20H	SIMD_UOP_TYPE_EXEC.ARITHMETIC	SIMD packed arithmetic micro-ops executed.	This event counts the number of SIMD packed arithmetic micro-ops executed.
COH	00H	INST_RETIRED.ANY_P	Instructions retired.	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers. INST_RETIRED.ANY_P is an architectural performance event.
COH	01H	INST_RETIRED.LOADS	Instructions retired, which contain a load.	This event counts the number of instructions retired that contain a load operation.
COH	02H	INST_RETIRED.STORES	Instructions retired, which contain a store.	This event counts the number of instructions retired that contain a store operation.
COH	04H	INST_RETIRED.OTHER	Instructions retired, with no load or store operation.	This event counts the number of instructions retired that do not contain a load or a store operation.
C1H	01H	X87_OPS_RETIRED.FXCH	FXCH instructions retired.	This event counts the number of FXCH instructions retired. Modern compilers generate more efficient code and are less likely to use this instruction. If you obtain a high count for this event consider recompiling the code.
C1H	FEH	X87_OPS_RETIRED.ANY	Retired floating-point computational operations (precise event).	<p>This event counts the number of floating-point computational operations retired. It counts:</p> <ul style="list-style-type: none"> <li>▪ Floating point computational operations executed by the assist handler.</li> <li>▪ Sub-operations of complex floating-point instructions like transcendental instructions.</li> </ul> <p>This event does not count:</p> <ul style="list-style-type: none"> <li>▪ Floating-point computational operations that cause traps or assists.</li> <li>▪ Floating-point loads and stores.</li> </ul> <p>When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.</p>
C2H	01H	UOPS_RETIRED.LD_IND_BR	Fused load+op or load+indirect branch retired.	<p>This event counts the number of retired micro-ops that fused a load with another operation. This includes:</p> <ul style="list-style-type: none"> <li>▪ Fusion of a load and an arithmetic operation, such as with the following instruction: ADD EAX, [EBX] where the content of the memory location specified by EBX register is loaded, added to EAX register, and the result is stored in EAX.</li> <li>▪ Fusion of a load and a branch in an indirect branch operation, such as with the following instructions: <ul style="list-style-type: none"> <li>▪ JMP [RDI+200]</li> <li>▪ RET</li> </ul> </li> <li>▪ Fusion decreases the number of micro-ops in the processor pipeline. A high value for this event count indicates that the code is using the processor resources effectively.</li> </ul>

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
C2H	02H	UOPS_RETIREDD. STD_STA	Fused store address + data retired.	This event counts the number of store address calculations that are fused with store data emission into one micro-op. Traditionally, each store operation required two micro-ops. This event counts fusion of retired micro-ops only. Fusion decreases the number of micro-ops in the processor pipeline. A high value for this event count indicates that the code is using the processor resources effectively.
C2H	04H	UOPS_RETIREDD. MACRO_FUSION	Retired instruction pairs fused into one micro-op.	This event counts the number of times CMP or TEST instructions were fused with a conditional branch instruction into one micro-op. It counts fusion by retired micro-ops only. Fusion decreases the number of micro-ops in the processor pipeline. A high value for this event count indicates that the code uses the processor resources more effectively.
C2H	07H	UOPS_RETIREDD. FUSED	Fused micro-ops retired.	This event counts the total number of retired fused micro-ops. The counts include the following fusion types: <ul style="list-style-type: none"> <li>▪ Fusion of load operation with an arithmetic operation or with an indirect branch (counted by event UOPS_RETIREDD.LD_IND_BR)</li> <li>▪ Fusion of store address and data (counted by event UOPS_RETIREDD.STD_STA)</li> <li>▪ Fusion of CMP or TEST instruction with a conditional branch instruction (counted by event UOPS_RETIREDD.MACRO_FUSION)</li> </ul> Fusion decreases the number of micro-ops in the processor pipeline. A high value for this event count indicates that the code is using the processor resources effectively.
C2H	08H	UOPS_RETIREDD. NON_FUSED	Non-fused micro-ops retired.	This event counts the number of micro-ops retired that were not fused.
C2H	0FH	UOPS_RETIREDD. ANY	Micro-ops retired.	This event counts the number of micro-ops retired. The processor decodes complex macro instructions into a sequence of simpler micro-ops. Most instructions are composed of one or two micro-ops. Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists. In some cases micro-op sequences are fused or whole instructions are fused into one micro-op. See other UOPS_RETIREDD events for differentiating retired fused and non-fused micro-ops.
C3H	01H	MACHINE_ NUKES.SMC	Self-Modifying Code detected.	This event counts the number of times that a program writes to a code section. Self-modifying code causes a severe penalty in all Intel 64 and IA-32 processors.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
C3H	04H	MACHINE_NUKES.MEM_ORDER	Execution pipeline restart due to memory ordering conflict or memory disambiguation misprediction.	This event counts the number of times the pipeline is restarted due to either multi-threaded memory ordering conflicts or memory disambiguation misprediction.  A multi-threaded memory ordering conflict occurs when a store, which is executed in another core, hits a load that is executed out of order in this core but not yet retired. As a result, the load needs to be restarted to satisfy the memory ordering model.  See Chapter 8, "Multiple-Processor Management" in the <i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A</i> .  To count memory disambiguation mispredictions, use the event MEMORY_DISAMBIGUATION.RESET.
C4H	00H	BR_INST_RETIRED.ANY	Retired branch instructions.	This event counts the number of branch instructions retired. This is an architectural performance event.
C4H	01H	BR_INST_RETIRED.PRED_NOT_TAKEN	Retired branch instructions that were predicted not-taken.	This event counts the number of branch instructions retired that were correctly predicted to be not-taken.
C4H	02H	BR_INST_RETIRED.MISPRED_NOT_TAKEN	Retired branch instructions that were mispredicted not-taken.	This event counts the number of branch instructions retired that were mispredicted and not-taken.
C4H	04H	BR_INST_RETIRED.PRED_TAKEN	Retired branch instructions that were predicted taken.	This event counts the number of branch instructions retired that were correctly predicted to be taken.
C4H	08H	BR_INST_RETIRED.MISPRED_TAKEN	Retired branch instructions that were mispredicted taken.	This event counts the number of branch instructions retired that were mispredicted and taken.
C4H	0CH	BR_INST_RETIRED.TAKEN	Retired taken branch instructions.	This event counts the number of branches retired that were taken.
C5H	00H	BR_INST_RETIRED.MISPRED	Retired mispredicted branch instructions. (precise event)	This event counts the number of retired branch instructions that were mispredicted by the processor. A branch misprediction occurs when the processor predicts that the branch would be taken, but it is not, or vice-versa.  This is an architectural performance event.
C6H	01H	CYCLES_INT_MASKED	Cycles during which interrupts are disabled.	This event counts the number of cycles during which interrupts are disabled.
C6H	02H	CYCLES_INT_PENDING_AND_MASKED	Cycles during which interrupts are pending and disabled.	This event counts the number of cycles during which there are pending interrupts but interrupts are disabled.
C7H	01H	SIMD_INST_RETIRED.PACKED_SINGLE	Retired SSE packed-single instructions.	This event counts the number of SSE packed-single instructions retired.
C7H	02H	SIMD_INST_RETIRED.SCALAR_SINGLE	Retired SSE scalar-single instructions.	This event counts the number of SSE scalar-single instructions retired.
C7H	04H	SIMD_INST_RETIRED.PACKED_DOUBLE	Retired SSE2 packed-double instructions.	This event counts the number of SSE2 packed-double instructions retired.
C7H	08H	SIMD_INST_RETIRED.SCALAR_DOUBLE	Retired SSE2 scalar-double instructions.	This event counts the number of SSE2 scalar-double instructions retired.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
C7H	10H	SIMD_INST_RETIRE.D.VECTOR	Retired SSE2 vector integer instructions.	This event counts the number of SSE2 vector integer instructions retired.
C7H	1FH	SIMD_INST_RETIRE.ANY	Retired Streaming SIMD instructions (precise event).	This event counts the overall number of retired SIMD instructions that use XMM registers. To count each type of SIMD instruction separately, use the following events: <ul style="list-style-type: none"> <li>▪ SIMD_INST_RETIRE.PACKED_SINGLE</li> <li>▪ SIMD_INST_RETIRE.SCALAR_SINGLE</li> <li>▪ SIMD_INST_RETIRE.PACKED_DOUBLE</li> <li>▪ SIMD_INST_RETIRE.SCALAR_DOUBLE</li> <li>▪ and SIMD_INST_RETIRE.VECTOR</li> </ul> When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.
C8H	00H	HW_INT_RCV	Hardware interrupts received.	This event counts the number of hardware interrupts received by the processor.
C9H	00H	ITLB_MISS_RETIRE	Retired instructions that missed the ITLB.	This event counts the number of retired instructions that missed the ITLB when they were fetched.
CAH	01H	SIMD_COMP_INST_RETIRE.PACKED_SINGLE	Retired computational SSE packed-single instructions.	This event counts the number of computational SSE packed-single instructions retired. Computational instructions perform arithmetic computations (for example: add, multiply and divide). Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	02H	SIMD_COMP_INST_RETIRE.SCALAR_SINGLE	Retired computational SSE scalar-single instructions.	This event counts the number of computational SSE scalar-single instructions retired. Computational instructions perform arithmetic computations (for example: add, multiply and divide). Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	04H	SIMD_COMP_INST_RETIRE.PACKED_DOUBLE	Retired computational SSE2 packed-double instructions.	This event counts the number of computational SSE2 packed-double instructions retired. Computational instructions perform arithmetic computations (for example: add, multiply and divide). Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	08H	SIMD_COMP_INST_RETIRE.D.SCALAR_DOUBLE	Retired computational SSE2 scalar-double instructions.	This event counts the number of computational SSE2 scalar-double instructions retired. Computational instructions perform arithmetic computations (for example: add, multiply and divide). Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
CBH	01H	MEM_LOAD_RETIREDD.L1D_MISS	Retired loads that miss the L1 data cache (precise event).	<p>This event counts the number of retired load operations that missed the L1 data cache. This includes loads from cache lines that are currently being fetched, due to a previous L1 data cache miss to the same cache line.</p> <p>This event counts loads from cacheable memory only. The event does not count loads by software prefetches.</p> <p>When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.</p> <p>Use IA32_PMC0 only.</p>
CBH	02H	MEM_LOAD_RETIREDD.L1D_LINE_MISS	L1 data cache line missed by retired loads (precise event).	<p>This event counts the number of load operations that miss the L1 data cache and send a request to the L2 cache to fetch the missing cache line. That is the missing cache line fetching has not yet started.</p> <p>The event count is equal to the number of cache lines fetched from the L2 cache by retired loads.</p> <p>This event counts loads from cacheable memory only. The event does not count loads by software prefetches.</p> <p>The event might not be counted if the load is blocked (see LOAD_BLOCK events).</p> <p>When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.</p> <p>Use IA32_PMC0 only.</p>
CBH	04H	MEM_LOAD_RETIREDD.L2_MISS	Retired loads that miss the L2 cache (precise event).	<p>This event counts the number of retired load operations that missed the L2 cache.</p> <p>This event counts loads from cacheable memory only. It does not count loads by software prefetches.</p> <p>When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.</p> <p>Use IA32_PMC0 only.</p>

Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
CBH	08H	MEM_LOAD_RETIREDD.L2_LINE_MISS	L2 cache line missed by retired loads (precise event).	<p>This event counts the number of load operations that miss the L2 cache and result in a bus request to fetch the missing cache line. That is the missing cache line fetching has not yet started.</p> <p>This event count is equal to the number of cache lines fetched from memory by retired loads.</p> <p>This event counts loads from cacheable memory only. The event does not count loads by software prefetches.</p> <p>The event might not be counted if the load is blocked (see LOAD_BLOCK events).</p> <p>When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.</p> <p>Use IA32_PMC0 only.</p>
CBH	10H	MEM_LOAD_RETIREDD.DTLB_MISS	Retired loads that miss the DTLB (precise event).	<p>This event counts the number of retired loads that missed the DTLB. The DTLB miss is not counted if the load operation causes a fault.</p> <p>This event counts loads from cacheable memory only. The event does not count loads by software prefetches.</p> <p>When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.</p> <p>Use IA32_PMC0 only.</p>
CCH	01H	FP_MMX_TRANS_TO_MMX	Transitions from Floating Point to MMX Instructions.	This event counts the first MMX instructions following a floating-point instruction. Use this event to estimate the penalties for the transitions between floating-point and MMX states.
CCH	02H	FP_MMX_TRANS_TO_FP	Transitions from MMX Instructions to Floating Point Instructions.	This event counts the first floating-point instructions following any MMX instruction. Use this event to estimate the penalties for the transitions between floating-point and MMX states.
CDH	00H	SIMD_ASSIST	SIMD assists invoked.	This event counts the number of SIMD assists invoked. SIMD assists are invoked when an EMMS instruction is executed, changing the MMX state in the floating point stack.
CEH	00H	SIMD_INSTR_RETIREDD	SIMD Instructions retired.	This event counts the number of retired SIMD instructions that use MMX registers.
CFH	00H	SIMD_SAT_INSTR_RETIREDD	Saturated arithmetic instructions retired.	This event counts the number of saturated arithmetic SIMD instructions that retired.
D2H	01H	RAT_STALLS.ROB_READ_PORT	ROB read port stalls cycles.	<p>This event counts the number of cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the out-of-order pipeline.</p> <p>Note that, at this stage in the pipeline, additional stalls may occur at the same cycle and prevent the stalled micro-ops from entering the pipe. In such a case, micro-ops retry entering the execution pipe in the next cycle and the ROB-read-port stall is counted again.</p>



**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
D2H	02H	RAT_STALLS.PARTIAL_CYCLES	Partial register stall cycles.	This event counts the number of cycles instruction execution latency became longer than the defined latency because the instruction uses a register that was partially written by previous instructions.
D2H	04H	RAT_STALLS.FLAGS	Flag stall cycles.	This event counts the number of cycles during which execution stalled due to several reasons, one of which is a partial flag register stall. A partial register stall may occur when two conditions are met: <ul style="list-style-type: none"> <li>▪ An instruction modifies some, but not all, of the flags in the flag register.</li> <li>▪ The next instruction, which depends on flags, depends on flags that were not modified by this instruction.</li> </ul>
D2H	08H	RAT_STALLS.FPSW	FPU status word stall.	This event indicates that the FPU status word (FPSW) is written. To obtain the number of times the FPSW is written divide the event count by 2. The FPSW is written by instructions with long latency; a small count may indicate a high penalty.
D2H	0FH	RAT_STALLS.ANY	All RAT stall cycles.	This event counts the number of stall cycles due to conditions described by: <ul style="list-style-type: none"> <li>▪ RAT_STALLS.ROB_READ_PORT</li> <li>▪ RAT_STALLS.PARTIAL</li> <li>▪ RAT_STALLS.FLAGS</li> <li>▪ RAT_STALLS.FPSW.</li> </ul>
D4H	01H	SEG_RENAME_STALLS.ES	Segment rename stalls - ES.	This event counts the number of stalls due to the lack of renaming resources for the ES segment register. If a segment is renamed, but not retired and a second update to the same segment occurs, a stall occurs in the front end of the pipeline until the renamed segment retires.
D4H	02H	SEG_RENAME_STALLS.DS	Segment rename stalls - DS.	This event counts the number of stalls due to the lack of renaming resources for the DS segment register. If a segment is renamed, but not retired and a second update to the same segment occurs, a stall occurs in the front end of the pipeline until the renamed segment retires.
D4H	04H	SEG_RENAME_STALLS.FS	Segment rename stalls - FS.	This event counts the number of stalls due to the lack of renaming resources for the FS segment register. If a segment is renamed, but not retired and a second update to the same segment occurs, a stall occurs in the front end of the pipeline until the renamed segment retires.
D4H	08H	SEG_RENAME_STALLS.GS	Segment rename stalls - GS.	This event counts the number of stalls due to the lack of renaming resources for the GS segment register. If a segment is renamed, but not retired and a second update to the same segment occurs, a stall occurs in the front end of the pipeline until the renamed segment retires.
D4H	0FH	SEG_RENAME_STALLS.ANY	Any (ES/DS/FS/GS) segment rename stall.	This event counts the number of stalls due to the lack of renaming resources for the ES, DS, FS, and GS segment registers. If a segment is renamed but not retired and a second update to the same segment occurs, a stall occurs in the front end of the pipeline until the renamed segment retires.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
D5H	01H	SEG_REG_RENAMES.ES	Segment renames - ES.	This event counts the number of times the ES segment register is renamed.
D5H	02H	SEG_REG_RENAMES.DS	Segment renames - DS.	This event counts the number of times the DS segment register is renamed.
D5H	04H	SEG_REG_RENAMES.FS	Segment renames - FS.	This event counts the number of times the FS segment register is renamed.
D5H	08H	SEG_REG_RENAMES.GS	Segment renames - GS.	This event counts the number of times the GS segment register is renamed.
D5H	0FH	SEG_REG_RENAMES.ANY	Any (ES/DS/FS/GS) segment rename.	This event counts the number of times any of the four segment registers (ES/DS/FS/GS) is renamed.
DCH	01H	RESOURCE_STALLS.ROB_FULL	Cycles during which the ROB full.	This event counts the number of cycles when the number of instructions in the pipeline waiting for retirement reaches the limit the processor can handle.  A high count for this event indicates that there are long latency operations in the pipe (possibly load and store operations that miss the L2 cache, and other instructions that depend on these cannot execute until the former instructions complete execution). In this situation new instructions cannot enter the pipe and start execution.
DCH	02H	RESOURCE_STALLS.RS_FULL	Cycles during which the RS full.	This event counts the number of cycles when the number of instructions in the pipeline waiting for execution reaches the limit the processor can handle.  A high count of this event indicates that there are long latency operations in the pipe (possibly load and store operations that miss the L2 cache, and other instructions that depend on these cannot execute until the former instructions complete execution). In this situation new instructions cannot enter the pipe and start execution.
DCH	04	RESOURCE_STALLS.LD_ST	Cycles during which the pipeline has exceeded load or store limit or waiting to commit all stores.	This event counts the number of cycles while resource-related stalls occur due to: <ul style="list-style-type: none"> <li>▪ The number of load instructions in the pipeline reached the limit the processor can handle. The stall ends when a loading instruction retires.</li> <li>▪ The number of store instructions in the pipeline reached the limit the processor can handle. The stall ends when a storing instruction commits its data to the cache or memory.</li> <li>▪ There is an instruction in the pipe that can be executed only when all previous stores complete and their data is committed in the caches or memory. For example, the SFENCE and MFENCE instructions require this behavior.</li> </ul>
DCH	08H	RESOURCE_STALLS.FPCW	Cycles stalled due to FPU control word write.	This event counts the number of cycles while execution was stalled due to writing the floating-point unit (FPU) control word.
DCH	10H	RESOURCE_STALLS.BR_MISS_CLEAR	Cycles stalled due to branch misprediction.	This event counts the number of cycles after a branch misprediction is detected at execution until the branch and all older micro-ops retire. During this time new micro-ops cannot enter the out-of-order pipeline.

**Table 19-25. Performance Events in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Event Num	Umask Value	Event Name	Definition	Description and Comment
DCH	1FH	RESOURCE_STALLS.ANY	Resource related stalls.	This event counts the number of cycles while resource-related stalls occurs for any conditions described by the following events: <ul style="list-style-type: none"> <li>▪ RESOURCE_STALLS.ROB_FULL</li> <li>▪ RESOURCE_STALLS.RS_FULL</li> <li>▪ RESOURCE_STALLS.ID_ST</li> <li>▪ RESOURCE_STALLS.FPCW</li> <li>▪ RESOURCE_STALLS.BR_MISS_CLEAR</li> </ul>
E0H	00H	BR_INST_DECODED	Branch instructions decoded.	This event counts the number of branch instructions decoded.
E4H	00H	BOGUS_BR	Bogus branches.	This event counts the number of byte sequences that were mistakenly detected as taken branch instructions. This results in a BACLEAR event. This occurs mainly after task switches.
E6H	00H	BACLEAR_S	BACLEAR_S asserted.	This event counts the number of times the front end is re-steered, mainly when the BPU cannot provide a correct prediction and this is corrected by other branch handling mechanisms at the front end. This can occur if the code has many branches such that they cannot be consumed by the BPU. Each BACLEAR asserted costs approximately 7 cycles of instruction fetch. The effect on total execution time depends on the surrounding code.
F0H	00H	PREF_RQSTS_UP	Upward prefetches issued from DPL.	This event counts the number of upward prefetches issued from the Data Prefetch Logic (DPL) to the L2 cache. A prefetch request issued to the L2 cache cannot be cancelled and the requested cache line is fetched to the L2 cache.
F8H	00H	PREF_RQSTS_DN	Downward prefetches issued from DPL.	This event counts the number of downward prefetches issued from the Data Prefetch Logic (DPL) to the L2 cache. A prefetch request issued to the L2 cache cannot be cancelled and the requested cache line is fetched to the L2 cache.

## 19.13 PERFORMANCE MONITORING EVENTS FOR PROCESSORS BASED ON THE GOLDMONT PLUS MICROARCHITECTURE

Intel Atom processors based on the Goldmont Plus microarchitecture support the architectural performance monitoring events listed in Table 19-1 and fixed-function performance events using a fixed counter. They also support the following performance monitoring events listed in Table 19-27. These events apply to processors with CPUID signature of 06\_7AH. In addition, processors based on the Goldmont Plus microarchitecture also support the events listed in Table 19-27 (see Section 19.14, “Performance Monitoring Events for Processors Based on the Goldmont Microarchitecture”). For an event listed in Table 19-27 that also appears in the model-specific tables of prior generations, Table 19-27 supersedes prior generation tables.

Performance monitoring event descriptions may refer to terminology described in Section B.2, “Intel® Xeon® processor 5500 Series,” in Appendix B of the *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

In Goldmont Plus microarchitecture, performance monitoring events that support Processor Event Based Sampling (PEBS) and PEBS records that contain processor state information that are associated with at-retirement tagging are marked by “Precise Event”.

**Table 19-26. Performance Events for the Goldmont Plus Microarchitecture**

Event Num.	Umask Value	Event Name	Description	Comment
00H	01H	INST_RETIRED.ANY	Counts the number of instructions that retire execution. For instructions that consist of multiple uops, this event counts the retirement of the last uop of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers. This event uses fixed counter 0. You cannot collect a PEBS record for this event.	Fixed Event, Precise Event, Not Reduced Skid
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED_4K	Counts page walks completed due to demand data loads (including SW prefetches) whose address translations missed in all TLB levels and were mapped to 4K pages. The page walks can end with or without a page fault.	
08H	04H	DTLB_LOAD_MISSES.WALK_COMPLETED_2M_4M	Counts page walks completed due to demand data loads (including SW prefetches) whose address translations missed in all TLB levels and were mapped to 2M or 4M pages. The page walks can end with or without a page fault.	
08H	08H	DTLB_LOAD_MISSES.WALK_COMPLETED_1GB	Counts page walks completed due to demand data loads (including SW prefetches) whose address translations missed in all TLB levels and were mapped to 1GB pages. The page walks can end with or without a page fault.	
08H	10H	DTLB_LOAD_MISSES.WALK_PENDING	Counts once per cycle for each page walk occurring due to a load (demand data loads or SW prefetches). Includes cycles spent traversing the Extended Page Table (EPT). Average cycles per walk can be calculated by dividing by the number of walks.	
49H	02H	DTLB_STORE_MISSES.WALK_COMPLETED_4K	Counts page walks completed due to demand data stores whose address translations missed in the TLB and were mapped to 4K pages. The page walks can end with or without a page fault.	
49H	04H	DTLB_STORE_MISSES.WALK_COMPLETED_2M_4M	Counts page walks completed due to demand data stores whose address translations missed in the TLB and were mapped to 2M or 4M pages. The page walks can end with or without a page fault.	
49H	08H	DTLB_STORE_MISSES.WALK_COMPLETED_1GB	Counts page walks completed due to demand data stores whose address translations missed in the TLB and were mapped to 1GB pages. The page walks can end with or without a page fault.	
49H	10H	DTLB_STORE_MISSES.WALK_PENDING	Counts once per cycle for each page walk occurring due to a demand data store. Includes cycles spent traversing the Extended Page Table (EPT). Average cycles per walk can be calculated by dividing by the number of walks.	

**Table 19-26. Performance Events for the Goldmont Plus Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Name	Description	Comment
4FH	10H	EPT.WALK_PENDING	Counts once per cycle for each page walk only while traversing the Extended Page Table (EPT), and does not count during the rest of the translation. The EPT is used for translating Guest-Physical Addresses to Physical Addresses for Virtual Machine Monitors (VMMs). Average cycles per walk can be calculated by dividing the count by number of walks.	
85H	02H	ITLB_MISSES.WALK_COMPLETED_4K	Counts page walks completed due to instruction fetches whose address translations missed in the TLB and were mapped to 4K pages. The page walks can end with or without a page fault.	
85H	04H	ITLB_MISSES.WALK_COMPLETED_2M_4M	Counts page walks completed due to instruction fetches whose address translations missed in the TLB and were mapped to 2M or 4M pages. The page walks can end with or without a page fault.	
85H	08H	ITLB_MISSES.WALK_COMPLETED_1GB	Counts page walks completed due to instruction fetches whose address translations missed in the TLB and were mapped to 1GB pages. The page walks can end with or without a page fault.	
85H	10H	ITLB_MISSES.WALK_PENDING	Counts once per cycle for each page walk occurring due to an instruction fetch. Includes cycles spent traversing the Extended Page Table (EPT). Average cycles per walk can be calculated by dividing by the number of walks.	
BDH	20H	TLB_FLUSHES.STLB_ANY	Counts STLB flushes. The TLBs are flushed on instructions like INVLPG and MOV to CR3.	
C3H	20H	MACHINE_CLEARS.PAGE_FAULT	Counts the number of times that the machines clears due to a page fault. Covers both I-side and D-side (Loads/Stores) page faults. A page fault occurs when either page is not present, or an access violation.	

## 19.14 PERFORMANCE MONITORING EVENTS FOR PROCESSORS BASED ON THE GOLDMONT MICROARCHITECTURE

Intel Atom processors based on the Goldmont microarchitecture support the architectural performance monitoring events listed in Table 19-1 and fixed-function performance events using a fixed counter. In addition, they also support the following model-specific performance monitoring events listed in Table 19-27. These events apply to processors with CPUID signatures of 06\_5CH, 06\_5FH, and 06\_7AH.

Performance monitoring event descriptions may refer to terminology described in Section B.2, “Intel® Xeon® processor 5500 Series,” in Appendix B of the *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

In Goldmont microarchitecture, performance monitoring events that support Processor Event Based Sampling (PEBS) and PEBS records that contain processor state information that are associated with at-retirement tagging are marked by “Precise Event”.

**Table 19-27. Performance Events for the Goldmont Microarchitecture**

Event Num.	Umask Value	Event Name	Description	Comment
03H	10H	LD_BLOCKS.ALL_BLOCK	Counts anytime a load that retires is blocked for any reason.	Precise Event
03H	08H	LD_BLOCKS.UTLB_MISS	Counts loads blocked because they are unable to find their physical address in the micro TLB (UTLB).	Precise Event
03H	02H	LD_BLOCKS.STORE_FORWARD	Counts a load blocked from using a store forward because of an address/size mismatch; only one of the loads blocked from each store will be counted.	Precise Event

Table 19-27. Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
03H	01H	LD_BLOCKS.DATA_UNKNOWN	Counts a load blocked from using a store forward, but did not occur because the store data was not available at the right time. The forward might occur subsequently when the data is available.	Precise Event
03H	04H	LD_BLOCKS.4K_ALIAS	Counts loads that block because their address modulo 4K matches a pending store.	Precise Event
05H	01H	PAGE_WALKS.D_SIDE_CYCLES	Counts every core cycle when a Data-side (walks due to data operation) page walk is in progress.	
05H	02H	PAGE_WALKS.I_SIDE_CYCLES	Counts every core cycle when an Instruction-side (walks due to an instruction fetch) page walk is in progress.	
05H	03H	PAGE_WALKS.CYCLES	Counts every core cycle a page-walk is in progress due to either a data memory operation, or an instruction fetch.	
0EH	00H	UOPS_ISSUED.ANY	Counts uops issued by the front end and allocated into the back end of the machine. This event counts uops that retire as well as uops that were speculatively executed but didn't retire. The sort of speculative uops that might be counted includes, but is not limited to those uops issued in the shadow of a mispredicted branch, those uops that are inserted during an assist (such as for a denormal floating-point result), and (previously allocated) uops that might be canceled during a machine clear.	
13H	02H	MISALIGN_MEM_REF.LOAD_PAGE_SPLIT	Counts when a memory load of a uop that spans a page boundary (a split) is retired.	Precise Event
13H	04H	MISALIGN_MEM_REF.STORE_PAGE_SPLIT	Counts when a memory store of a uop that spans a page boundary (a split) is retired.	Precise Event
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	Counts memory requests originating from the core that reference a cache line in the L2 cache.	
2EH	41H	LONGEST_LAT_CACHE.MISS	Counts memory requests originating from the core that miss in the L2 cache.	
30H	00H	L2_REJECT_XQ.ALL	Counts the number of demand and prefetch transactions that the L2 XQ rejects due to a full or near full condition which likely indicates back pressure from the intra-die interconnect (IDI) fabric. The XQ may reject transactions from the L2Q (non-cacheable requests), L2 misses and L2 write-back victims.	
31H	00H	CORE_REJECT_L2Q.ALL	Counts the number of demand and L1 prefetcher requests rejected by the L2Q due to a full or nearly full condition which likely indicates back pressure from L2Q. It also counts requests that would have gone directly to the XQ, but are rejected due to a full or nearly full condition, indicating back pressure from the IDI link. The L2Q may also reject transactions from a core to ensure fairness between cores, or to delay a core's dirty eviction when the address conflicts with incoming external snoops.	
3CH	00H	CPU_CLK_UNHALTED.CORE_P	Core cycles when core is not halted. This event uses a programmable general purpose performance counter.	
3CH	01H	CPU_CLK_UNHALTED.REF	Reference cycles when core is not halted. This event uses a programmable general purpose performance counter.	
51H	01H	DL1.DIRTY_EVICTION	Counts when a modified (dirty) cache line is evicted from the data L1 cache and needs to be written back to memory. No count will occur if the evicted line is clean, and hence does not require a writeback.	

**Table 19-27. Performance Events for the Goldmont Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Name	Description	Comment
80H	01H	ICACHE.HIT	Counts requests to the Instruction Cache (ICache) for one or more bytes in an ICache Line and that cache line is in the Icache (hit). The event strives to count on a cache line basis, so that multiple accesses which hit in a single cache line count as one ICACHE.HIT. Specifically, the event counts when straight line code crosses the cache line boundary, or when a branch target is to a new line, and that cache line is in the ICache. This event counts differently than Intel processors based on the Silvermont microarchitecture.	
80H	02H	ICACHE.MISSES	Counts requests to the Instruction Cache (ICache) for one or more bytes in an ICache Line and that cache line is not in the Icache (miss). The event strives to count on a cache line basis, so that multiple accesses which miss in a single cache line count as one ICACHE.MISS. Specifically, the event counts when straight line code crosses the cache line boundary, or when a branch target is to a new line, and that cache line is not in the ICache. This event counts differently than Intel processors based on the Silvermont microarchitecture.	
80H	03H	ICACHE.ACCESSSES	Counts requests to the Instruction Cache (ICache) for one or more bytes in an ICache Line. The event strives to count on a cache line basis, so that multiple fetches to a single cache line count as one ICACHE.ACCESS. Specifically, the event counts when accesses from straight line code crosses the cache line boundary, or when a branch target is to a new line. This event counts differently than Intel processors based on the Silvermont microarchitecture.	
81H	04H	ITLB.MISS	Counts the number of times the machine was unable to find a translation in the Instruction Translation Lookaside Buffer (ITLB) for a linear address of an instruction fetch. It counts when new translations are filled into the ITLB. The event is speculative in nature, but will not count translations (page walks) that are begun and not finished, or translations that are finished but not filled into the ITLB.	
86H	00H	FETCH_STALL.ALL	Counts cycles that fetch is stalled due to any reason. That is, the decoder queue is able to accept bytes, but the fetch unit is unable to provide bytes. This will include cycles due to an ITLB miss, ICache miss and other events.	
86H	01H	FETCH_STALL.ITLB_FILL_PENDING_CYCLES	Counts cycles that fetch is stalled due to an outstanding ITLB miss. That is, the decoder queue is able to accept bytes, but the fetch unit is unable to provide bytes due to an ITLB miss. Note: this event is not the same as page walk cycles to retrieve an instruction translation.	
86H	02H	FETCH_STALL.ICACHE_FILL_PENDING_CYCLES	Counts cycles that an ICache miss is outstanding, and instruction fetch is stalled. That is, the decoder queue is able to accept bytes, but the fetch unit is unable to provide bytes, while an Icache miss is outstanding. Note this event is not the same as cycles to retrieve an instruction due to an Icache miss. Rather, it is the part of the Instruction Cache (ICache) miss time where no bytes are available for the decoder.	



Table 19-27. Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
9CH	00H	UOPS_NOT_DELIVERED.ANY	<p>This event is used to measure front-end inefficiencies, i.e., when the front end of the machine is not delivering uops to the back end and the back end has not stalled. This event can be used to identify if the machine is truly front-end bound. When this event occurs, it is an indication that the front end of the machine is operating at less than its theoretical peak performance.</p> <p>Background: We can think of the processor pipeline as being divided into 2 broader parts: the front end and the back end. The front end is responsible for fetching the instruction, decoding into uops in machine understandable format and putting them into a uop queue to be consumed by the back end. The back end then takes these uops and allocates the required resources. When all resources are ready, uops are executed. If the back end is not ready to accept uops from the front end, then we do not want to count these as front-end bottlenecks. However, whenever we have bottlenecks in the back end, we will have allocation unit stalls and eventually force the front end to wait until the back end is ready to receive more uops. This event counts only when the back end is requesting more micro-uops and the front end is not able to provide them. When 3 uops are requested and no uops are delivered, the event counts 3. When 3 are requested, and only 1 is delivered, the event counts 2. When only 2 are delivered, the event counts 1. Alternatively stated, the event will not count if 3 uops are delivered, or if the back end is stalled and not requesting any uops at all. Counts indicate missed opportunities for the front end to deliver a uop to the back end. Some examples of conditions that cause front-end inefficiencies are: lcache misses, ITLB misses, and decoder restrictions that limit the front-end bandwidth.</p> <p>Known Issues: Some uops require multiple allocation slots. These uops will not be charged as a front end 'not delivered' opportunity, and will be regarded as a back-end problem. For example, the INC instruction has one uop that requires 2 issue slots. A stream of INC instructions will not count as UOPS_NOT_DELIVERED, even though only one instruction can be issued per clock. The low uop issue rate for a stream of INC instructions is considered to be a back-end issue.</p>	
B7H	01H, 02H	OFFCORE_RESPONSE	Requires MSR_OFFCORE_RESP[0,1] to specify request type and response. (Duplicated for both MSRs.)	
COH	00H	INST_RETIRED.ANY_P	<p>Counts the number of instructions that retire execution. For instructions that consist of multiple uops, this event counts the retirement of the last uop of the instruction. The event continues counting during hardware interrupts, traps, and inside interrupt handlers. This is an architectural performance event. This event uses a programmable general purpose performance counter. *This event is a Precise Event: the EventingRIP field in the PEBS record is precise to the address of the instruction which caused the event.</p> <p>Note: Because PEBS records can be collected only on IA32_PMC0, only one event can use the PEBS facility at a time.</p>	Precise Event
C2H	00H	UOPS_RETIRED.ANY	Counts uops which have retired.	Precise Event, Not Reduced Skid



**Table 19-27. Performance Events for the Goldmont Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Name	Description	Comment
C2H	01H	UOPS_RETIREDC2H	Counts uops retired that are from the complex flows issued by the micro-sequencer (MS). Counts both the uops from a micro-coded instruction, and the uops that might be generated from a micro-coded assist.	Precise Event, Not Reduced Skid
C2H	08H	UOPS_RETIREDC2H	Counts the number of floating point divide uops retired.	Precise Event
C2H	10H	UOPS_RETIREDC2H	Counts the number of integer divide uops retired.	Precise Event
C3H	01H	MACHINE_CLEARSC3H	Counts the number of times that the processor detects that a program is writing to a code section and has to perform a machine clear because of that modification. Self-modifying code (SMC) causes a severe penalty in all Intel architecture processors.	
C3H	02H	MACHINE_CLEARSC3H	Counts machine clears due to memory ordering issues. This occurs when a snoop request happens and the machine is uncertain if memory ordering will be preserved as another core is in the process of modifying the data.	
C3H	04H	MACHINE_CLEARSC3H	Counts machine clears due to floating-point (FP) operations needing assists. For instance, if the result was a floating-point denormal, the hardware clears the pipeline and reissues uops to produce the correct IEEE compliant denormal result.	
C3H	08H	MACHINE_CLEARSC3H	Counts machine clears due to memory disambiguation. Memory disambiguation happens when a load which has been issued conflicts with a previous un-retired store in the pipeline whose address was not known at issue time, but is later resolved to be the same as the load address.	
C3H	00H	MACHINE_CLEARSC3H	Counts machine clears for any reason.	
C4H	00H	BR_INST_RETIREDC4H	Counts branch instructions retired for all branch types. This is an architectural performance event.	Precise Event
C4H	7EH	BR_INST_RETIREDC4H	Counts retired Jcc (Jump on Conditional Code/Jump if Condition is Met) branch instructions retired, including both when the branch was taken and when it was not taken.	Precise Event
C4H	80H	BR_INST_RETIREDC4H	Counts the number of taken branch instructions retired.	Precise Event
C4H	FEH	BR_INST_RETIREDC4H	Counts Jcc (Jump on Conditional Code/Jump if Condition is Met) branch instructions retired that were taken and does not count when the Jcc branch instruction were not taken.	Precise Event
C4H	F9H	BR_INST_RETIREDC4H	Counts near CALL branch instructions retired.	Precise Event
C4H	FDH	BR_INST_RETIREDC4H	Counts near relative CALL branch instructions retired.	Precise Event
C4H	FBH	BR_INST_RETIREDC4H	Counts near indirect CALL branch instructions retired.	Precise Event
C4H	F7H	BR_INST_RETIREDC4H	Counts near return branch instructions retired.	Precise Event
C4H	EBH	BR_INST_RETIREDC4H	Counts near indirect call or near indirect jmp branch instructions retired.	Precise Event
C4H	BFH	BR_INST_RETIREDC4H	Counts far branch instructions retired. This includes far jump, far call and return, and Interrupt call and return.	Precise Event
C5H	00H	BR_MISP_RETIREDC5H	Counts mispredicted branch instructions retired including all branch types.	Precise Event

Table 19-27. Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
C5H	7EH	BR_MISP_RETIRED.JCC	Counts mispredicted retired Jcc (Jump on Conditional Code/Jump if Condition is Met) branch instructions retired, including both when the branch was supposed to be taken and when it was not supposed to be taken (but the processor predicted the opposite condition).	Precise Event
C5H	FEH	BR_MISP_RETIRED.TAKEN_JCC	Counts mispredicted retired Jcc (Jump on Conditional Code/Jump if Condition is Met) branch instructions retired that were supposed to be taken but the processor predicted that it would not be taken.	Precise Event
C5H	FBH	BR_MISP_RETIRED.IND_CALL	Counts mispredicted near indirect CALL branch instructions retired, where the target address taken was not what the processor predicted.	Precise Event
C5H	F7H	BR_MISP_RETIRED.RETURN	Counts mispredicted near RET branch instructions retired, where the return address taken was not what the processor predicted.	Precise Event
C5H	EBH	BR_MISP_RETIRED.NON_RETURN_IND	Counts mispredicted branch instructions retired that were near indirect call or near indirect jmp, where the target address taken was not what the processor predicted.	Precise Event
CAH	01H	ISSUE_SLOTS_NOT_CONSUMED.RESOURCE_FULL	Counts the number of issue slots per core cycle that were not consumed because of a full resource in the back end. Including but not limited to resources include the Re-order Buffer (ROB), reservation stations (RS), load/store buffers, physical registers, or any other needed machine resource that is currently unavailable. Note that uops must be available for consumption in order for this event to fire. If a uop is not available (Instruction Queue is empty), this event will not count.	
CAH	02H	ISSUE_SLOTS_NOT_CONSUMED.RECOVERY	Counts the number of issue slots per core cycle that were not consumed by the back end because allocation is stalled waiting for a mispredicted jump to retire or other branch-like conditions (e.g. the event is relevant during certain microcode flows). Counts all issue slots blocked while within this window, including slots where uops were not available in the Instruction Queue.	
CAH	00H	ISSUE_SLOTS_NOT_CONSUMED.ANY	Counts the number of issue slots per core cycle that were not consumed by the back end due to either a full resource in the back end (RESOURCE_FULL), or due to the processor recovering from some event (RECOVERY).	
CBH	01H	HW_INTERRUPTS.RECEIVED	Counts hardware interrupts received by the processor.	
CBH	02H	HW_INTERRUPTS.MASKED	Counts the number of core cycles during which interrupts are masked (disabled). Increments by 1 each core cycle that EFLAGS.IF is 0, regardless of whether interrupts are pending or not.	
CBH	04H	HW_INTERRUPTS.PENDING_AND_MASKED	Counts core cycles during which there are pending interrupts, but interrupts are masked (EFLAGS.IF = 0).	
CDH	00H	CYCLES_DIV_BUSY.ALL	Counts core cycles if either divide unit is busy.	
CDH	01H	CYCLES_DIV_BUSY.IDIV	Counts core cycles if the integer divide unit is busy.	
CDH	02H	CYCLES_DIV_BUSY.FPDIV	Counts core cycles if the floating point divide unit is busy.	
DOH	81H	MEM_UOPS_RETIRED.ALL_LOADS	Counts the number of load uops retired.	Precise Event
DOH	82H	MEM_UOPS_RETIRED.ALL_STORES	Counts the number of store uops retired.	Precise Event

**Table 19-27. Performance Events for the Goldmont Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Name	Description	Comment
DOH	83H	MEM_UOPS_RETIRED.ALL	Counts the number of memory uops retired that are either a load or a store or both.	Precise Event
DOH	11H	MEM_UOPS_RETIRED.DTLB_MISS_LOADS	Counts load uops retired that caused a DTLB miss.	Precise Event
DOH	12H	MEM_UOPS_RETIRED.DTLB_MISS_STORES	Counts store uops retired that caused a DTLB miss.	Precise Event
DOH	13H	MEM_UOPS_RETIRED.DTLB_MISS	Counts uops retired that had a DTLB miss on load, store or either. Note that when two distinct memory operations to the same page miss the DTLB, only one of them will be recorded as a DTLB miss.	Precise Event
DOH	21H	MEM_UOPS_RETIRED.LOCK_LOADS	Counts locked memory uops retired. This includes 'regular' locks and bus locks. To specifically count bus locks only, see the offcore response event. A locked access is one with a lock prefix, or an exchange to memory.	Precise Event
DOH	41H	MEM_UOPS_RETIRED.SPLIT_LOADS	Counts load uops retired where the data requested spans a 64 byte cache line boundary.	Precise Event
DOH	42H	MEM_UOPS_RETIRED.SPLIT_STORES	Counts store uops retired where the data requested spans a 64 byte cache line boundary.	Precise Event
DOH	43H	MEM_UOPS_RETIRED.SPLIT	Counts memory uops retired where the data requested spans a 64 byte cache line boundary.	Precise Event
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Counts load uops retired that hit the L1 data cache.	Precise Event
D1H	08H	MEM_LOAD_UOPS_RETIRED.L1_MISS	Counts load uops retired that miss the L1 data cache.	Precise Event
D1H	02H	MEM_LOAD_UOPS_RETIRED.L2_HIT	Counts load uops retired that hit in the L2 cache.	Precise Event
0xD1H	10H	MEM_LOAD_UOPS_RETIRED.L2_MISS	Counts load uops retired that miss in the L2 cache.	Precise Event
D1H	20H	MEM_LOAD_UOPS_RETIRED.HITM	Counts load uops retired where the cache line containing the data was in the modified state of another core or modules cache (HITM). More specifically, this means that when the load address was checked by other caching agents (typically another processor) in the system, one of those caching agents indicated that they had a dirty copy of the data. Loads that obtain a HITM response incur greater latency than most that is typical for a load. In addition, since HITM indicates that some other processor had this data in its cache, it implies that the data was shared between processors, or potentially was a lock or semaphore value. This event is useful for locating sharing, false sharing, and contended locks.	Precise Event

**Table 19-27. Performance Events for the Goldmont Microarchitecture (Contd.)**

Event Num.	Umask Value	Event Name	Description	Comment
D1H	40H	MEM_LOAD_UOPS_RETIRED.WCB_HIT	Counts memory load uops retired where the data is retrieved from the WCB (or fill buffer), indicating that the load found its data while that data was in the process of being brought into the L1 cache. Typically a load will receive this indication when some other load or prefetch missed the L1 cache and was in the process of retrieving the cache line containing the data, but that process had not yet finished (and written the data back to the cache). For example, consider load X and Y, both referencing the same cache line that is not in the L1 cache. If load X misses cache first, it obtains and WCB (or fill buffer) begins the process of requesting the data. When load Y requests the data, it will either hit the WCB, or the L1 cache, depending on exactly what time the request to Y occurs.	Precise Event
D1H	80H	MEM_LOAD_UOPS_RETIRED.DRAM_HIT	Counts memory load uops retired where the data is retrieved from DRAM. Event is counted at retirement, so the speculative loads are ignored. A memory load can hit (or miss) the L1 cache, hit (or miss) the L2 cache, hit DRAM, hit in the WCB or receive a HITM response.	Precise Event
E6H	01H	BACLEARS.ALL	Counts the number of times a BACLEAR is signaled for any reason, including, but not limited to indirect branch/call, Jcc (Jump on Conditional Code/Jump if Condition is Met) branch, unconditional branch/call, and returns.	
E6H	08H	BACLEARS.RETURN	Counts BACLEARS on return instructions.	
E6H	10H	BACLEARS.COND	Counts BACLEARS on Jcc (Jump on Conditional Code/Jump if Condition is Met) branches.	
E7H	01H	MS_DECODED.MS_ENTRY	Counts the number of times the Microcode Sequencer (MS) starts a flow of uops from the MSROM. It does not count every time a uop is read from the MSROM. The most common case that this counts is when a micro-coded instruction is encountered by the front end of the machine. Other cases include when an instruction encounters a fault, trap, or microcode assist of any sort that initiates a flow of uops. The event will count MS startups for uops that are speculative, and subsequently cleared by branch mispredict or a machine clear.	
E9H	01H	DECODE_RESTRICTION.PREDECODE_WRONG	Counts the number of times the prediction (from the pre-decode cache) for instruction length is incorrect.	

## 19.15 PERFORMANCE MONITORING EVENTS FOR PROCESSORS BASED ON THE SILVERMONT MICROARCHITECTURE

Processors based on the Silvermont microarchitecture support the architectural performance monitoring events listed in Table 19-1 and fixed-function performance events using fixed counter. In addition, they also support the following model-specific performance monitoring events listed in Table 19-28. These processors have the CPUID signatures of 06\_37H, 06\_4AH, 06\_4DH, 06\_5AH, and 06\_5DH.

Performance monitoring event descriptions may refer to terminology described in Section B.2, “Intel® Xeon® processor 5500 Series,” in Appendix B of the *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

**Table 19-28. Performance Events for Silvermont Microarchitecture**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
03H	01H	REHABQ.LD_BLOCK_ST_FORWARD	Loads blocked due to store forward restriction.	This event counts the number of retired loads that were prohibited from receiving forwarded data from the store because of address mismatch.
03H	02H	REHABQ.LD_BLOCK_STD_NOTREADY	Loads blocked due to store data not ready.	This event counts the cases where a forward was technically possible, but did not occur because the store data was not available at the right time.
03H	04H	REHABQ.ST_SPLITS	Store uops that split cache line boundary.	This event counts the number of retire stores that experienced cache line boundary splits.
03H	08H	REHABQ.LD_SPLITS	Load uops that split cache line boundary.	This event counts the number of retire loads that experienced cache line boundary splits.
03H	10H	REHABQ.LOCK	Uops with lock semantics.	This event counts the number of retired memory operations with lock semantics. These are either implicit locked instructions such as the XCHG instruction or instructions with an explicit LOCK prefix (FOH).
03H	20H	REHABQ.STA_FULL	Store address buffer full.	This event counts the number of retired stores that are delayed because there is not a store address buffer available.
03H	40H	REHABQ.ANY_LD	Any reissued load uops.	This event counts the number of load uops reissued from Rehabq.
03H	80H	REHABQ.ANY_ST	Any reissued store uops.	This event counts the number of store uops reissued from Rehabq.
04H	01H	MEM_UOPS_RETIREDL1_MISS_LOADS	Loads retired that missed L1 data cache.	This event counts the number of load ops retired that miss in L1 Data cache. Note that prefetch misses will not be counted.
04H	02H	MEM_UOPS_RETIREDL2_HIT_LOADS	Loads retired that hit L2.	This event counts the number of load micro-ops retired that hit L2.
04H	04H	MEM_UOPS_RETIREDL2_MISS_LOADS	Loads retired that missed L2.	This event counts the number of load micro-ops retired that missed L2.
04H	08H	MEM_UOPS_RETIREDDTLB_MISS_LOADS	Loads missed DTLB.	This event counts the number of load ops retired that had DTLB miss.
04H	10H	MEM_UOPS_RETIREDDTLB_MISS_LOADS	Loads missed UTLB.	This event counts the number of load ops retired that had UTLB miss.
04H	20H	MEM_UOPS_RETIREDHITM	Cross core or cross module hitm.	This event counts the number of load ops retired that got data from the other core or from the other module.
04H	40H	MEM_UOPS_RETIREDAALL_LOADS	All Loads.	This event counts the number of load ops retired.
04H	80H	MEM_UOP_RETIREDAALL_STORES	All Stores.	This event counts the number of store ops retired.
05H	01H	PAGE_WALKS.D_SIDE_CYCLES	Duration of D-side page-walks in core cycles.	This event counts every cycle when a D-side (walks due to a load) page walk is in progress. Page walk duration divided by number of page walks is the average duration of page-walks. Edge trigger bit must be cleared. Set Edge to count the number of page walks.
05H	02H	PAGE_WALKS.I_SIDE_CYCLES	Duration of I-side page-walks in core cycles.	This event counts every cycle when an I-side (walks due to an instruction fetch) page walk is in progress. Page walk duration divided by number of page walks is the average duration of page-walks. Edge trigger bit must be cleared. Set Edge to count the number of page walks.

Table 19-28. Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
05H	03H	PAGE_WALKS.WALKS	Total number of page-walks that are completed (I-side and D-side).	This event counts when a data (D) page walk or an instruction (I) page walk is completed or started. Since a page walk implies a TLB miss, the number of TLB misses can be counted by counting the number of pagewalks. Edge trigger bit must be set. Clear Edge to count the number of cycles.
2EH	41H	LONGEST_LAT_CACHE.MISS	L2 cache request misses.	This event counts the total number of L2 cache references and the number of L2 cache misses respectively. L3 is not supported in Silvermont microarchitecture.
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	L2 cache requests from this core.	This event counts requests originating from the core that references a cache line in the L2 cache. L3 is not supported in Silvermont microarchitecture.
30H	00H	L2_REJECT_XQ.ALL	Counts the number of request from the L2 that were not accepted into the XQ.	This event counts the number of demand and prefetch transactions that the L2 XQ rejects due to a full or near full condition which likely indicates back pressure from the IDI link. The XQ may reject transactions from the L2Q (non-cacheable requests), BBS (L2 misses) and WOB (L2 write-back victims).
31H	00H	CORE_REJECT_L2Q.ALL	Counts the number of request that were not accepted into the L2Q because the L2Q is FULL.	This event counts the number of demand and L1 prefetcher requests rejected by the L2Q due to a full or nearly full condition which likely indicates back pressure from L2Q. It also counts requests that would have gone directly to the XQ, but are rejected due to a full or nearly full condition, indicating back pressure from the IDI link. The L2Q may also reject transactions from a core to insure fairness between cores, or to delay a core's dirty eviction when the address conflicts incoming external snoops. (Note that L2 prefetcher requests that are dropped are not counted by this event.)
3CH	00H	CPU_CLK_UNHALTED.CORE_P	Core cycles when core is not halted.	This event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. In mobile systems the core frequency may change from time to time. For this reason this event may have a changing ratio with regards to time.
N/A	N/A	CPU_CLK_UNHALTED.CORE	Core cycles when core is not halted.	This uses the fixed counter 1 to count the same condition as CPU_CLK_UNHALTED.CORE_P does.
3CH	01H	CPU_CLK_UNHALTED.REF_P	Bus cycles when core is not halted.	This event counts the number of bus cycles that the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. In mobile systems the core frequency may change from time. This event is not affected by core frequency changes.
N/A	N/A	CPU_CLK_UNHALTED.REF_TSC	Reference cycles when core is not halted.	This event counts the number of reference cycles at a TSC rate that the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. In mobile systems the core frequency may change from time. This event is not affected by core frequency changes.
80H	01H	ICACHE.HIT	Instruction fetches from lcache.	This event counts all instruction fetches from the instruction cache.
80H	02H	ICACHE.MISSES	lcache miss.	This event counts all instruction fetches that miss the Instruction cache or produce memory requests. This includes uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.

Table 19-28. Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
80H	03H	ICACHE.ACCESSSES	Instruction fetches.	This event counts all instruction fetches, including uncacheable fetches.
B7H	01H	OFFCORE_RESPONSE_0	See Section 18.5.2.2.	Requires MSR_OFFCORE_RESP0 to specify request type and response.
B7H	02H	OFFCORE_RESPONSE_1	See Section 18.5.2.2.	Requires MSR_OFFCORE_RESP1 to specify request type and response.
C0H	00H	INST_RETIRED.ANY_P	Instructions retired (PEBS supported with IA32_PMC0).	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers.
N/A	N/A	INST_RETIRED.ANY	Instructions retired.	This uses the fixed counter 0 to count the same condition as INST_RETIRED.ANY_P does.
C2H	01H	UOPS_RETIRED.MS	MSROM micro-ops retired.	This event counts the number of micro-ops retired that were supplied from MSROM.
C2H	10H	UOPS_RETIRED.ALL	Micro-ops retired.	This event counts the number of micro-ops retired.
C3H	01H	MACHINE_CLEARS.SMC	Self-Modifying Code detected.	This event counts the number of times that a program writes to a code section. Self-modifying code causes a severe penalty in all Intel® architecture processors.
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Stalls due to Memory ordering.	This event counts the number of times that pipeline was cleared due to memory ordering issues.
C3H	04H	MACHINE_CLEARS.FP_ASSIST	Stalls due to FP assists.	This event counts the number of times that pipeline stalled due to FP operations needing assists.
C3H	08H	MACHINE_CLEARS.ALL	Stalls due to any causes.	This event counts the number of times that pipeline stalled due to due to any causes (including SMC, MO, FP assist, etc.).
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Retired branch instructions.	This event counts the number of branch instructions retired.
C4H	7EH	BR_INST_RETIRED.JCC	Retired branch instructions that were conditional jumps.	This event counts the number of branch instructions retired that were conditional jumps.
C4H	BFH	BR_INST_RETIRED.FAR_BRANCH	Retired far branch instructions.	This event counts the number of far branch instructions retired.
C4H	EBH	BR_INST_RETIRED.NO_N_RETURN_IND	Retired instructions of near indirect jmp or call.	This event counts the number of branch instructions retired that were near indirect call or near indirect jmp.
C4H	F7H	BR_INST_RETIRED.RETURN	Retired near return instructions.	This event counts the number of near RET branch instructions retired.
C4H	F9H	BR_INST_RETIRED.CALL	Retired near call instructions.	This event counts the number of near CALL branch instructions retired.
C4H	FBH	BR_INST_RETIRED.IND_CALL	Retired near indirect call instructions.	This event counts the number of near indirect CALL branch instructions retired.
C4H	FDH	BR_INST_RETIRED.REL_CALL	Retired near relative call instructions.	This event counts the number of near relative CALL branch instructions retired.
C4H	FEH	BR_INST_RETIRED.TAKEN_JCC	Retired conditional jumps that were taken.	This event counts the number of branch instructions retired that were conditional jumps and taken.
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Retired mispredicted branch instructions.	This event counts the number of mispredicted branch instructions retired.



Table 19-28. Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
C5H	7EH	BR_MISP_RETIRED.JCC	Retired mispredicted conditional jumps.	This event counts the number of mispredicted branch instructions retired that were conditional jumps.
C5H	BFH	BR_MISP_RETIRED.FAR	Retired mispredicted far branch instructions.	This event counts the number of mispredicted far branch instructions retired.
C5H	EBH	BR_MISP_RETIRED.NON_RETURN_IND	Retired mispredicted instructions of near indirect jmp or call.	This event counts the number of mispredicted branch instructions retired that were near indirect call or near indirect jmp.
C5H	F7H	BR_MISP_RETIRED.RETURN	Retired mispredicted near return instructions.	This event counts the number of mispredicted near RET branch instructions retired.
C5H	F9H	BR_MISP_RETIRED.CALL	Retired mispredicted near call instructions.	This event counts the number of mispredicted near CALL branch instructions retired.
C5H	FBH	BR_MISP_RETIRED.IND_CALL	Retired mispredicted near indirect call instructions.	This event counts the number of mispredicted near indirect CALL branch instructions retired.
C5H	FDH	BR_MISP_RETIRED.REL_CALL	Retired mispredicted near relative call instructions	This event counts the number of mispredicted near relative CALL branch instructions retired.
C5H	FEH	BR_MISP_RETIRED.TAKEN_JCC	Retired mispredicted conditional jumps that were taken.	This event counts the number of mispredicted branch instructions retired that were conditional jumps and taken.
CAH	01H	NO_ALLOC_CYCLES.ROB_FULL	Counts the number of cycles when no uops are allocated and the ROB is full (less than 2 entries available).	Counts the number of cycles when no uops are allocated and the ROB is full (less than 2 entries available).
CAH	20H	NO_ALLOC_CYCLES.RAT_STALL	Counts the number of cycles when no uops are allocated and a RATstall is asserted.	Counts the number of cycles when no uops are allocated and a RATstall is asserted.
CAH	3FH	NO_ALLOC_CYCLES.ALL	Front end not delivering.	This event counts the number of cycles when the front end does not provide any instructions to be allocated for any reason.
CAH	50H	NO_ALLOC_CYCLES.NO_T_DELIVERED	Front end not delivering back end not stalled.	This event counts the number of cycles when the front end does not provide any instructions to be allocated but the back end is not stalled.
CBH	01H	RS_FULL_STALL.MEC	MEC RS full.	This event counts the number of cycles the allocation pipe line stalled due to the RS for the MEC cluster is full.
CBH	1FH	RS_FULL_STALL.ALL	Any RS full.	This event counts the number of cycles that the allocation pipe line stalled due to any one of the RS is full.
CDH	01H	CYCLES_DIV_BUSY.ANY	Divider Busy.	This event counts the number of cycles the divider is busy.
E6H	01H	BACLEARS.ALL	BACLEARS asserted for any branch.	This event counts the number of baclears for any type of branch.
E6H	08H	BACLEARS.RETURN	BACLEARS asserted for return branch.	This event counts the number of baclears for return branches.



**Table 19-28. Performance Events for Silvermont Microarchitecture**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
E6H	10H	BACLEARS.COND	BACLEARS asserted for conditional branch.	This event counts the number of baclears for conditional branches.
E7H	01H	MS_DECODED.MS_ENTRY	MS Decode starts.	This event counts the number of times the MSROM starts a flow of UOPS.

### 19.15.1 Performance Monitoring Events for Processors Based on the Airmont Microarchitecture

Intel processors based on the Airmont microarchitecture support the same architectural and the model-specific performance monitoring events as processors based on the Silvermont microarchitecture. All of the events listed in Table 19-28 apply. These processors have the CPUID signatures that include 06\_4CH.

## 19.16 PERFORMANCE MONITORING EVENTS FOR 45 NM AND 32 NM INTEL® ATOM™ PROCESSORS

45 nm and 32 nm processors based on the Intel® Atom™ microarchitecture support the architectural performance monitoring events listed in Table 19-1 and fixed-function performance events using fixed counter listed in Table 19-24. In addition, they also support the following model-specific performance monitoring events listed in Table 19-29.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
02H	81H	STORE_FORWARDS.GOOD	Good store forwards.	This event counts the number of times store data was forwarded directly to a load.
06H	00H	SEGMENT_REG_LOADS.ANY	Number of segment register loads.	This event counts the number of segment register load operations. Instructions that load new values into segment registers cause a penalty. This event indicates performance issues in 16-bit code. If this event occurs frequently, it may be useful to calculate the number of instructions retired per segment register load. If the resulting calculation is low (on average a small number of instructions are executed between segment register loads), then the code's segment register usage should be optimized.  As a result of branch misprediction, this event is speculative and may include segment register loads that do not actually occur. However, most segment register loads are internally serialized and such speculative effects are minimized.
07H	01H	PREFETCH.PREFETCHT0	Streaming SIMD Extensions (SSE) PrefetchT0 instructions executed.	This event counts the number of times the SSE instruction prefetchT0 is executed. This instruction prefetches the data to the L1 data cache and L2 cache.
07H	06H	PREFETCH.SW_L2	Streaming SIMD Extensions (SSE) PrefetchT1 and PrefetchT2 instructions executed.	This event counts the number of times the SSE instructions prefetchT1 and prefetchT2 are executed. These instructions prefetch the data to the L2 cache.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
07H	08H	PREFETCH.PREFETCHNTA	Streaming SIMD Extensions (SSE) Prefetch NTA instructions executed.	This event counts the number of times the SSE instruction prefetchNTA is executed. This instruction prefetches the data to the L1 data cache.
08H	07H	DATA_TLB_MISSES.DTLB_MISS	Memory accesses that missed the DTLB.	This event counts the number of Data Table Lookaside Buffer (DTLB) misses. The count includes misses detected as a result of speculative accesses. Typically a high count for this event indicates that the code accesses a large number of data pages.
08H	05H	DATA_TLB_MISSES.DTLB_MISS_LD	DTLB misses due to load operations.	This event counts the number of Data Table Lookaside Buffer (DTLB) misses due to load operations. This count includes misses detected as a result of speculative accesses.
08H	09H	DATA_TLB_MISSES.LO_DTLB_MISS_LD	LO_DTLB misses due to load operations.	This event counts the number of LO_DTLB misses due to load operations. This count includes misses detected as a result of speculative accesses.
08H	06H	DATA_TLB_MISSES.DTLB_MISS_ST	DTLB misses due to store operations.	This event counts the number of Data Table Lookaside Buffer (DTLB) misses due to store operations. This count includes misses detected as a result of speculative accesses.
0CH	03H	PAGE_WALKS.WALKS	Number of page-walks executed.	This event counts the number of page-walks executed due to either a DTLB or ITLB miss. The page walk duration, PAGE_WALKS.CYCLES, divided by number of page walks is the average duration of a page walk. This can hint to whether most of the page-walks are satisfied by the caches or cause an L2 cache miss. Edge trigger bit must be set.
0CH	03H	PAGE_WALKS.CYCLES	Duration of page-walks in core cycles.	This event counts the duration of page-walks in core cycles. The paging mode in use typically affects the duration of page walks. Page walk duration divided by number of page walks is the average duration of page-walks. This can hint at whether most of the page-walks are satisfied by the caches or cause an L2 cache miss. Edge trigger bit must be cleared.
10H	01H	X87_COMP_OPS_EXE.ANY.S	Floating point computational micro-ops executed.	This event counts the number of x87 floating point computational micro-ops executed.
10H	81H	X87_COMP_OPS_EXE.ANY.AR	Floating point computational micro-ops retired.	This event counts the number of x87 floating point computational micro-ops retired.
11H	01H	FP_ASSIST	Floating point assists.	This event counts the number of floating point operations executed that required micro-code assist intervention. These assists are required in the following cases. X87 instructions: 1. NaN or denormal are loaded to a register or used as input from memory. 2. Division by 0. 3. Underflow output.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
11H	81H	FP_ASSIST.AR	Floating point assists.	This event counts the number of floating point operations executed that required micro-code assist intervention. These assists are required in the following cases. X87 instructions: 1. NaN or denormal are loaded to a register or used as input from memory. 2. Division by 0. 3. Underflow output.
12H	01H	MUL.S	Multiply operations executed.	This event counts the number of multiply operations executed. This includes integer as well as floating point multiply operations.
12H	81H	MUL.AR	Multiply operations retired.	This event counts the number of multiply operations retired. This includes integer as well as floating point multiply operations.
13H	01H	DIV.S	Divide operations executed.	This event counts the number of divide operations executed. This includes integer divides, floating point divides and square-root operations executed.
13H	81H	DIV.AR	Divide operations retired.	This event counts the number of divide operations retired. This includes integer divides, floating point divides and square-root operations executed.
14H	01H	CYCLES_DIV_BUSY	Cycles the divider is busy.	This event counts the number of cycles the divider is busy executing divide or square root operations. The divide can be integer, X87 or Streaming SIMD Extensions (SSE). The square root operation can be either X87 or SSE.
21H	See Table 18-61	L2_ADS	Cycles L2 address bus is in use.	This event counts the number of cycles the L2 address bus is being used for accesses to the L2 cache or bus queue. This event can count occurrences for this core or both cores.
22H	See Table 18-61	L2_DBUS_BUSY	Cycles the L2 cache data bus is busy.	This event counts core cycles during which the L2 cache data bus is busy transferring data from the L2 cache to the core. It counts for all L1 cache misses (data and instruction) that hit the L2 cache. The count will increment by two for a full cache-line request.
24H	See Table 18-61 and Table 18-63	L2_LINES_IN	L2 cache misses.	This event counts the number of cache lines allocated in the L2 cache. Cache lines are allocated in the L2 cache as a result of requests from the L1 data and instruction caches and the L2 hardware prefetchers to cache lines that are missing in the L2 cache. This event can count occurrences for this core or both cores. This event can also count demand requests and L2 hardware prefetch requests together or separately.
25H	See Table 18-61	L2_M_LINES_IN	L2 cache line modifications.	This event counts whenever a modified cache line is written back from the L1 data cache to the L2 cache. This event can count occurrences for this core or both cores.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
26H	See Table 18-61 and Table 18-63	L2_LINES_OUT	L2 cache lines evicted.	This event counts the number of L2 cache lines evicted. This event can count occurrences for this core or both cores. This event can also count evictions due to demand requests and L2 hardware prefetch requests together or separately.
27H	See Table 18-61 and Table 18-63	L2_M_LINES_OUT	Modified lines evicted from the L2 cache.	This event counts the number of L2 modified cache lines evicted. These lines are written back to memory unless they also exist in a shared-state in one of the L1 data caches. This event can count occurrences for this core or both cores. This event can also count evictions due to demand requests and L2 hardware prefetch requests together or separately.
28H	See Table 18-61 and Table 18-64	L2_IFETCH	L2 cacheable instruction fetch requests.	This event counts the number of instruction cache line requests from the ICache. It does not include fetch requests from uncacheable memory. It does not include ITLB miss accesses. This event can count occurrences for this core or both cores. This event can also count accesses to cache lines at different MESI states.
29H	See Table 18-61, Table 18-63 and Table 18-64	L2_LD	L2 cache reads.	This event counts L2 cache read requests coming from the L1 data cache and L2 prefetchers. This event can count occurrences for this core or both cores. This event can count occurrences - for this core or both cores. - due to demand requests and L2 hardware prefetch requests together or separately. - of accesses to cache lines at different MESI states.
2AH	See Table 18-61 and Table 18-64	L2_ST	L2 store requests.	This event counts all store operations that miss the L1 data cache and request the data from the L2 cache. This event can count occurrences for this core or both cores. This event can also count accesses to cache lines at different MESI states.
2BH	See Table 18-61 and Table 18-64	L2_LOCK	L2 locked accesses.	This event counts all locked accesses to cache lines that miss the L1 data cache. This event can count occurrences for this core or both cores. This event can also count accesses to cache lines at different MESI states.
2EH	See Table 18-61, Table 18-63 and Table 18-64	L2_RQSTS	L2 cache requests.	This event counts all completed L2 cache requests. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, instruction fetches, and all L2 hardware prefetch requests. This event can count occurrences - for this core or both cores. - due to demand requests and L2 hardware prefetch requests together, or separately. - of accesses to cache lines at different MESI states.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
2EH	41H	L2_RQSTS.SELF.DEMAND.I_STATE	L2 cache demand requests from this core that missed the L2.	This event counts all completed L2 cache demand requests from this core that miss the L2 cache. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, and instruction fetches. This is an architectural performance event.
2EH	4FH	L2_RQSTS.SELF.DEMAND.MESI	L2 cache demand requests from this core.	This event counts all completed L2 cache demand requests from this core. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, and instruction fetches. This is an architectural performance event.
30H	See Table 18-61, Table 18-63 and Table 18-64	L2_REJECT_BUSQ	Rejected L2 cache requests.	This event indicates that a pending L2 cache request that requires a bus transaction is delayed from moving to the bus queue. Some of the reasons for this event are: - The bus queue is full. - The bus queue already holds an entry for a cache line in the same set. The number of events is greater or equal to the number of requests that were rejected. - For this core or both cores. - Due to demand requests and L2 hardware prefetch requests together, or separately. - Of accesses to cache lines at different MESI states.
32H	See Table 18-61	L2_NO_REQ	Cycles no L2 cache requests are pending.	This event counts the number of cycles that no L2 cache requests are pending.
3AH	00H	EIST_TRANS	Number of Enhanced Intel SpeedStep(R) Technology (EIST) transitions.	This event counts the number of Enhanced Intel SpeedStep(R) Technology (EIST) transitions that include a frequency change, either with or without VID change. This event is incremented only while the counting core is in CO state. In situations where an EIST transition was caused by hardware as a result of CxE state transitions, those EIST transitions will also be registered in this event.  Enhanced Intel Speedstep Technology transitions are commonly initiated by OS, but can be initiated by HW internally. For example: CxE states are C-states (C1,C2,C3...) which not only place the CPU into a sleep state by turning off the clock and other components, but also lower the voltage (which reduces the leakage power consumption). The same is true for thermal throttling transition which uses Enhanced Intel Speedstep Technology internally.
3BH	COH	THERMAL_TRIP	Number of thermal trips.	This event counts the number of thermal trips. A thermal trip occurs whenever the processor temperature exceeds the thermal trip threshold temperature. Following a thermal trip, the processor automatically reduces frequency and voltage. The processor checks the temperature every millisecond, and returns to normal when the temperature falls below the thermal trip threshold temperature.

Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)

Event Num.	Umask Value	Event Name	Definition	Description and Comment
3CH	00H	CPU_CLK_UNHALTED.CORE_P	Core cycles when core is not halted.	<p>This event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. This event is a component in many key event ratios.</p> <p>In mobile systems the core frequency may change from time to time. For this reason this event may have a changing ratio with regards to time. In systems with a constant core frequency, this event can give you a measurement of the elapsed time while the core was not in halt state by dividing the event count by the core frequency.</p> <ul style="list-style-type: none"> <li>-This is an architectural performance event.</li> <li>- The event CPU_CLK_UNHALTED.CORE_P is counted by a programmable counter.</li> <li>- The event CPU_CLK_UNHALTED.CORE is counted by a designated fixed counter, leaving the two programmable counters available for other events.</li> </ul>
3CH	01H	CPU_CLK_UNHALTED.BUS	Bus cycles when core is not halted.	<p>This event counts the number of bus cycles while the core is not in the halt state. This event can give you a measurement of the elapsed time while the core was not in the halt state, by dividing the event count by the bus frequency. The core enters the halt state when it is running the HLT instruction.</p> <p>The event also has a constant ratio with CPU_CLK_UNHALTED.REF event, which is the maximum bus to processor frequency ratio.</p> <p>Non-halted bus cycles are a component in many key event ratios.</p>
3CH	02H	CPU_CLK_UNHALTED.NO_OTHER	Bus cycles when core is active and the other is halted.	<p>This event counts the number of bus cycles during which the core remains non-halted, and the other core on the processor is halted.</p> <p>This event can be used to determine the amount of parallelism exploited by an application or a system. Divide this event count by the bus frequency to determine the amount of time that only one core was in use.</p>
40H	21H	L1D_CACHE.LD	L1 Cacheable Data Reads.	This event counts the number of data reads from cacheable memory.
40H	22H	L1D_CACHE.ST	L1 Cacheable Data Writes.	This event counts the number of data writes to cacheable memory.
60H	See Table 18-61 and Table 18-62.	BUS_REQUEST_OUTSTANDING	Outstanding cacheable data read bus requests duration.	This event counts the number of pending full cache line read transactions on the bus occurring in each cycle. A read transaction is pending from the cycle it is sent on the bus until the full cache line is received by the processor. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
61H	See Table 18-62.	BUS_BNR_DRV	Number of Bus Not Ready signals asserted.	<p>This event counts the number of Bus Not Ready (BNR) signals that the processor asserts on the bus to suspend additional bus requests by other bus agents. A bus agent asserts the BNR signal when the number of data and snoop transactions is close to the maximum that the bus can handle.</p> <p>While this signal is asserted, new transactions cannot be submitted on the bus. As a result, transaction latency may have higher impact on program performance. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.</p>
62H	See Table 18-62.	BUS_DRDY_CLOCKS	Bus cycles when data is sent on the bus.	<p>This event counts the number of bus cycles during which the DRDY (Data Ready) signal is asserted on the bus. The DRDY signal is asserted when data is sent on the bus.</p> <p>This event counts the number of bus cycles during which this agent (the processor) writes data on the bus back to memory or to other bus agents. This includes all explicit and implicit data writebacks, as well as partial writes.</p> <p>Note: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.</p>
63H	See Table 18-61 and Table 18-62.	BUS_LOCK_CLOCKS	Bus cycles when a LOCK signal is asserted.	<p>This event counts the number of bus cycles, during which the LOCK signal is asserted on the bus. A LOCK signal is asserted when there is a locked memory access, due to:</p> <ul style="list-style-type: none"> <li>- Uncacheable memory.</li> <li>- Locked operation that spans two cache lines.</li> <li>- Page-walk from an uncacheable page table.</li> </ul> <p>Bus locks have a very high performance penalty and it is highly recommended to avoid such accesses. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.</p>
64H	See Table 18-61.	BUS_DATA_RCV	Bus cycles while processor receives data.	<p>This event counts the number of cycles during which the processor is busy receiving data. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.</p>
65H	See Table 18-61 and Table 18-62.	BUS_TRANS_BRD	Burst read bus transactions.	<p>This event counts the number of burst read transactions including:</p> <ul style="list-style-type: none"> <li>- L1 data cache read misses (and L1 data cache hardware prefetches).</li> <li>- L2 hardware prefetches by the DPL and L2 streamer.</li> <li>- IFU read misses of cacheable lines.</li> </ul> <p>It does not include RFO transactions.</p>
66H	See Table 18-61 and Table 18-62.	BUS_TRANS_RFO	RFO bus transactions.	<p>This event counts the number of Read For Ownership (RFO) bus transactions, due to store operations that miss the L1 data cache and the L2 cache. This event also counts RFO bus transactions due to locked operations.</p>

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
67H	See Table 18-61 and Table 18-62.	BUS_TRANS_WB	Explicit writeback bus transactions.	This event counts all explicit writeback bus transactions due to dirty line evictions. It does not count implicit writebacks due to invalidation by a snoop request.
68H	See Table 18-61 and Table 18-62.	BUS_TRANS_IFETCH	Instruction-fetch bus transactions.	This event counts all instruction fetch full cache line bus transactions.
69H	See Table 18-61 and Table 18-62.	BUS_TRANS_INVALID	Invalidate bus transactions.	This event counts all invalidate transactions. Invalidate transactions are generated when: - A store operation hits a shared line in the L2 cache. - A full cache line write misses the L2 cache or hits a shared line in the L2 cache.
6AH	See Table 18-61 and Table 18-62.	BUS_TRANS_PWR	Partial write bus transaction.	This event counts partial write bus transactions.
6BH	See Table 18-61 and Table 18-62.	BUS_TRANS_P	Partial bus transactions.	This event counts all (read and write) partial bus transactions.
6CH	See Table 18-61 and Table 18-62.	BUS_TRANS_IO	IO bus transactions.	This event counts the number of completed I/O bus transactions as a result of IN and OUT instructions. The count does not include memory mapped IO.
6DH	See Table 18-61 and Table 18-62.	BUS_TRANS_DEF	Deferred bus transactions.	This event counts the number of deferred transactions.
6EH	See Table 18-61 and Table 18-62.	BUS_TRANS_BURST	Burst (full cache-line) bus transactions.	This event counts burst (full cache line) transactions including: - Burst reads. - RFOs. - Explicit writebacks. - Write combine lines.



**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
6FH	See Table 18-61 and Table 18-62.	BUS_TRANS_MEM	Memory bus transactions.	This event counts all memory bus transactions including: - Burst transactions. - Partial reads and writes. - Invalidate transactions. The BUS_TRANS_MEM count is the sum of BUS_TRANS_BURST, BUS_TRANS_P and BUS_TRANS_INVALID.
70H	See Table 18-61 and Table 18-62.	BUS_TRANS_ANY	All bus transactions.	This event counts all bus transactions. This includes: - Memory transactions. - IO transactions (non memory-mapped). - Deferred transaction completion. - Other less frequent transactions, such as interrupts.
77H	See Table 18-61 and Table 18-64.	EXT_SNOOP	External snoops.	This event counts the snoop responses to bus transactions. Responses can be counted separately by type and by bus agent. Note: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.
7AH	See Table 18-62.	BUS_HIT_DRV	HIT signal asserted.	This event counts the number of bus cycles during which the processor drives the HIT# pin to signal HIT snoop response. Note: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.
7BH	See Table 18-62.	BUS_HITM_DRV	HITM signal asserted.	This event counts the number of bus cycles during which the processor drives the HITM# pin to signal HITM snoop response. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.
7DH	See Table 18-61.	BUSQ_EMPTY	Bus queue is empty.	This event counts the number of cycles during which the core did not have any pending transactions in the bus queue. Note: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.
7EH	See Table 18-61 and Table 18-62.	SNOOP_STALL_DRV	Bus stalled for snoops.	This event counts the number of times that the bus snoop stall signal is asserted. During the snoop stall cycles no new bus transactions requiring a snoop response can be initiated on the bus. Note: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.
7FH	See Table 18-61.	BUS_IO_WAIT	IO requests waiting in the bus queue.	This event counts the number of core cycles during which IO requests wait in the bus queue. This event counts IO requests from the core.
80H	03H	ICACHE.ACCESSSES	Instruction fetches.	This event counts all instruction fetches, including uncacheable fetches.
80H	02H	ICACHE.MISSES	Icache miss.	This event counts all instruction fetches that miss the Instruction cache or produce memory requests. This includes uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.
82H	04H	ITLB.FLUSH	ITLB flushes.	This event counts the number of ITLB flushes.
82H	02H	ITLB.MISSES	ITLB misses.	This event counts the number of instruction fetches that miss the ITLB.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
AAH	02H	MACRO_INSTS.CISC_DECODED	CISC macro instructions decoded.	This event counts the number of complex instructions decoded, but not necessarily executed or retired. Only one complex instruction can be decoded at a time.
AAH	03H	MACRO_INSTS.ALL_DECODED	All Instructions decoded.	This event counts the number of instructions decoded.
B0H	00H	SIMD_UOPS_EXEC.S	SIMD micro-ops executed (excluding stores).	This event counts all the SIMD micro-ops executed. This event does not count MOVQ and MOVD stores from register to memory.
B0H	80H	SIMD_UOPS_EXEC.AR	SIMD micro-ops retired (excluding stores).	This event counts the number of SIMD saturated arithmetic micro-ops executed.
B1H	00H	SIMD_SAT_UOP_EXEC.S	SIMD saturated arithmetic micro-ops executed.	This event counts the number of SIMD saturated arithmetic micro-ops executed.
B1H	80H	SIMD_SAT_UOP_EXEC.AR	SIMD saturated arithmetic micro-ops retired.	This event counts the number of SIMD saturated arithmetic micro-ops retired.
B3H	01H	SIMD_UOP_TYPE_EXEC.MUL.S	SIMD packed multiply micro-ops executed.	This event counts the number of SIMD packed multiply micro-ops executed.
B3H	81H	SIMD_UOP_TYPE_EXEC.MUL.AR	SIMD packed multiply micro-ops retired.	This event counts the number of SIMD packed multiply micro-ops retired.
B3H	02H	SIMD_UOP_TYPE_EXEC.SHIFT.S	SIMD packed shift micro-ops executed.	This event counts the number of SIMD packed shift micro-ops executed.
B3H	82H	SIMD_UOP_TYPE_EXEC.SHIFT.AR	SIMD packed shift micro-ops retired.	This event counts the number of SIMD packed shift micro-ops retired.
B3H	04H	SIMD_UOP_TYPE_EXEC.PACK.S	SIMD pack micro-ops executed.	This event counts the number of SIMD pack micro-ops executed.
B3H	84H	SIMD_UOP_TYPE_EXEC.PACK.AR	SIMD pack micro-ops retired.	This event counts the number of SIMD pack micro-ops retired.
B3H	08H	SIMD_UOP_TYPE_EXEC.UNPACK.S	SIMD unpack micro-ops executed.	This event counts the number of SIMD unpack micro-ops executed.
B3H	88H	SIMD_UOP_TYPE_EXEC.UNPACK.AR	SIMD unpack micro-ops retired.	This event counts the number of SIMD unpack micro-ops retired.
B3H	10H	SIMD_UOP_TYPE_EXEC.LOGICAL.S	SIMD packed logical micro-ops executed.	This event counts the number of SIMD packed logical micro-ops executed.
B3H	90H	SIMD_UOP_TYPE_EXEC.LOGICAL.AR	SIMD packed logical micro-ops retired.	This event counts the number of SIMD packed logical micro-ops retired.
B3H	20H	SIMD_UOP_TYPE_EXEC.ARITHMETIC.S	SIMD packed arithmetic micro-ops executed.	This event counts the number of SIMD packed arithmetic micro-ops executed.
B3H	A0H	SIMD_UOP_TYPE_EXEC.ARITHMETIC.AR	SIMD packed arithmetic micro-ops retired.	This event counts the number of SIMD packed arithmetic micro-ops retired.
COH	00H	INST_RETIRED.ANY_P	Instructions retired (precise event).	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
N/A	00H	INST_RETIRED.ANY	Instructions retired.	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers.
C2H	10H	UOPS_RETIRED.ANY	Micro-ops retired.	This event counts the number of micro-ops retired. The processor decodes complex macro instructions into a sequence of simpler micro-ops. Most instructions are composed of one or two micro-ops. Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists. In some cases micro-op sequences are fused or whole instructions are fused into one micro-op. See other UOPS_RETIRED events for differentiating retired fused and non-fused micro-ops.
C3H	01H	MACHINE_CLEAR.SMC	Self-Modifying Code detected.	This event counts the number of times that a program writes to a code section. Self-modifying code causes a severe penalty in all Intel® architecture processors.
C4H	00H	BR_INST_RETIRED.ANY	Retired branch instructions.	This event counts the number of branch instructions retired. <b>This is an architectural performance event.</b>
C4H	01H	BR_INST_RETIRED.PRED_NOT_TAKEN	Retired branch instructions that were predicted not-taken.	This event counts the number of branch instructions retired that were correctly predicted to be not-taken.
C4H	02H	BR_INST_RETIRED.MISPRED_NOT_TAKEN	Retired branch instructions that were mispredicted not-taken.	This event counts the number of branch instructions retired that were mispredicted and not-taken.
C4H	04H	BR_INST_RETIRED.PRED_TAKEN	Retired branch instructions that were predicted taken.	This event counts the number of branch instructions retired that were correctly predicted to be taken.
C4H	08H	BR_INST_RETIRED.MISPRED_TAKEN	Retired branch instructions that were mispredicted taken.	This event counts the number of branch instructions retired that were mispredicted and taken.
C4H	0AH	BR_INST_RETIRED.MISPRED	Retired mispredicted branch instructions (precise event).	This event counts the number of retired branch instructions that were mispredicted by the processor. A branch misprediction occurs when the processor predicts that the branch would be taken, but it is not, or vice-versa. Mispredicted branches degrade the performance because the processor starts executing instructions along a wrong path it predicts. When the misprediction is discovered, all the instructions executed in the wrong path must be discarded, and the processor must start again on the correct path.  Using the Profile-Guided Optimization (PGO) features of the Intel® C++ compiler may help reduce branch mispredictions. See the compiler documentation for more information on this feature.

Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)

Event Num.	Umask Value	Event Name	Definition	Description and Comment
				<p>To determine the branch misprediction ratio, divide the BR_INST_RETIREDMISPRED event count by the number of BR_INST_RETIREDAANY event count. To determine the number of mispredicted branches per instruction, divide the number of mispredicted branches by the INST_RETIREDAANY event count. To measure the impact of the branch mispredictions use the event RESOURCE_STALLS.BR_MISS_CLEAR.</p> <p>Tips:</p> <ul style="list-style-type: none"> <li>- See the optimization guide for tips on reducing branch mispredictions.</li> <li>- PGO's purpose is to have straight line code for the most frequent execution paths, reducing branches taken and increasing the "basic block" size, possibly also reducing the code footprint or working-set.</li> </ul>
C4H	0CH	BR_INST_RETIREDTAKEN	Retired taken branch instructions.	This event counts the number of branches retired that were taken.
C4H	0FH	BR_INST_RETIREDAANY1	Retired branch instructions.	This event counts the number of branch instructions retired that were mispredicted. This event is a duplicate of BR_INST_RETIREDMISPRED.
C5H	00H	BR_INST_RETIREDMISPRED	Retired mispredicted branch instructions (precise event).	<p>This event counts the number of retired branch instructions that were mispredicted by the processor. A branch misprediction occurs when the processor predicts that the branch would be taken, but it is not, or vice-versa. Mispredicted branches degrade the performance because the processor starts executing instructions along a wrong path it predicts. When the misprediction is discovered, all the instructions executed in the wrong path must be discarded, and the processor must start again on the correct path.</p> <p>Using the Profile-Guided Optimization (PGO) features of the Intel® C++ compiler may help reduce branch mispredictions. See the compiler documentation for more information on this feature.</p> <p>To determine the branch misprediction ratio, divide the BR_INST_RETIREDMISPRED event count by the number of BR_INST_RETIREDAANY event count. To determine the number of mispredicted branches per instruction, divide the number of mispredicted branches by the INST_RETIREDAANY event count. To measure the impact of the branch mispredictions use the event RESOURCE_STALLS.BR_MISS_CLEAR.</p> <p>Tips:</p> <ul style="list-style-type: none"> <li>- See the optimization guide for tips on reducing branch mispredictions.</li> <li>- PGO's purpose is to have straight line code for the most frequent execution paths, reducing branches taken and increasing the "basic block" size, possibly also reducing the code footprint or working-set.</li> </ul>
C6H	01H	CYCLES_INT_MASKED.CYCLES_INT_MASKED	Cycles during which interrupts are disabled.	This event counts the number of cycles during which interrupts are disabled.
C6H	02H	CYCLES_INT_MASKED.CYCLES_INT_PENDING_AND_MASKED	Cycles during which interrupts are pending and disabled.	This event counts the number of cycles during which there are pending interrupts but interrupts are disabled.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
C7H	01H	SIMD_INST_RETIRED.PACKED_SINGLE	Retired Streaming SIMD Extensions (SSE) packed-single instructions.	This event counts the number of SSE packed-single instructions retired.
C7H	02H	SIMD_INST_RETIRED.SCALAR_SINGLE	Retired Streaming SIMD Extensions (SSE) scalar-single instructions.	This event counts the number of SSE scalar-single instructions retired.
C7H	04H	SIMD_INST_RETIRED.PACKED_DOUBLE	Retired Streaming SIMD Extensions 2 (SSE2) packed-double instructions.	This event counts the number of SSE2 packed-double instructions retired.
C7H	08H	SIMD_INST_RETIRED.SCALAR_DOUBLE	Retired Streaming SIMD Extensions 2 (SSE2) scalar-double instructions.	This event counts the number of SSE2 scalar-double instructions retired.
C7H	10H	SIMD_INST_RETIRED.VECTOR	Retired Streaming SIMD Extensions 2 (SSE2) vector instructions.	This event counts the number of SSE2 vector instructions retired.
C7H	1FH	SIMD_INST_RETIRED.ANY	Retired Streaming SIMD instructions.	This event counts the overall number of SIMD instructions retired. To count each type of SIMD instruction separately, use the following events: SIMD_INST_RETIRED.PACKED_SINGLE SIMD_INST_RETIRED.SCALAR_SINGLE SIMD_INST_RETIRED.PACKED_DOUBLE SIMD_INST_RETIRED.SCALAR_DOUBLE SIMD_INST_RETIRED.VECTOR.
C8H	00H	HW_INT_RCV	Hardware interrupts received.	This event counts the number of hardware interrupts received by the processor. This event will count twice for dual-pipe micro-ops.
CAH	01H	SIMD_COMP_INST_RETIRED.PACKED_SINGLE	Retired computational Streaming SIMD Extensions (SSE) packed-single instructions.	This event counts the number of computational SSE packed-single instructions retired. Computational instructions perform arithmetic computations, like add, multiply and divide. Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	02H	SIMD_COMP_INST_RETIRED.SCALAR_SINGLE	Retired computational Streaming SIMD Extensions (SSE) scalar-single instructions.	This event counts the number of computational SSE scalar-single instructions retired. Computational instructions perform arithmetic computations, like add, multiply and divide. Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	04H	SIMD_COMP_INST_RETIRED.PACKED_DOUBLE	Retired computational Streaming SIMD Extensions 2 (SSE2) packed-double instructions.	This event counts the number of computational SSE2 packed-double instructions retired. Computational instructions perform arithmetic computations, like add, multiply and divide. Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.

**Table 19-29. Performance Events for 45 nm, 32 nm Intel® Atom™ Processors (Contd.)**

Event Num.	Umask Value	Event Name	Definition	Description and Comment
CAH	08H	SIMD_COMP_INST_RETIRED.SCALAR_DOUBLE	Retired computational Streaming SIMD Extensions 2 (SSE2) scalar-double instructions.	This event counts the number of computational SSE2 scalar-double instructions retired. Computational instructions perform arithmetic computations, like add, multiply and divide. Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CBH	01H	MEM_LOAD_RETIRED.L2_HIT	Retired loads that hit the L2 cache (precise event).	This event counts the number of retired load operations that missed the L1 data cache and hit the L2 cache.
CBH	02H	MEM_LOAD_RETIRED.L2_MISS	Retired loads that miss the L2 cache (precise event).	This event counts the number of retired load operations that missed the L2 cache.
CBH	04H	MEM_LOAD_RETIRED.DTLB_MISS	Retired loads that miss the DTLB (precise event).	This event counts the number of retired loads that missed the DTLB. The DTLB miss is not counted if the load operation causes a fault.
CDH	00H	SIMD_ASSIST	SIMD assists invoked.	This event counts the number of SIMD assists invoked. SIMD assists are invoked when an EMMS instruction is executed after MMX™ technology code has changed the MMX state in the floating point stack. For example, these assists are required in the following cases. Streaming SIMD Extensions (SSE) instructions: 1. Denormal input when the DAZ (Denormals Are Zeros) flag is off. 2. Underflow result when the FTZ (Flush To Zero) flag is off.
CEH	00H	SIMD_INSTR_RETIRED	SIMD Instructions retired.	This event counts the number of SIMD instructions that retired.
CFH	00H	SIMD_SAT_INSTR_RETIRED	Saturated arithmetic instructions retired.	This event counts the number of saturated arithmetic SIMD instructions that retired.
E0H	01H	BR_INST_DECODED	Branch instructions decoded.	This event counts the number of branch instructions decoded.
E4H	01H	BOGUS_BR	Bogus branches.	This event counts the number of byte sequences that were mistakenly detected as taken branch instructions. This results in a BACLEAR event and the BTB is flushed. This occurs mainly after task switches.
E6H	01H	BACLEAR.ANY	BACLEARs asserted.	This event counts the number of times the front end is redirected for a branch prediction, mainly when an early branch prediction is corrected by other branch handling mechanisms in the front end. This can occur if the code has many branches such that they cannot be consumed by the branch predictor. Each Baclear asserted costs approximately 7 cycles. The effect on total execution time depends on the surrounding code.

## 19.17 PERFORMANCE MONITORING EVENTS FOR INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Table 19-30 lists model-specific performance events for Intel® Core™ Duo processors. If a model-specific event requires qualification in core specificity, it is indicated in the comment column. Table 19-30 also applies to Intel® Core™ Solo processors; bits in the unit mask corresponding to core-specificity are reserved and should be 00B.

**Table 19-30. Performance Events in Intel® Core™ Solo and Intel® Core™ Duo Processors**

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
03H	LD_Blocks	00H	Load operations delayed due to store buffer blocks. The preceding store may be blocked due to unknown address, unknown data, or conflict due to partial overlap between the load and store.	
04H	SD_Drains	00H	Cycles while draining store buffers.	
05H	Misalign_Mem_Ref	00H	Misaligned data memory references (MOB splits of loads and stores).	
06H	Seg_Reg_Loads	00H	Segment register loads.	
07H	SSE_PrefNta_Ret	00H	SSE software prefetch instruction PREFETCHNTA retired.	
07H	SSE_PrefT1_Ret	01H	SSE software prefetch instruction PREFETCHT1 retired.	
07H	SSE_PrefT2_Ret	02H	SSE software prefetch instruction PREFETCHT2 retired.	
07H	SSE_NTStores_Ret	03H	SSE streaming store instruction retired.	
10H	FP_Comps_Op_Exe	00H	FP computational Instruction executed. FADD, FSUB, FCOM, FMULs, MUL, IMUL, FDIVs, DIV, IDIV, FPREMs, FSQRT are included; but exclude FADD or FMUL used in the middle of a transcendental instruction.	
11H	FP_Assist	00H	FP exceptions experienced microcode assists.	IA32_PMC1 only.
12H	Mul	00H	Multiply operations (a speculative count, including FP and integer multiplies).	IA32_PMC1 only.
13H	Div	00H	Divide operations (a speculative count, including FP and integer divisions).	IA32_PMC1 only.
14H	Cycles_Div_Busy	00H	Cycles the divider is busy.	IA32_PMC0 only.
21H	L2_ADS	00H	L2 Address strobos.	Requires core-specificity.
22H	Dbus_Busy	00H	Core cycle during which data bus was busy (increments by 4).	Requires core-specificity.
23H	Dbus_Busy_Rd	00H	Cycles data bus is busy transferring data to a core (increments by 4).	Requires core-specificity.
24H	L2_Lines_In	00H	L2 cache lines allocated.	Requires core-specificity and HW prefetch qualification.
25H	L2_M_Lines_In	00H	L2 Modified-state cache lines allocated.	Requires core-specificity.
26H	L2_Lines_Out	00H	L2 cache lines evicted.	Requires core-specificity and HW prefetch qualification.
27H	L2_M_Lines_Out	00H	L2 Modified-state cache lines evicted.	Requires core-specificity and HW prefetch qualification.

Table 19-30. Performance Events in Intel® Core™ Solo and Intel® Core™ Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
28H	L2_IFetch	Requires MESI qualification	L2 instruction fetches from instruction fetch unit (includes speculative fetches).	Requires core-specificity.
29H	L2_LD	Requires MESI qualification	L2 cache reads.	Requires core-specificity.
2AH	L2_ST	Requires MESI qualification	L2 cache writes (includes speculation).	Requires core-specificity.
2EH	L2_Rqsts	Requires MESI qualification	L2 cache reference requests.	Requires core-specificity, HW prefetch qualification.
30H	L2_Reject_Cycles	Requires MESI qualification	Cycles L2 is busy and rejecting new requests.	
32H	L2_No_Request_Cycles	Requires MESI qualification	Cycles there is no request to access L2.	
3AH	EST_Trans_All	00H	Any Intel Enhanced SpeedStep(R) Technology transitions.	
3AH	EST_Trans_All	10H	Intel Enhanced SpeedStep Technology frequency transitions.	
3BH	Thermal_Trip	C0H	Duration in a thermal trip based on the current core clock.	Use edge trigger to count occurrence.
3CH	NonHlt_Ref_Cycles	01H	Non-halted bus cycles.	
3CH	Serial_Execution_Cycles	02H	Non-halted bus cycles of this core executing code while the other core is halted.	
40H	DCache_Cache_LD	Requires MESI qualification	L1 cacheable data read operations.	
41H	DCache_Cache_ST	Requires MESI qualification	L1 cacheable data write operations.	
42H	DCache_Cache_Lock	Requires MESI qualification	L1 cacheable lock read operations to invalid state.	
43H	Data_Mem_Ref	01H	L1 data read and writes of cacheable and non-cacheable types.	
44H	Data_Mem_Cache_Ref	02H	L1 data cacheable read and write operations.	
45H	DCache_Repl	0FH	L1 data cache line replacements.	
46H	DCache_M_Repl	00H	L1 data M-state cache line allocated.	
47H	DCache_M_Evict	00H	L1 data M-state cache line evicted.	
48H	DCache_Pend_Miss	00H	Weighted cycles of L1 miss outstanding.	Use Cmask =1 to count duration.
49H	Dtlb_Miss	00H	Data references that missed TLB.	
4BH	SSE_PrefNta_Miss	00H	PREFETCHNTA missed all caches.	
4BH	SSE_PrefT1_Miss	01H	PREFETCHT1 missed all caches.	
4BH	SSE_PrefT2_Miss	02H	PREFETCHT2 missed all caches.	
4BH	SSE_NTStores_Miss	03H	SSE streaming store instruction missed all caches.	
4FH	L1_Pref_Req	00H	L1 prefetch requests due to DCU cache misses.	May overcount if request re-submitted.



**Table 19-30. Performance Events in Intel® Core™ Solo and Intel® Core™ Duo Processors (Contd.)**

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
60H	Bus_Req_Outstanding	00; Requires core-specificity, and agent specificity	Weighted cycles of cacheable bus data read requests. This event counts full-line read request from DCU or HW prefetcher, but not RFO, write, instruction fetches, or others.	Use Cmask =1 to count duration. Use Umask bit 12 to include HWP or exclude HWP separately.
61H	Bus_BNR_Clocks	00H	External bus cycles while BNR asserted.	
62H	Bus_DRDY_Clocks	00H	External bus cycles while DRDY asserted.	Requires agent specificity.
63H	Bus_Locks_Clocks	00H	External bus cycles while bus lock signal asserted.	Requires core specificity.
64H	Bus_Data_Rcv	40H	Number of data chunks received by this processor.	
65H	Bus_Trans_Brd	See comment.	Burst read bus transactions (data or code).	Requires core specificity.
66H	Bus_Trans_RFO	See comment.	Completed read for ownership (RFO) transactions.	Requires agent specificity.
68H	Bus_Trans_Ifetch	See comment.	Completed instruction fetch transactions.	
69H	Bus_Trans_Inval	See comment.	Completed invalidate transactions.	Requires core specificity.
6AH	Bus_Trans_Pwr	See comment.	Completed partial write transactions.	Each transaction counts its address strobe. Retried transaction may be counted more than once.
6BH	Bus_Trans_P	See comment.	Completed partial transactions (include partial read + partial write + line write).	
6CH	Bus_Trans_IO	See comment.	Completed I/O transactions (read and write).	
6DH	Bus_Trans_Def	20H	Completed defer transactions.	Requires core specificity. Retried transaction may be counted more than once.
67H	Bus_Trans_WB	COH	Completed writeback transactions from DCU (does not include L2 writebacks).	Requires agent specificity.
6EH	Bus_Trans_Burst	COH	Completed burst transactions (full line transactions include reads, write, RFO, and writebacks).	Each transaction counts its address strobe.
6FH	Bus_Trans_Mem	COH	Completed memory transactions. This includes Bus_Trans_Burst + Bus_Trans_P+Bus_Trans_Inval.	Retried transaction may be counted more than once.
70H	Bus_Trans_Any	COH	Any completed bus transactions.	
77H	Bus_Snoops	00H	Counts any snoop on the bus.	Requires MESI qualification. Requires agent specificity.
78H	DCU_Snoop_To_Share	01H	DCU snoops to share-state L1 cache line due to L1 misses.	Requires core specificity.
7DH	Bus_Not_In_Use	00H	Number of cycles there is no transaction from the core.	Requires core specificity.
7EH	Bus_Snoop_Stall	00H	Number of bus cycles while bus snoop is stalled.	
80H	ICache_Reads	00H	Number of instruction fetches from ICache, streaming buffers (both cacheable and uncacheable fetches).	

Table 19-30. Performance Events in Intel® Core™ Solo and Intel® Core™ Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
81H	ICache_Misses	00H	Number of instruction fetch misses from ICache, streaming buffers.	
85H	ITLB_Misses	00H	Number of iTLB misses.	
86H	IFU_Mem_Stall	00H	Cycles IFU is stalled while waiting for data from memory.	
87H	ILD_Stall	00H	Number of instruction length decoder stalls (Counts number of LCP stalls).	
88H	Br_Inst_Exec	00H	Branch instruction executed (includes speculation).	
89H	Br_Misssp_Exec	00H	Branch instructions executed and mispredicted at execution (includes branches that do not have prediction or mispredicted).	
8AH	Br_BAC_Misssp_Exec	00H	Branch instructions executed that were mispredicted at front end.	
8BH	Br_Cnd_Exec	00H	Conditional branch instructions executed.	
8CH	Br_Cnd_Misssp_Exec	00H	Conditional branch instructions executed that were mispredicted.	
8DH	Br_Ind_Exec	00H	Indirect branch instructions executed.	
8EH	Br_Ind_Misssp_Exec	00H	Indirect branch instructions executed that were mispredicted.	
8FH	Br_Ret_Exec	00H	Return branch instructions executed.	
90H	Br_Ret_Misssp_Exec	00H	Return branch instructions executed that were mispredicted.	
91H	Br_Ret_BAC_Misssp_Exec	00H	Return branch instructions executed that were mispredicted at the front end.	
92H	Br_Call_Exec	00H	Return call instructions executed.	
93H	Br_Call_Misssp_Exec	00H	Return call instructions executed that were mispredicted.	
94H	Br_Ind_Call_Exec	00H	Indirect call branch instructions executed.	
A2H	Resource_Stall	00H	Cycles while there is a resource related stall (renaming, buffer entries) as seen by allocator.	
B0H	MMX_Instr_Exec	00H	Number of MMX instructions executed (does not include MOVQ and MOVD stores).	
B1H	SIMD_Int_Sat_Exec	00H	Number of SIMD Integer saturating instructions executed.	
B3H	SIMD_Int_Pmul_Exec	01H	Number of SIMD Integer packed multiply instructions executed.	
B3H	SIMD_Int_Psft_Exec	02H	Number of SIMD Integer packed shift instructions executed.	
B3H	SIMD_Int_Pck_Exec	04H	Number of SIMD Integer pack operations instruction executed.	
B3H	SIMD_Int_Upck_Exec	08H	Number of SIMD Integer unpack instructions executed.	
B3H	SIMD_Int_Plog_Exec	10H	Number of SIMD Integer packed logical instructions executed.	
B3H	SIMD_Int_Pari_Exec	20H	Number of SIMD Integer packed arithmetic instructions executed.	

**Table 19-30. Performance Events in Intel® Core™ Solo and Intel® Core™ Duo Processors (Contd.)**

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
C0H	Instr_Ret	00H	Number of instruction retired (Macro fused instruction count as 2).	
C1H	FP_Comp_Instr_Ret	00H	Number of FP compute instructions retired (X87 instruction or instruction that contains X87 operations).	Use IA32_PMC0 only.
C2H	Uops_Ret	00H	Number of micro-ops retired (include fused uops).	
C3H	SMC_Detected	00H	Number of times self-modifying code condition detected.	
C4H	Br_Instr_Ret	00H	Number of branch instructions retired.	
C5H	Br_MisPred_Ret	00H	Number of mispredicted branch instructions retired.	
C6H	Cycles_Int_Masked	00H	Cycles while interrupt is disabled.	
C7H	Cycles_Int_Pedning_Masked	00H	Cycles while interrupt is disabled and interrupts are pending.	
C8H	HW_Int_Rx	00H	Number of hardware interrupts received.	
C9H	Br_Taken_Ret	00H	Number of taken branch instruction retired.	
CAH	Br_MisPred_Taken_Ret	00H	Number of taken and mispredicted branch instructions retired.	
CCH	MMX_FP_Trans	00H	Number of transitions from MMX to X87.	
CCH	FP_MMX_Trans	01H	Number of transitions from X87 to MMX.	
CDH	MMX_Assist	00H	Number of EMMS executed.	
CEH	MMX_Instr_Ret	00H	Number of MMX instruction retired.	
D0H	Instr_Decoded	00H	Number of instruction decoded.	
D7H	ESP_Uops	00H	Number of ESP folding instruction decoded.	
D8H	SIMD_FP_SP_Ret	00H	Number of SSE/SSE2 single precision instructions retired (packed and scalar).	
D8H	SIMD_FP_SP_S_Ret	01H	Number of SSE/SSE2 scalar single precision instructions retired.	
D8H	SIMD_FP_DP_P_Ret	02H	Number of SSE/SSE2 packed double precision instructions retired.	
D8H	SIMD_FP_DP_S_Ret	03H	Number of SSE/SSE2 scalar double precision instructions retired.	
D8H	SIMD_Int_128_Ret	04H	Number of SSE2 128 bit integer instructions retired.	
D9H	SIMD_FP_SP_P_Comp_Ret	00H	Number of SSE/SSE2 packed single precision compute instructions retired (does not include AND, OR, XOR).	
D9H	SIMD_FP_SP_S_Comp_Ret	01H	Number of SSE/SSE2 scalar single precision compute instructions retired (does not include AND, OR, XOR).	
D9H	SIMD_FP_DP_P_Comp_Ret	02H	Number of SSE/SSE2 packed double precision compute instructions retired (does not include AND, OR, XOR).	
D9H	SIMD_FP_DP_S_Comp_Ret	03H	Number of SSE/SSE2 scalar double precision compute instructions retired (does not include AND, OR, XOR).	

**Table 19-30. Performance Events in Intel® Core™ Solo and Intel® Core™ Duo Processors (Contd.)**

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
DAH	Fused_Uops_Ret	00H	All fused uops retired.	
DAH	Fused_Ld_Uops_Ret	01H	Fused load uops retired.	
DAH	Fused_St_Uops_Ret	02H	Fused store uops retired.	
DBH	Unfusion	00H	Number of unfusion events in the ROB (due to exception).	
E0H	Br_Instr_Decoded	00H	Branch instructions decoded.	
E2H	BTB_Misses	00H	Number of branches the BTB did not produce a prediction.	
E4H	Br_Bogus	00H	Number of bogus branches.	
E6H	BAClears	00H	Number of BAClears asserted.	
F0H	Pref_Rqsts_Up	00H	Number of hardware prefetch requests issued in forward streams.	
F8H	Pref_Rqsts_Dn	00H	Number of hardware prefetch requests issued in backward streams.	

## 19.18 PENTIUM® 4 AND INTEL® XEON® PROCESSOR PERFORMANCE MONITORING EVENTS

Tables 19-31, 19-32 and 19-33 list performance monitoring events that can be counted or sampled on processors based on Intel NetBurst® microarchitecture. Table 19-31 lists the non-retirement events, and Table 19-32 lists the at-retirement events. Tables 19-34, 19-35, and 19-36 describes three sets of parameters that are available for three of the at-retirement counting events defined in Table 19-32. Table 19-37 shows which of the non-retirement and at retirement events are logical processor specific (TS) (see Section 18.6.4.4, "Performance Monitoring Events") and which are non-logical processor specific (TI).

Some of the Pentium 4 and Intel Xeon processor performance monitoring events may be available only to specific models. The performance monitoring events listed in Tables 19-31 and 19-32 apply to processors with CPUID signature that matches family encoding 15, model encoding 0, 1, 2 3, 4, or 6. Table applies to processors with a CPUID signature that matches family encoding 15, model encoding 3, 4 or 6.

The functionality of performance monitoring events in Pentium 4 and Intel Xeon processors is also available when IA-32e mode is enabled.

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting**

Event Name	Event Parameters	Parameter Value	Description
TC_deliver_mode			This event counts the duration (in clock cycles) of the operating modes of the trace cache and decode engine in the processor package. The mode is specified by one or more of the event mask bits.
	ESCR restrictions	MSR_TC_ESCR0 MSR_TC_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	01H	ESCR[31:25]

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask	Bit	ESCR[24:9]
		0: DD	Both logical processors are in deliver mode.
		1: DB	Logical processor 0 is in deliver mode and logical processor 1 is in build mode.
		2: DI	Logical processor 0 is in deliver mode and logical processor 1 is either halted, under a machine clear condition or transitioning to a long microcode flow.
		3: BD	Logical processor 0 is in build mode and logical processor 1 is in deliver mode.
	ESCR Event Mask	4: BB	Both logical processors are in build mode.
		5: BI	Logical processor 0 is in build mode and logical processor 1 is either halted, under a machine clear condition or transitioning to a long microcode flow.
		6: ID	Logical processor 0 is either halted, under a machine clear condition or transitioning to a long microcode flow. Logical processor 1 is in deliver mode.
		7: IB	Logical processor 0 is either halted, under a machine clear condition or transitioning to a long microcode flow. Logical processor 1 is in build mode.
		CCCR Select	01H
	Event Specific Notes		If only one logical processor is available from a physical processor package, the event mask should be interpreted as logical processor 1 is halted. Event mask bit 2 was previously known as "DELIVER", bit 5 was previously known as "BUILD".
BPU_fetch_request			This event counts instruction fetch requests of specified request type by the Branch Prediction unit. Specify one or more mask bits to qualify the request type(s).
	ESCR restrictions	MSR_BPU_ESCR0 MSR_BPU_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask	Bit 0: TCMISS	ESCR[24:9] Trace cache lookup miss
	CCCR Select	00H	CCCR[15:13]

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
ITLB_reference			This event counts translations using the Instruction Translation Look-aside Buffer (ITLB).
	ESCR restrictions	MSR_ITLB_ESCR0 MSR_ITLB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	18H	ESCR[31:25]
	ESCR Event Mask	Bit 0: HIT 1: MISS 2: HIT_UC	ESCR[24:9]  ITLB hit ITLB miss Uncacheable ITLB hit
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		All page references regardless of the page size are looked up as actual 4-KByte pages. Use the page_walk_type event with the ITMISS mask for a more conservative count.
memory_cancel			This event counts the canceling of various type of request in the Data cache Address Control unit (DAC). Specify one or more mask bits to select the type of requests that are canceled.
	ESCR restrictions	MSR_DAC_ESCR0 MSR_DAC_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	Bit 2: ST_RB_FULL 3: 64K_CONF	ESCR[24:9]  Replayed because no store request buffer is available. Conflicts due to 64-KByte aliasing.
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		All_CACHE_MISS includes uncacheable memory in count.
memory_complete			This event counts the completion of a load split, store split, uncacheable (UC) split, or UC load. Specify one or more mask bits to select the operations to be counted.
	ESCR restrictions	MSR_SAAT_ESCR0 MSR_SAAT_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	08H	ESCR[31:25]

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask	Bit 0: LSC 1: SSC	ESCR[24:9]  Load split completed, excluding UC/WC loads. Any split stores completed.
	CCCR Select	02H	CCCR[15:13]
load_port_replay			This event counts replayed events at the load port. Specify one or more mask bits to select the cause of the replay.
	ESCR restrictions	MSR_SAAT_ESCRO MSR_SAAT_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	04H	ESCR[31:25]
	ESCR Event Mask	Bit 1: SPLIT_LD	ESCR[24:9] Split load.
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		Must use ESCR1 for at-retirement counting.
store_port_replay			This event counts replayed events at the store port. Specify one or more mask bits to select the cause of the replay.
	ESCR restrictions	MSR_SAAT_ESCRO MSR_SAAT_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	05H	ESCR[31:25]
	ESCR Event Mask	Bit 1: SPLIT_ST	ESCR[24:9] Split store
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		Must use ESCR1 for at-retirement counting.
MOB_load_replay			This event triggers if the memory order buffer (MOB) caused a load operation to be replayed. Specify one or more mask bits to select the cause of the replay.
	ESCR restrictions	MSR_MOB_ESCRO MSR_MOB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	03H	ESCR[31:25]

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask	Bit 1: NO_STA  3: NO_STD	ESCR[24:9]  Replayed because of unknown store address. Replayed because of unknown store data.
		4: PARTIAL_DATA  5: UNALGN_ADDR	Replayed because of partially overlapped data access between the load and store operations. Replayed because the lower 4 bits of the linear address do not match between the load and store operations.
	CCCR Select	02H	CCCR[15:13]
page_walk_type			This event counts various types of page walks that the page miss handler (PMH) performs.
	ESCR restrictions	MSR_PMH_ESCR0 MSR_PMH_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	01H	ESCR[31:25]
	ESCR Event Mask	Bit 0: DTMISS  1: ITMISS	ESCR[24:9]  Page walk for a data TLB miss (either load or store). Page walk for an instruction TLB miss.
	CCCR Select	04H	CCCR[15:13]
BSQ_cache_reference			This event counts cache references (2nd level cache or 3rd level cache) as seen by the bus unit.  Specify one or more mask bit to select an access according to the access type (read type includes both load and RFO, write type includes writebacks and evictions) and the access result (hit, misses).
	ESCR restrictions	MSR_BSU_ESCR0 MSR_BSU_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	0CH	ESCR[31:25]



**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
		Bit 0: RD_2ndL_HITS 1: RD_2ndL_HITE 2: RD_2ndL_HITM 3: RD_3rdL_HITS 4: RD_3rdL_HITE 5: RD_3rdL_HITM	ESCR[24:9]  Read 2nd level cache hit Shared (includes load and RFO).  Read 2nd level cache hit Exclusive (includes load and RFO).  Read 2nd level cache hit Modified (includes load and RFO).  Read 3rd level cache hit Shared (includes load and RFO).  Read 3rd level cache hit Exclusive (includes load and RFO). Read 3rd level cache hit Modified (includes load and RFO).
	ESCR Event Mask	8: RD_2ndL_MISS 9: RD_3rdL_MISS 10: WR_2ndL_MISS	Read 2nd level cache miss (includes load and RFO).  Read 3rd level cache miss (includes load and RFO).  A Writeback lookup from DAC misses the 2nd level cache (unlikely to happen).
	CCCR Select	07H	CCCR[15:13]
	Event Specific Notes		1: The implementation of this event in current Pentium 4 and Xeon processors treats either a load operation or a request for ownership (RFO) request as a "read" type operation.  2: Currently this event causes both over and undercounting by as much as a factor of two due to an erratum.  3: It is possible for a transaction that is started as a prefetch to change the transaction's internal status, making it no longer a prefetch. or change the access result status (hit, miss) as seen by this event.
IOQ_allocation			This event counts the various types of transactions on the bus. A count is generated each time a transaction is allocated into the IOQ that matches the specified mask bits. An allocated entry can be a sector (64 bytes) or a chunks of 8 bytes.  Requests are counted once per retry. The event mask bits constitute 4 bit fields. A transaction type is specified by interpreting the values of each bit field.  Specify one or more event mask bits in a bit field to select the value of the bit field.  Each field (bits 0-4 are one field) are independent of and can be ORed with the others. The request type field is further combined with bit 5 and 6 to form a binary expression. Bits 7 and 8 form a bit field to specify the memory type of the target address.  Bits 13 and 14 form a bit field to specify the source agent of the request. Bit 15 affects read operation only. The event is triggered by evaluating the logical expression: (((Request type) OR Bit 5 OR Bit 6) OR (Memory type)) AND (Source agent).

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	ESCR restrictions	MSR_FSB_ESCR0, MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1; ESCR1: 2, 3	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask	Bits 0-4 (single field) 5: ALL_READ 6: ALL_WRITE 7: MEM_UC 8: MEM_WC 9: MEM_WT 10: MEM_WP 11: MEM_WB 13: OWN 14: OTHER 15: PREFETCH	ESCR[24:9]  Bus request type (use 00001 for invalid or default). Count read entries. Count write entries. Count UC memory access entries. Count WC memory access entries. Count write-through (WT) memory access entries. Count write-protected (WP) memory access entries. Count WB memory access entries. Count all store requests driven by processor, as opposed to other processor or DMA. Count all requests driven by other processors or DMA. Include HW and SW prefetch requests in the count.
	CCCR Select	06H	CCCR[15:13]
	Event Specific Notes		<p>1: If PREFETCH bit is cleared, sectors fetched using prefetch are excluded in the counts. If PREFETCH bit is set, all sectors or chunks read are counted.</p> <p>2: Specify the edge trigger in CCCR to avoid double counting.</p> <p>3: The mapping of interpreted bit field values to transaction types may differ with different processor model implementations of the Pentium 4 processor family. Applications that program performance monitoring events should use CPUID to determine processor models when using this event. The logic equations that trigger the event are model-specific (see 4a and 4b below).</p> <p>4a: For Pentium 4 and Xeon Processors starting with CPUID Model field encoding equal to 2 or greater, this event is triggered by evaluating the logical expression ((Request type) and (Bit 5 or Bit 6) and (Memory type) and (Source agent)).</p> <p>4b: For Pentium 4 and Xeon Processors with CPUID Model field encoding less than 2, this event is triggered by evaluating the logical expression [((Request type) or Bit 5 or Bit 6) or (Memory type)] and (Source agent). Note that event mask bits for memory type are ignored if either ALL_READ or ALL_WRITE is specified.</p> <p>5: This event is known to ignore CPL in early implementations of Pentium 4 and Xeon Processors. Both user requests and OS requests are included in the count. This behavior is fixed starting with Pentium 4 and Xeon Processors with CPUID signature F27H (Family 15, Model 2, Stepping 7).</p>

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
			<p>6: For write-through (WT) and write-protected (WP) memory types, this event counts reads as the number of 64-byte sectors. Writes are counted by individual chunks.</p> <p>7: For uncacheable (UC) memory types, this event counts the number of 8-byte chunks allocated.</p> <p>8: For Pentium 4 and Xeon Processors with CPUID Signature less than F27H, only MSR_FSB_ESCR0 is available.</p>
IOQ_active_entries			<p>This event counts the number of entries (clipped at 15) in the IOQ that are active. An allocated entry can be a sector (64 bytes) or a chunks of 8 bytes.</p> <p>The event must be programmed in conjunction with IOQ_allocation. Specify one or more event mask bits to select the transactions that is counted.</p>
	ESCR restrictions	MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR1: 2, 3	
	ESCR Event Select	01AH	ESCR[30:25]
	ESCR Event Mask	<p>Bits</p> <p>0-4 (single field)</p> <p>5: ALL_READ</p> <p>6: ALL_WRITE</p> <p>7: MEM_UC</p> <p>8: MEM_WC</p> <p>9: MEM_WT</p> <p>10: MEM_WP</p> <p>11: MEM_WB</p> <p>13: OWN</p> <p>14: OTHER</p> <p>15: PREFETCH</p>	<p>ESCR[24:9]</p> <p>Bus request type (use 00001 for invalid or default). Count read entries.</p> <p>Count write entries.</p> <p>Count UC memory access entries.</p> <p>Count WC memory access entries.</p> <p>Count write-through (WT) memory access entries.</p> <p>Count write-protected (WP) memory access entries.</p> <p>Count WB memory access entries.</p> <p>Count all store requests driven by processor, as opposed to other processor or DMA.</p> <p>Count all requests driven by other processors or DMA.</p> <p>Include HW and SW prefetch requests in the count.</p>
	CCCR Select	06H	CCCR[15:13]
	Event Specific Notes		<p>1: Specified desired mask bits in ESCR0 and ESCR1.</p> <p>2: See the ioq_allocation event for descriptions of the mask bits.</p> <p>3: Edge triggering should not be used when counting cycles.</p> <p>4: The mapping of interpreted bit field values to transaction types may differ across different processor model implementations of the Pentium 4 processor family. Applications that programs performance monitoring events should use the CPUID instruction to detect processor models when using this event. The logical expression that triggers this event as describe below:</p> <p>5a:For Pentium 4 and Xeon Processors starting with CPUID MODEL field encoding equal to 2 or greater, this event is triggered by evaluating the logical expression ((Request type) and (Bit 5 or Bit 6) and (Memory type) and (Source agent)).</p>

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
			<p>5b: For Pentium 4 and Xeon Processors starting with CPUID MODEL field encoding less than 2, this event is triggered by evaluating the logical expression [((Request type) or Bit 5 or Bit 6) or (Memory type)] and (Source agent). Event mask bits for memory type are ignored if either ALL_READ or ALL_WRITE is specified.</p> <p>5c: This event is known to ignore CPL in the current implementations of Pentium 4 and Xeon Processors Both user requests and OS requests are included in the count.</p> <p>6: An allocated entry can be a full line (64 bytes) or in individual chunks of 8 bytes.</p>
FSB_data_activity			This event increments once for each DRDY or DBSY event that occurs on the front side bus. The event allows selection of a specific DRDY or DBSY event.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	17H	ESCR[31:25]
	ESCR Event Mask	Bit 0: DRDY_DRV  1: DRDY_OWN  2: DRDY_OTHER  3: DBSY_DRV  4: DBSY_OWN	ESCR[24:9]  Count when this processor drives data onto the bus - includes writes and implicit writebacks. Asserted two processor clock cycles for partial writes and 4 processor clocks (usually in consecutive bus clocks) for full line writes.  Count when this processor reads data from the bus - includes loads and some PIC transactions. Asserted two processor clock cycles for partial reads and 4 processor clocks (usually in consecutive bus clocks) for full line reads. Count DRDY events that we drive. Count DRDY events sampled that we own.  Count when data is on the bus but not being sampled by the processor. It may or may not be being driven by this processor. Asserted two processor clock cycles for partial transactions and 4 processor clocks (usually in consecutive bus clocks) for full line transactions.  Count when this processor reserves the bus for use in the next bus cycle in order to drive data. Asserted for two processor clock cycles for full line writes and not at all for partial line writes. May be asserted multiple times (in consecutive bus clocks) if we stall the bus waiting for a cache lock to complete.  Count when some agent reserves the bus for use in the next bus cycle to drive data that this processor will sample. Asserted for two processor clock cycles for full line writes and not at all for partial line writes. May be asserted multiple times (all one bus clock apart) if we stall the bus for some reason.

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description	
		5:DBSY_OTHER	Count when some agent reserves the bus for use in the next bus cycle to drive data that this processor will NOT sample. It may or may not be being driven by this processor.  Asserted two processor clock cycles for partial transactions and 4 processor clocks (usually in consecutive bus clocks) for full line transactions.	
	CCCR Select	06H	CCCR[15:13]	
	Event Specific Notes		Specify edge trigger in the CCCR MSR to avoid double counting. DRDY_OWN and DRDY_OTHER are mutually exclusive; similarly for DBSY_OWN and DBSY_OTHER.	
BSQ_allocation			This event counts allocations in the Bus Sequence Unit (BSQ) according to the specified mask bit encoding. The event mask bits consist of four sub-groups: <ul style="list-style-type: none"> <li>▪ Request type.</li> <li>▪ Request length.</li> <li>▪ Memory type.</li> <li>▪ Sub-group consisting mostly of independent bits (bits 5, 6, 7, 8, 9, and 10).</li> </ul> Specify an encoding for each sub-group.	
	ESCR restrictions	MSR_BSU_ESCR0		
	Counter numbers per ESCR	ESCR0: 0, 1		
	ESCR Event Select	05H	ESCR[31:25]	
	ESCR Event Mask	Bit		ESCR[24:9]
		0: REQ_TYPE0 1: REQ_TYPE1		Request type encoding (bit 0 and 1) are: 0 - Read (excludes read invalidate). 1 - Read invalidate. 2 - Write (other than writebacks). 3 - Writeback (evicted from cache). (public)
	2: REQ_LEN0 3: REQ_LEN1		Request length encoding (bit 2, 3) are: 0 - 0 chunks 1 - 1 chunks 3 - 8 chunks	
	5: REQ_IO_TYPE		Request type is input or output.	
	6: REQ_LOCK_TYPE		Request type is bus lock.	
	7: REQ_CACHE_TYPE		Request type is cacheable.	
	8: REQ_SPLIT_TYPE		Request type is a bus 8-byte chunk split across 8-byte boundary.	
	9: REQ_DEM_TYPE		Request type is a demand if set. Request type is HW.SW prefetch if 0.	
	10: REQ_ORD_TYPE		Request is an ordered type.	

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
		11: MEM_TYPE0 12: MEM_TYPE1 13: MEM_TYPE2	Memory type encodings (bit 11-13) are: 0 - UC 1 - WC 4 - WT 5 - WP 6 - WB
	CCCR Select	07H	CCCR[15:13]
	Event Specific Notes		<p>1: Specify edge trigger in CCCR to avoid double counting.</p> <p>2: A writebacks to 3rd level cache from 2nd level cache counts as a separate entry, this is in addition to the entry allocated for a request to the bus.</p> <p>3: A read request to WB memory type results in a request to the 64-byte sector, containing the target address, followed by a prefetch request to an adjacent sector.</p> <p>4: For Pentium 4 and Xeon processors with CPUID model encoding value equals to 0 and 1, an allocated BSQ entry includes both the demand sector and prefetched 2nd sector.</p> <p>5: An allocated BSQ entry for a data chunk is any request less than 64 bytes.</p> <p>6a: This event may undercount for requests of split type transactions if the data address straddled across modulo-64 byte boundary.</p> <p>6b: This event may undercount for requests of read request of 16-byte operands from WC or UC address.</p> <p>6c: This event may undercount WC partial requests originated from store operands that are dwords.</p>
bsq_active_entries			<p>This event represents the number of BSQ entries (clipped at 15) currently active (valid) which meet the subevent mask criteria during allocation in the BSQ. Active request entries are allocated on the BSQ until de-allocated.</p> <p>De-allocation of an entry does not necessarily imply the request is filled. This event must be programmed in conjunction with BSQ_allocation. Specify one or more event mask bits to select the transactions that is counted.</p>
	ESCR restrictions	ESCR1	
	Counter numbers per ESCR	ESCR1: 2, 3	
	ESCR Event Select	06H	ESCR[30:25]
	ESCR Event Mask		ESCR[24:9]
	CCCR Select	07H	CCCR[15:13]
	Event Specific Notes		<p>1: Specified desired mask bits in ESCR0 and ESCR1.</p> <p>2: See the BSQ_allocation event for descriptions of the mask bits.</p> <p>3: Edge triggering should not be used when counting cycles.</p> <p>4: This event can be used to estimate the latency of a transaction from allocation to de-allocation in the BSQ. The latency observed by BSQ_allocation includes the latency of FSB, plus additional overhead.</p>

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
			<p>5: Additional overhead may include the time it takes to issue two requests (the sector by demand and the adjacent sector via prefetch). Since adjacent sector prefetches have lower priority than demand fetches, on a heavily used system there is a high probability that the adjacent sector prefetch will have to wait until the next bus arbitration.</p> <p>6: For Pentium 4 and Xeon processors with CPUID model encoding value less than 3, this event is updated every clock.</p> <p>7: For Pentium 4 and Xeon processors with CPUID model encoding value equals to 3 or 4, this event is updated every other clock.</p>
SSE_input_assist			This event counts the number of times an assist is requested to handle problems with input operands for SSE/SSE2/SSE3 operations; most notably denormal source operands when the DAZ bit is not set. Set bit 15 of the event mask to use this event.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	34H	ESCR[31:25]
	ESCR Event Mask	15: ALL	ESCR[24:9] Count assists for SSE/SSE2/SSE3 $\mu$ ops.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		<p>1: Not all requests for assists are actually taken. This event is known to overcount in that it counts requests for assists from instructions on the non-retired path that do not incur a performance penalty. An assist is actually taken only for non-bogus <math>\mu</math>ops. Any appreciable counts for this event are an indication that the DAZ or FTZ bit should be set and/or the source code should be changed to eliminate the condition.</p> <p>2: Two common situations for an SSE/SSE2/SSE3 operation needing an assist are: (1) when a denormal constant is used as an input and the Denormals-Are-Zero (DAZ) mode is not set, (2) when the input operand uses the underflowed result of a previous SSE/SSE2/SSE3 operation and neither the DAZ nor Flush-To-Zero (FTZ) modes are set.</p> <p>3: Enabling the DAZ mode prevents SSE/SSE2/SSE3 operations from needing assists in the first situation. Enabling the FTZ mode prevents SSE/SSE2/SSE3 operations from needing assists in the second situation.</p>
packed_SP_uop			This event increments for each packed single-precision $\mu$ op, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	08H	ESCR[31:25]

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all $\mu$ ops operating on packed single-precision operands.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		1: If an instruction contains more than one packed SP $\mu$ ops, each packed SP $\mu$ op that is specified by the event mask will be counted. 2: This metric counts instances of packed memory $\mu$ ops in a repeat move string.
packed_DP_uop			This event increments for each packed double-precision $\mu$ op, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCRO MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCRO: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	OCH	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all $\mu$ ops operating on packed double-precision operands.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one packed DP $\mu$ ops, each packed DP $\mu$ op that is specified by the event mask will be counted.
scalar_SP_uop			This event increments for each scalar single-precision $\mu$ op, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCRO MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCRO: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	OAH	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all $\mu$ ops operating on scalar single-precision operands.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one scalar SP $\mu$ ops, each scalar SP $\mu$ op that is specified by the event mask will be counted.
scalar_DP_uop			This event increments for each scalar double-precision $\mu$ op, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCRO MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCRO: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	0EH	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all $\mu$ ops operating on scalar double-precision operands.
	CCCR Select	01H	CCCR[15:13]



**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		If an instruction contains more than one scalar DP $\mu$ ops, each scalar DP $\mu$ op that is specified by the event mask is counted.
64bit_MMX_uop			This event increments for each MMX instruction, which operate on 64-bit SIMD operands.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all $\mu$ ops operating on 64-bit SIMD integer operands in memory or MMX registers.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one 64-bit MMX $\mu$ ops, each 64-bit MMX $\mu$ op that is specified by the event mask will be counted.
128bit_MMX_uop			This event increments for each integer SIMD SSE2 instruction, which operate on 128-bit SIMD operands.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	1AH	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all $\mu$ ops operating on 128-bit SIMD integer operands in memory or XMM registers.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one 128-bit MMX $\mu$ ops, each 128-bit MMX $\mu$ op that is specified by the event mask will be counted.
x87_FP_uop			This event increments for each x87 floating-point $\mu$ op, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	04H	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all x87 FP $\mu$ ops.
	CCCR Select	01H	CCCR[15:13]

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		1: If an instruction contains more than one x87 FP $\mu$ ops, each x87 FP $\mu$ op that is specified by the event mask will be counted. 2: This event does not count x87 FP $\mu$ op for load, store, move between registers.
TC_misc			This event counts miscellaneous events detected by the TC. The counter will count twice for each occurrence.
	ESCR restrictions	MSR_TC_ESCR0 MSR_TC_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	06H	ESCR[31:25]
	CCCR Select	01H	CCCR[15:13]
	ESCR Event Mask	Bit 4: FLUSH	ESCR[24:9] Number of flushes
global_power_events			This event accumulates the time during which a processor is not stopped.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	013H	ESCR[31:25]
	ESCR Event Mask	Bit 0: Running	ESCR[24:9] The processor is active (includes the handling of HLT STPCLK and throttling.
	CCCR Select	06H	CCCR[15:13]
tc_ms_xfer			This event counts the number of times that uop delivery changed from TC to MS ROM.
	ESCR restrictions	MSR_MS_ESCR0 MSR_MS_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	05H	ESCR[31:25]
	ESCR Event Mask	Bit 0: CISC	ESCR[24:9] A TC to MS transfer occurred.
	CCCR Select	0H	CCCR[15:13]
uop_queue_writes			This event counts the number of valid uops written to the uop queue. Specify one or more mask bits to select the source type of writes.
	ESCR restrictions	MSR_MS_ESCR0 MSR_MS_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Select	09H	ESCR[31:25]
	ESCR Event Mask	Bit 0: FROM_TC_BUILD 1: FROM_TC_DELIVER 2: FROM_ROM	ESCR[24:9]  The uops being written are from TC build mode.  The uops being written are from TC deliver mode. The uops being written are from microcode ROM.
	CCCR Select	0H	CCCR[15:13]
retired_mispred_branch_type			This event counts retiring mispredicted branches by type.
	ESCR restrictions	MSR_TBPU_ESCR0 MSR_TBPU_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	05H	ESCR[30:25]
	ESCR Event Mask	Bit 1: CONDITIONAL 2: CALL	ESCR[24:9]  Conditional jumps. Indirect call branches.
		3: RETURN 4: INDIRECT	Return branches. Returns, indirect calls, or indirect jumps.
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		This event may overcount conditional branches if: <ul style="list-style-type: none"> <li>▪ Mispredictions cause the trace cache and delivery engine to build new traces.</li> <li>▪ When the processor’s pipeline is being cleared.</li> </ul>
retired_branch_type			This event counts retiring branches by type. Specify one or more mask bits to qualify the branch by its type.
	ESCR restrictions	MSR_TBPU_ESCR0 MSR_TBPU_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	04H	ESCR[30:25]
	ESCR Event Mask	Bit 1: CONDITIONAL 2: CALL 3: RETURN 4: INDIRECT	ESCR[24:9]  Conditional jumps. Direct or indirect calls. Return branches. Returns, indirect calls, or indirect jumps.
	CCCR Select	02H	CCCR[15:13]

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		This event may overcount conditional branches if : <ul style="list-style-type: none"> <li>▪ Mispredictions cause the trace cache and delivery engine to build new traces.</li> <li>▪ When the processor's pipeline is being cleared.</li> </ul>
resource_stall			This event monitors the occurrence or latency of stalls in the Allocator.
	ESCR restrictions	MSR_ALF_ESCR0 MSR_ALF_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	01H	ESCR[30:25]
	Event Masks	Bit 5: SBFULL	ESCR[24:9]  A Stall due to lack of store buffers.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.
WC_Buffer			This event counts Write Combining Buffer operations that are selected by the event mask.
	ESCR restrictions	MSR_DAC_ESCR0 MSR_DAC_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	05H	ESCR[30:25]
	Event Masks	Bit 0: WCB_EVICTS	ESCR[24:9]  WC Buffer evictions of all causes.
		1: WCB_FULL_EVICT	WC Buffer eviction: no WC buffer is available.
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		This event is useful for detecting the subset of 64K aliasing cases that are more costly (i.e. 64K aliasing cases involving stores) as long as there are no significant contributions due to write combining buffer full or hit-modified conditions.
b2b_cycles			This event can be configured to count the number back-to-back bus cycles using sub-event mask bits 1 through 6.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	016H	ESCR[30:25]
	Event Masks	Bit	ESCR[24:9]

**Table 19-31. Performance Monitoring Events Supported by Intel NetBurst® Microarchitecture for Non-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.
bnr			This event can be configured to count bus not ready conditions using sub-event mask bits 0 through 2.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	08H	ESCR[30:25]
	Event Masks	Bit	ESCR[24:9]
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.
snoop			This event can be configured to count snoop hit modified bus traffic using sub-event mask bits 2, 6 and 7.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	06H	ESCR[30:25]
	Event Masks	Bit	ESCR[24:9]
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.
Response			This event can be configured to count different types of responses using sub-event mask bits 1,2, 8, and 9.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	04H	ESCR[30:25]
	Event Masks	Bit	ESCR[24:9]
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.

**Table 19-32. Performance Monitoring Events For Intel NetBurst® Microarchitecture  
for At-Retirement Counting**

Event Name	Event Parameters	Parameter Value	Description
front_end_event			This event counts the retirement of tagged $\mu$ ops, which are specified through the front-end tagging mechanism. The event mask specifies bogus or non-bogus $\mu$ ops.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	08H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS 1: BOGUS	ESCR[24:9]  The marked $\mu$ ops are not bogus. The marked $\mu$ ops are bogus.
	CCCR Select	05H	CCCR[15:13]
	Can Support PEBS	Yes	
	Require Additional MSRs for tagging	Selected ESCRs and/or MSR_TC_PRECISE_EVENT	See list of metrics supported by Front_end tagging in Table A-3
execution_event			This event counts the retirement of tagged $\mu$ ops, which are specified through the execution tagging mechanism.  The event mask allows from one to four types of $\mu$ ops to be specified as either bogus or non-bogus $\mu$ ops to be tagged.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	0CH	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS0 1: NBOGUS1 2: NBOGUS2 3: NBOGUS3 4: BOGUS0 5: BOGUS1 6: BOGUS2 7: BOGUS3	ESCR[24:9]  The marked $\mu$ ops are not bogus. The marked $\mu$ ops are not bogus. The marked $\mu$ ops are not bogus. The marked $\mu$ ops are not bogus. The marked $\mu$ ops are bogus. The marked $\mu$ ops are bogus. The marked $\mu$ ops are bogus. The marked $\mu$ ops are bogus.
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		Each of the 4 slots to specify the bogus/non-bogus $\mu$ ops must be coordinated with the 4 TagValue bits in the ESCR (for example, NBOGUS0 must accompany a '1' in the lowest bit of the TagValue field in ESCR, NBOGUS1 must accompany a '1' in the next but lowest bit of the TagValue field).
	Can Support PEBS	Yes	

**Table 19-32. Performance Monitoring Events For Intel NetBurst® Microarchitecture for At-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	Require Additional MSRs for tagging	An ESCR for an upstream event	See list of metrics supported by execution tagging in Table A-4.
replay_event			This event counts the retirement of tagged $\mu$ ops, which are specified through the replay tagging mechanism. The event mask specifies bogus or non-bogus $\mu$ ops.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	09H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS 1: BOGUS	ESCR[24:9]  The marked $\mu$ ops are not bogus. The marked $\mu$ ops are bogus.
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		Supports counting tagged $\mu$ ops with additional MSRs.
	Can Support PEBS	Yes	
	Require Additional MSRs for tagging	IA32_PEBS_ENABLE MSR_PEBS_MATRIX_VERT Selected ESCR	See list of metrics supported by replay tagging in Table A-5.
instr_retired			This event counts instructions that are retired during a clock cycle. Mask bits specify bogus or non-bogus (and whether they are tagged using the front-end tagging mechanism).
	ESCR restrictions	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUSNTAG 1: NBOGUSTAG 2: BOGUSNTAG 3: BOGUSTAG	ESCR[24:9]  Non-bogus instructions that are not tagged. Non-bogus instructions that are tagged.  Bogus instructions that are not tagged. Bogus instructions that are tagged.
	CCCR Select	04H	CCCR[15:13]
	Event Specific Notes		1: The event count may vary depending on the microarchitectural states of the processor when the event detection is enabled. 2: The event may count more than once for some instructions with complex uop flows and were interrupted before retirement.

**Table 19-32. Performance Monitoring Events For Intel NetBurst® Microarchitecture for At-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	Can Support PEBS	No	
uops_retired			This event counts $\mu$ ops that are retired during a clock cycle. Mask bits specify bogus or non-bogus.
	ESCR restrictions	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	01H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS 1: BOGUS	ESCR[24:9]  The marked $\mu$ ops are not bogus. The marked $\mu$ ops are bogus.
	CCCR Select	04H	CCCR[15:13]
	Event Specific Notes		P6: EMON_UOPS_RETIRE
	Can Support PEBS	No	
uop_type			This event is used in conjunction with the front-end at-retirement mechanism to tag load and store $\mu$ ops.
	ESCR restrictions	MSR_RAT_ESCR0 MSR_RAT_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	Bit 1: TAGLOADS 2: TAGSTORES	ESCR[24:9]  The $\mu$ op is a load operation. The $\mu$ op is a store operation.
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		Setting the TAGLOADS and TAGSTORES mask bits does not cause a counter to increment. They are only used to tag uops.
	Can Support PEBS	No	
branch_retired			This event counts the retirement of a branch. Specify one or more mask bits to select any combination of taken, not-taken, predicted and mispredicted.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	See Table 18-70 for the addresses of the ESCR MSRs
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	The counter numbers associated with each ESCR are provided. The performance counters and corresponding CCCRs can be obtained from Table 18-70.
	ESCR Event Select	06H	ESCR[31:25]



**Table 19-32. Performance Monitoring Events For Intel NetBurst® Microarchitecture for At-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask	Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9]  Branch not-taken predicted Branch not-taken mispredicted Branch taken predicted Branch taken mispredicted
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	
	mispred_branch_retired		
	ESCR restrictions	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS	ESCR[24:9] The retired instruction is not bogus.
	CCCR Select	04H	CCCR[15:13]
	Can Support PEBS	No	
	x87_assist		
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask	Bit 0: FPSU 1: FPSO 2: POAO 3: POAU 4: PREA	ESCR[24:9]  Handle FP stack underflow. Handle FP stack overflow. Handle x87 output overflow. Handle x87 output underflow. Handle x87 input assist.
	CCCR Select	05H	CCCR[15:13]
	Can Support PEBS	No	

**Table 19-32. Performance Monitoring Events For Intel NetBurst® Microarchitecture for At-Retirement Counting (Contd.)**

Event Name	Event Parameters	Parameter Value	Description
machine_clear			This event increments according to the mask bit specified while the entire pipeline of the machine is cleared. Specify one of the mask bit to select the cause.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	Bit 0: CLEAR  2: MOCLEAR  6: SMCLEAR	ESCR[24:9]  Counts for a portion of the many cycles while the machine is cleared for any cause. Use Edge triggering for this bit only to get a count of occurrence versus a duration.  Increments each time the machine is cleared due to memory ordering issues.  Increments each time the machine is cleared due to self-modifying code issues.
	CCCR Select	05H	CCCR[15:13]
	Can Support PEBS	No	

**Table 19-33. Intel NetBurst® Microarchitecture Model-Specific Performance Monitoring Events (For Model Encoding 3, 4 or 6)**

Event Name	Event Parameters	Parameter Value	Description
instr_completed			This event counts instructions that have completed and retired during a clock cycle. Mask bits specify whether the instruction is bogus or non-bogus and whether they are:
	ESCR restrictions	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	07H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS 1: BOGUS	ESCR[24:9]  Non-bogus instructions Bogus instructions
	CCCR Select	04H	CCCR[15:13]
	Event Specific Notes		This metric differs from instr_retired, since it counts instructions completed, rather than the number of times that instructions started.
	Can Support PEBS	No	

**Table 19-34. List of Metrics Available for Front\_end Tagging (For Front\_end Event Only)**

Front-end metric <sup>1</sup>	MSR_TC_PRECISE_EVENT MSR Bit field	Additional MSR	Event mask value for Front_end_event
memory_loads	None	Set TAGLOADS bit in ESCR corresponding to event Uop_Type.	NBOGUS
memory_stores	None	Set TAGSTORES bit in the ESCR corresponding to event Uop_Type.	NBOGUS

**NOTES:**

1. There may be some undercounting of front end events when there is an overflow or underflow of the floating point stack.

**Table 19-35. List of Metrics Available for Execution Tagging (For Execution Event Only)**

Execution metric	Upstream ESCR	TagValue in Upstream ESCR	Event mask value for execution_event
packed_SP_retired	Set ALL bit in event mask, TagUop bit in ESCR of packed_SP_uop.	1	NBOGUS0
packed_DP_retired	Set ALL bit in event mask, TagUop bit in ESCR of packed_DP_uop.	1	NBOGUS0
scalar_SP_retired	Set ALL bit in event mask, TagUop bit in ESCR of scalar_SP_uop.	1	NBOGUS0
scalar_DP_retired	Set ALL bit in event mask, TagUop bit in ESCR of scalar_DP_uop.	1	NBOGUS0
128_bit_MMX_retired	Set ALL bit in event mask, TagUop bit in ESCR of 128_bit_MMX_uop.	1	NBOGUS0
64_bit_MMX_retired	Set ALL bit in event mask, TagUop bit in ESCR of 64_bit_MMX_uop.	1	NBOGUS0
X87_FP_retired	Set ALL bit in event mask, TagUop bit in ESCR of x87_FP_uop.	1	NBOGUS0
X87_SIMD_memory_moves_retired	Set ALLP0, ALLP2 bits in event mask, TagUop bit in ESCR of X87_SIMD_moves_uop.	1	NBOGUS0

**Table 19-36. List of Metrics Available for Replay Tagging (For Replay Event Only)**

Replay metric <sup>1</sup>	IA32_PEBS_ENABLE Field to Set	MSR_PEBS_MATRIX_VERT Bit Field to Set	Additional MSR/ Event	Event Mask Value for Replay_event
1stL_cache_load_miss_retired	Bit 0, Bit 24, Bit 25	Bit 0	None	NBOGUS
2ndL_cache_load_miss_retired <sup>2</sup>	Bit 1, Bit 24, Bit 25	Bit 0	None	NBOGUS
DTLB_load_miss_retired	Bit 2, Bit 24, Bit 25	Bit 0	None	NBOGUS
DTLB_store_miss_retired	Bit 2, Bit 24, Bit 25	Bit 1	None	NBOGUS
DTLB_all_miss_retired	Bit 2, Bit 24, Bit 25	Bit 0, Bit 1	None	NBOGUS
Tagged_mispred_branch	Bit 15, Bit 16, Bit 24, Bit 25	Bit 4	None	NBOGUS

**Table 19-36. List of Metrics Available for Replay Tagging (For Replay Event Only) (Contd.)**

Replay metric <sup>1</sup>	IA32_PEBS_ENABLE Field to Set	MSR_PEBS_MATRIX_VERT Bit Field to Set	Additional MSR/ Event	Event Mask Value for Replay_event
MOB_load_replay_retired <sup>3</sup>	Bit 9, Bit 24, Bit 25	Bit 0	Select MOB_load_replay event and set PARTIAL_DATA and UNALGN_ADDR bit.	NBOGUS
split_load_retired	Bit 10, Bit 24, Bit 25	Bit 0	Select load_port_replay event with the MSR_SAAT_ESCR1 MSR and set the SPLIT_LD mask bit.	NBOGUS
split_store_retired	Bit 10, Bit 24, Bit 25	Bit 1	Select store_port_replay event with the MSR_SAAT_ESCR0 MSR and set the SPLIT_ST mask bit.	NBOGUS

**NOTES:**

1. Certain kinds of  $\mu$ ops cannot be tagged. These include I/O operations, UC and locked accesses, returns, and far transfers.
2. 2nd-level misses retired does not count all 2nd-level misses. It only includes those references that are found to be misses by the fast detection logic and not those that are later found to be misses.
3. While there are several causes for a MOB replay, the event counted with this event mask setting is the case where the data from a load that would otherwise be forwarded is not an aligned subset of the data from a preceding store.

**Table 19-37. Event Mask Qualification for Logical Processors**

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
Non-Retirement	BPU_fetch_request	Bit 0: TCMISS	TS
Non-Retirement	BSQ_allocation	Bit 0: REQ_TYPE0 1: REQ_TYPE1 2: REQ_LEN0 3: REQ_LEN1 5: REQ_IO_TYPE 6: REQ_LOCK_TYPE 7: REQ_CACHE_TYPE 8: REQ_SPLIT_TYPE 9: REQ_DEM_TYPE 10: REQ_ORD_TYPE 11: MEM_TYPE0 12: MEM_TYPE1 13: MEM_TYPE2	TS TS TS TS TS TS TS TS TS TS TS TS TS
Non-Retirement	BSQ_cache_reference	Bit 0: RD_2ndL_HITS 1: RD_2ndL_HITE 2: RD_2ndL_HITM 3: RD_3rdL_HITS 4: RD_3rdL_HITE 5: RD_3rdL_HITM 6: WR_2ndL_HIT 7: WR_3rdL_HIT 8: RD_2ndL_MISS 9: RD_3rdL_MISS 10: WR_2ndL_MISS 11: WR_3rdL_MISS	TS TS TS TS TS TS TS TS TS TS TS TS
Non-Retirement	memory_cancel	Bit 2: ST_RB_FULL 3: 64K_CONF	TS TS
Non-Retirement	SSE_input_assist	Bit 15: ALL	TI
Non-Retirement	64bit_MMX_uop	Bit 15: ALL	TI
Non-Retirement	packed_DP_uop	Bit 15: ALL	TI
Non-Retirement	packed_SP_uop	Bit 15: ALL	TI
Non-Retirement	scalar_DP_uop	Bit 15: ALL	TI
Non-Retirement	scalar_SP_uop	Bit 15: ALL	TI
Non-Retirement	128bit_MMX_uop	Bit 15: ALL	TI
Non-Retirement	x87_FP_uop	Bit 15: ALL	TI

Table 19-37. Event Mask Qualification for Logical Processors (Contd.)

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
Non-Retirement	x87_SIMD_moves_uop	Bit 3: ALLP0 4: ALLP2	TI TI
Non-Retirement	FSB_data_activity	Bit 0: DRDY_DRV 1: DRDY_OWN 2: DRDY_OTHER 3: DBSY_DRV 4: DBSY_OWN 5: DBSY_OTHER	TI TI TI TI TI TI
Non-Retirement	IOQ_allocation	Bit 0: ReqA0 1: ReqA1 2: ReqA2 3: ReqA3 4: ReqA4 5: ALL_READ 6: ALL_WRITE 7: MEM_UC 8: MEM_WC 9: MEM_WT 10: MEM_WP 11: MEM_WB 13: OWN 14: OTHER 15: PREFETCH	TS TS TS TS TS TS TS TS TS TS TS TS TS TS TS
Non-Retirement	IOQ_active_entries	Bit 0: ReqA0 1: ReqA1 2: ReqA2 3: ReqA3 4: ReqA4 5: ALL_READ 6: ALL_WRITE 7: MEM_UC 8: MEM_WC 9: MEM_WT 10: MEM_WP 11: MEM_WB	TS TS TS TS TS TS TS TS TS TS TS

**Table 19-37. Event Mask Qualification for Logical Processors (Contd.)**

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
		13: OWN	TS
		14: OTHER	TS
		15: PREFETCH	TS
Non-Retirement	global_power_events	Bit 0: RUNNING	TS
Non-Retirement	ITLB_reference	Bit	
		0: HIT	TS
		1: MISS	TS
		2: HIT_UC	TS
Non-Retirement	MOB_load_replay	Bit	
		1: NO_STA	TS
		3: NO_STD	TS
		4: PARTIAL_DATA	TS
		5: UNALGN_ADDR	TS
Non-Retirement	page_walk_type	Bit	
		0: DTMISS	TI
		1: ITMISS	TI
Non-Retirement	uop_type	Bit	
		1: TAGLOADS	TS
		2: TAGSTORES	TS
Non-Retirement	load_port_replay	Bit 1: SPLIT_LD	TS
Non-Retirement	store_port_replay	Bit 1: SPLIT_ST	TS
Non-Retirement	memory_complete	Bit	
		0: LSC	TS
		1: SSC	TS
		2: USC	TS
		3: ULC	TS
Non-Retirement	retired_mispred_branch_type	Bit	
		0: UNCONDITIONAL	TS
		1: CONDITIONAL	TS
		2: CALL	TS
		3: RETURN	TS
		4: INDIRECT	TS
Non-Retirement	retired_branch_type	Bit	
		0: UNCONDITIONAL	TS
		1: CONDITIONAL	TS
		2: CALL	TS
		3: RETURN	TS
		4: INDIRECT	TS

**Table 19-37. Event Mask Qualification for Logical Processors (Contd.)**

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
Non-Retirement	tc_ms_xfer	Bit 0: CISC	TS
Non-Retirement	tc_misc	Bit 4: FLUSH	TS
Non-Retirement	TC_deliver_mode	Bit 0: DD 1: DB 2: DI 3: BD 4: BB 5: BI 6: ID 7: IB	TI TI TI TI TI TI TI TI
Non-Retirement	uop_queue_writes	Bit 0: FROM_TC_BUILD 1: FROM_TC_DELIVER 2: FROM_ROM	TS TS TS
Non-Retirement	resource_stall	Bit 5: SBFULL	TS
Non-Retirement	WC_Buffer	Bit 0: WCB_EVICTS 1: WCB_FULL_EVICT 2: WCB_HITM_EVICT	TI TI TI TI
At Retirement	instr_retired	Bit 0: NBOGUSNTAG 1: NBOGUSTAG 2: BOGUSNTAG 3: BOGUSTAG	TS TS TS TS
At Retirement	machine_clear	Bit 0: CLEAR 2: MOCLEAR 6: SMCLEAR	TS TS TS
At Retirement	front_end_event	Bit 0: NBOGUS 1: BOGUS	TS TS
At Retirement	replay_event	Bit 0: NBOGUS 1: BOGUS	TS TS
At Retirement	execution_event	Bit 0: NONBOGUS0 1: NONBOGUS1	TS TS



**Table 19-37. Event Mask Qualification for Logical Processors (Contd.)**

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
		2: NONBOGUS2 3: NONBOGUS3 4: BOGUS0 5: BOGUS1 6: BOGUS2 7: BOGUS3	TS TS TS TS TS TS
At Retirement	x87_assist	Bit 0: FPSU 1: FPSO 2: POAO 3: POAU 4: PREA	TS TS TS TS TS
At Retirement	branch_retired	Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	TS TS TS TS
At Retirement	mispred_branch_retired	Bit 0: NBOGUS	TS
At Retirement	uops_retired	Bit 0: NBOGUS 1: BOGUS	TS TS
At Retirement	instr_completed	Bit 0: NBOGUS 1: BOGUS	TS TS

## 19.19 PERFORMANCE MONITORING EVENTS FOR INTEL® PENTIUM® M PROCESSORS

- The Pentium M processor’s performance monitoring events are based on monitoring events for the P6 family of processors. All of these performance events are model specific for the Pentium M processor and are not available in this form in other processors. Table 19-38 lists the performance monitoring events that were added in the Pentium M processor.

**Table 19-38. Performance Monitoring Events on Intel® Pentium® M Processors**

Name	Hex Values	Descriptions
Power Management		
EMON_EST_TRANS	58H	Number of Enhanced Intel SpeedStep technology transitions: Mask = 00H - All transitions Mask = 02H - Only Frequency transitions
EMON_THERMAL_TRIP	59H	Duration/Occurrences in thermal trip; to count number of thermal trips: bit 22 in PerfEvtSel0/1 needs to be set to enable edge detect.
BPU		
BR_INST_EXEC	88H	Branch instructions that were executed (not necessarily retired).
BR_MISSP_EXEC	89H	Branch instructions executed that were mispredicted at execution.
BR_BAC_MISSP_EXEC	8AH	Branch instructions executed that were mispredicted at front end (BAC).
BR_CND_EXEC	8BH	Conditional branch instructions that were executed.
BR_CND_MISSP_EXEC	8CH	Conditional branch instructions executed that were mispredicted.
BR_IND_EXEC	8DH	Indirect branch instructions executed.
BR_IND_MISSP_EXEC	8EH	Indirect branch instructions executed that were mispredicted.
BR_RET_EXEC	8FH	Return branch instructions executed.
BR_RET_MISSP_EXEC	90H	Return branch instructions executed that were mispredicted at execution.
BR_RET_BAC_MISSP_EXEC	91H	Return branch instructions executed that were mispredicted at front end (BAC).
BR_CALL_EXEC	92H	CALL instruction executed.
BR_CALL_MISSP_EXEC	93H	CALL instruction executed and miss predicted.
BR_IND_CALL_EXEC	94H	Indirect CALL instructions executed.
Decoder		
EMON_SIMD_INSTR_RETIRED	CEH	Number of retired MMX instructions.
EMON_SYNCH_UOPS	D3H	Sync micro-ops
EMON_ESP_UOPS	D7H	Total number of micro-ops
EMON_FUSED_UOPS_RET	DAH	Number of retired fused micro-ops: Mask = 0 - Fused micro-ops Mask = 1 - Only load+Op micro-ops Mask = 2 - Only std+sta micro-ops
EMON_UNFUSION	DBH	Number of unfusion events in the ROB, happened on a FP exception to a fused $\mu$ op.
Prefetcher		
EMON_PREF_RQSTS_UP	FOH	Number of upward prefetches issued.
EMON_PREF_RQSTS_DN	F8H	Number of downward prefetches issued.

A number of P6 family processor performance monitoring events are modified for the Pentium M processor. Table 19-39 lists the performance monitoring events that were changed in the Pentium M processor, and differ from performance monitoring events for the P6 family of processors.

**Table 19-39. Performance Monitoring Events Modified on Intel® Pentium® M Processors**

Name	Hex Values	Descriptions
CPU_CLK_UNHALTED	79H	Number of cycles during which the processor is not halted, and not in a thermal trip.
EMON_SSE_SSE2_INST_RETIRED	D8H	Streaming SIMD Extensions Instructions Retired: Mask = 0 - SSE packed single and scalar single Mask = 1 - SSE scalar-single Mask = 2 - SSE2 packed-double Mask = 3 - SSE2 scalar-double
EMON_SSE_SSE2_COMP_INST_RETIRED	D9H	Computational SSE Instructions Retired: Mask = 0 - SSE packed single Mask = 1 - SSE Scalar-single Mask = 2 - SSE2 packed-double Mask = 3 - SSE2 scalar-double
L2_LD	29H	L2 data loads
L2_LINES_IN	24H	L2 lines allocated
L2_LINES_OUT	26H	L2 lines evicted
L2_M_LINES_OUT	27H	Lw M-state lines evicted
		Mask[0] = 1 - count I state lines Mask[1] = 1 - count S state lines Mask[2] = 1 - count E state lines Mask[3] = 1 - count M state lines Mask[5:4]: 00H - Excluding hardware-prefetched lines 01H - Hardware-prefetched lines only 02H/03H - All (HW-prefetched lines and non HW -- Prefetched lines)

## 19.20 P6 FAMILY PROCESSOR PERFORMANCE MONITORING EVENTS

Table 19-40 lists the events that can be counted with the performance monitoring counters and read with the RDPMC instruction for the P6 family processors. The unit column gives the microarchitecture or bus unit that produces the event; the event number column gives the hexadecimal number identifying the event; the mnemonic event name column gives the name of the event; the unit mask column gives the unit mask required (if any); the description column describes the event; and the comments column gives additional information about the event.

All of these performance events are model specific for the P6 family processors and are not available in this form in the Pentium 4 processors or the Pentium processors. Some events (such as those added in later generations of the P6 family processors) are only available in specific processors in the P6 family. All performance event encodings not listed in Table 19-40 are reserved and their use will result in undefined counter results.

See the end of the table for notes related to certain entries in the table.

Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
Data Cache Unit (DCU)	43H	DATA_MEM_REFS	00H	All loads from any memory type. All stores to any memory type. Each part of a split is counted separately. The internal logic counts not only memory loads and stores, but also internal retries.  80-bit floating-point accesses are double counted, since they are decomposed into a 16-bit exponent load and a 64-bit mantissa load. Memory accesses are only counted when they are actually performed (such as a load that gets squashed because a previous cache miss is outstanding to the same address, and which finally gets performed, is only counted once).  Does not include I/O accesses, or other nonmemory accesses.	
	45H	DCU_LINES_IN	00H	Total lines allocated in DCU.	
	46H	DCU_M_LINES_IN	00H	Number of M state lines allocated in DCU.	
	47H	DCU_M_LINES_OUT	00H	Number of M state lines evicted from DCU. This includes evictions via snoop HITM, intervention or replacement.	
	48H	DCU_MISS_OUTSTANDING	00H	Weighted number of cycles while a DCU miss is outstanding, incremented by the number of outstanding cache misses at any particular time.  Cacheable read requests only are considered.  Uncacheable requests are excluded.  Read-for-ownerships are counted, as well as line fills, invalidates, and stores.	An access that also misses the L2 is short-changed by 2 cycles (i.e., if counts N cycles, should be N+2 cycles).  Subsequent loads to the same cache line will not result in any additional counts.  Count value not precise, but still useful.
Instruction Fetch Unit (IFU)	80H	IFU_IFETCH	00H	Number of instruction fetches, both cacheable and noncacheable, including UC fetches.	
	81H	IFU_IFETCH_MISS	00H	Number of instruction fetch misses All instruction fetches that do not hit the IFU (i.e., that produce memory requests). This includes UC accesses.	
	85H	ITLB_MISS	00H	Number of ITLB misses.	
	86H	IFU_MEM_STALL	00H	Number of cycles instruction fetch is stalled, for any reason.  Includes IFU cache misses, ITLB misses, ITLB faults, and other minor stalls.	
	87H	ILD_STALL	00H	Number of cycles that the instruction length decoder is stalled.	
L2 Cache <sup>1</sup>	28H	L2_IFETCH	MESI OFH	Number of L2 instruction fetches.  This event indicates that a normal instruction fetch was received by the L2.	

**Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters (Contd.)**

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
				The count includes only L2 cacheable instruction fetches; it does not include UC instruction fetches. It does not include ITLB miss accesses.	
	29H	L2_LD	MESI 0FH	Number of L2 data loads. This event indicates that a normal, unlocked, load memory access was received by the L2. It includes only L2 cacheable memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses such as UC/WT memory accesses. It does include L2 cacheable TLB miss memory accesses.	
	2AH	L2_ST	MESI 0FH	Number of L2 data stores. This event indicates that a normal, unlocked, store memory access was received by the L2. It indicates that the DCU sent a read-for-ownership request to the L2. It also includes Invalid to Modified requests sent by the DCU to the L2. It includes only L2 cacheable memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses such as UC/WT memory accesses. It includes TLB miss memory accesses.	
	24H	L2_LINES_IN	00H	Number of lines allocated in the L2.	
	26H	L2_LINES_OUT	00H	Number of lines removed from the L2 for any reason.	
	25H	L2_M_LINES_INM	00H	Number of modified lines allocated in the L2.	
	27H	L2_M_LINES_OUTM	00H	Number of modified lines removed from the L2 for any reason.	
	2EH	L2_RQSTS	MESI 0FH	Total number of L2 requests.	
	21H	L2_ADS	00H	Number of L2 address strobes.	
	22H	L2_DBUS_BUSY	00H	Number of cycles during which the L2 cache data bus was busy.	
	23H	L2_DBUS_BUSY_RD	00H	Number of cycles during which the data bus was busy transferring read data from L2 to the processor.	
External Bus Logic (EBL) <sup>2</sup>	62H	BUS_DRDY_CLOCKS	00H (Self) 20H (Any)	Number of clocks during which DRDY# is asserted. Utilization of the external system data bus during data transfers.	Unit Mask = 00H counts bus clocks when the processor is driving DRDY#. Unit Mask = 20H counts in processor clocks when any agent is driving DRDY#.

**Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters (Contd.)**

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	63H	BUS_LOCK_CLOCKS	00H (Self) 20H (Any)	Number of clocks during which LOCK# is asserted on the external system bus. <sup>3</sup>	Always counts in processor clocks.
	60H	BUS_REQ_OUTSTANDING	00H (Self)	Number of bus requests outstanding. This counter is incremented by the number of cacheable read bus requests outstanding in any given cycle.	Counts only DCU full-line cacheable reads, not RFOs, writes, instruction fetches, or anything else. Counts "waiting for bus to complete" (last data chunk received).
	65H	BUS_TRAN_BRD	00H (Self) 20H (Any)	Number of burst read transactions.	
	66H	BUS_TRAN_RFO	00H (Self) 20H (Any)	Number of completed read for ownership transactions.	
	67H	BUS_TRANS_WB	00H (Self) 20H (Any)	Number of completed write back transactions.	
	68H	BUS_TRAN_IFETCH	00H (Self) 20H (Any)	Number of completed instruction fetch transactions.	
	69H	BUS_TRAN_INVALID	00H (Self) 20H (Any)	Number of completed invalidate transactions.	
	6AH	BUS_TRAN_PWR	00H (Self) 20H (Any)	Number of completed partial write transactions.	
	6BH	BUS_TRANS_P	00H (Self) 20H (Any)	Number of completed partial transactions.	
	6CH	BUS_TRANS_IO	00H (Self) 20H (Any)	Number of completed I/O transactions.	
	6DH	BUS_TRAN_DEF	00H (Self) 20H (Any)	Number of completed deferred transactions.	

**Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters (Contd.)**

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	6EH	BUS_TRAN_BURST	00H (Self) 20H (Any)	Number of completed burst transactions.	
	70H	BUS_TRAN_ANY	00H (Self) 20H (Any)	Number of all completed bus transactions. Address bus utilization can be calculated knowing the minimum address bus occupancy. Includes special cycles, etc.	
	6FH	BUS_TRAN_MEM	00H (Self) 20H (Any)	Number of completed memory transactions.	
	64H	BUS_DATA_RCV	00H (Self)	Number of bus clock cycles during which this processor is receiving data.	
	61H	BUS_BNR_DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the BNR# pin.	
	7AH	BUS_HIT_DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HIT# pin.	Includes cycles due to snoop stalls. The event counts correctly, but BPM <sub>i</sub> (breakpoint monitor) pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers): <ul style="list-style-type: none"> <li>▪ If the core-clock-to- bus-clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM<sub>i</sub> pins will be asserted for a single clock when the counters overflow.</li> <li>▪ If the PC bit is clear, the processor toggles the BPM<sub>i</sub> pins when the counter overflows.</li> <li>▪ If the clock ratio is not 2:1 or 3:1, the BPM<sub>i</sub> pins will not function for these performance monitoring counter events.</li> </ul>
	7BH	BUS_HITM_DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HITM# pin.	Includes cycles due to snoop stalls. The event counts correctly, but BPM <sub>i</sub> (breakpoint monitor) pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers): <ul style="list-style-type: none"> <li>▪ If the core-clock-to- bus-clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM<sub>i</sub> pins will be asserted for a single clock when the counters overflow.</li> </ul>

Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
					<ul style="list-style-type: none"> <li>If the PC bit is clear, the processor toggles the BPM<sub>i</sub> pins when the counter overflows.</li> <li>If the clock ratio is not 2:1 or 3:1, the BPM<sub>i</sub> pins will not function for these performance monitoring counter events.</li> </ul>
	7EH	BUS_SNOOP_STALL	00H (Self)	Number of clock cycles during which the bus is snoop stalled.	
Floating-Point Unit	C1H	FLOPS	00H	<p>Number of computational floating-point operations retired.</p> <p>Excludes floating-point computational operations that cause traps or assists.</p> <p>Includes floating-point computational operations executed by the assist handler.</p> <p>Includes internal sub-operations for complex floating-point instructions like transcendentals.</p> <p>Excludes floating-point loads and stores.</p>	Counter 0 only.
	10H	FP_COMP_OPS_EXE	00H	<p>Number of computational floating-point operations executed.</p> <p>The number of FADD, FSUB, FCOM, FMULs, integer MULs and IMULs, FDIVs, FPREM, FSQRTS, integer DIVs, and IDIVs.</p> <p>This number does not include the number of cycles, but the number of operations.</p> <p>This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.</p>	Counter 0 only.
	11H	FP_ASSIST	00H	Number of floating-point exception cases handled by microcode.	Counter 1 only. This event includes counts due to speculative execution.
	12H	MUL	00H	<p>Number of multiplies.</p> <p>This count includes integer as well as FP multiplies and is speculative.</p>	Counter 1 only.
	13H	DIV	00H	<p>Number of divides.</p> <p>This count includes integer as well as FP divides and is speculative.</p>	Counter 1 only.
	14H	CYCLES_DIV_BUSY	00H	<p>Number of cycles during which the divider is busy, and cannot accept new divides.</p> <p>This includes integer and FP divides, FPREM, FPSQRT, etc. and is speculative.</p>	Counter 0 only.



**Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters (Contd.)**

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
Memory Ordering	03H	LD_BLOCKS	00H	Number of load operations delayed due to store buffer blocks.  Includes counts caused by preceding stores whose addresses are unknown, preceding stores whose addresses are known but whose data is unknown, and preceding stores that conflicts with the load but which incompletely overlap the load.	
	04H	SB_DRAINS	00H	Number of store buffer drain cycles.  Incremented every cycle the store buffer is draining.  Draining is caused by serializing operations like CPUID, synchronizing operations like XCHG, interrupt acknowledgment, as well as other conditions (such as cache flushing).	
	05H	MISALIGN_MEM_REF	00H	Number of misaligned data memory references.  Incremented by 1 every cycle, during which either the processor's load or store pipeline dispatches a misaligned $\mu$ op.  Counting is performed if it is the first or second half, or if it is blocked, squashed, or missed.  In this context, misaligned means crossing a 64-bit boundary.	MISALIGN_MEM_REF is only an approximation to the true number of misaligned memory references.  The value returned is roughly proportional to the number of misaligned memory accesses (the size of the problem).
	07H	EMON_KNI_PREF_DISPATCHED	00H 01H 02H 03H	Number of Streaming SIMD extensions prefetch/weakly-ordered instructions dispatched (speculative prefetches are included in counting):  0: prefetch NTA 1: prefetch T1 2: prefetch T2 3: weakly ordered stores	Counters 0 and 1. Pentium III processor only.
	4BH	EMON_KNI_PREF_MISS	00H 01H 02H 03H	Number of prefetch/weakly-ordered instructions that miss all caches:  0: prefetch NTA 1: prefetch T1 2: prefetch T2 3: weakly ordered stores	Counters 0 and 1. Pentium III processor only.
Instruction Decoding and Retirement	COH	INST_RETIRED	00H	Number of instructions retired.	A hardware interrupt received during/after the last iteration of the REP STOS flow causes the counter to undercount by 1 instruction.
					An SMI received while executing a HLT instruction will cause the performance counter to not count the RSM instruction and undercount by 1.

**Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters (Contd.)**

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	C2H	UOPS_RETIRED	00H	Number of $\mu$ ops retired.	
	D0H	INST_DECODED	00H	Number of instructions decoded.	
	D8H	EMON_KNI_INST_RETIRED	00H 01H	Number of Streaming SIMD extensions retired: 0: packed & scalar 1: scalar	Counters 0 and 1. Pentium III processor only.
	D9H	EMON_KNI_COMP_INST_RET	00H 01H	Number of Streaming SIMD extensions computation instructions retired: 0: packed and scalar 1: scalar	Counters 0 and 1. Pentium III processor only.
Interrupts	C8H	HW_INT_RX	00H	Number of hardware interrupts received.	
	C6H	CYCLES_INT_MASKED	00H	Number of processor cycles for which interrupts are disabled.	
	C7H	CYCLES_INT_PENDING_AND_MASKED	00H	Number of processor cycles for which interrupts are disabled and interrupts are pending.	
Branches	C4H	BR_INST_RETIRED	00H	Number of branch instructions retired.	
	C5H	BR_MISS_PRED_RETIRED	00H	Number of mispredicted branches retired.	
	C9H	BR_TAKEN_RETIRED	00H	Number of taken branches retired.	
	CAH	BR_MISS_PRED_TAKEN_RET	00H	Number of taken mispredictions branches retired.	
	E0H	BR_INST_DECODED	00H	Number of branch instructions decoded.	
	E2H	BTB_MISSES	00H	Number of branches for which the BTB did not produce a prediction.	
	E4H	BR_BOGUS	00H	Number of bogus branches.	
	E6H	BACLEAR	00H	Number of times BACLEAR is asserted. This is the number of times that a static branch prediction was made, in which the branch decoder decided to make a branch prediction because the BTB did not.	
Stalls	A2H	RESOURCE_STALLS	00H	Incremented by 1 during every cycle for which there is a resource related stall. Includes register renaming buffer entries, memory buffer entries.	

**Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters (Contd.)**

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
				Does not include stalls due to bus queue full, too many cache misses, etc. In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, such as if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	
	D2H	PARTIAL_RAT_STALLS	00H	Number of cycles or events for partial stalls. This includes flag partial stalls.	
Segment Register Loads	06H	SEGMENT_REG_LOADS	00H	Number of segment register loads.	
Clocks	79H	CPU_CLK_UNHALTED	00H	Number of cycles during which the processor is not halted.	
MMX Unit	B0H	MMX_INSTR_EXEC	00H	Number of MMX Instructions Executed.	Available in Intel Celeron, Pentium II and Pentium II Xeon processors only. Does not account for MOVQ and MOVD stores from register to memory.
	B1H	MMX_SAT_INSTR_EXEC	00H	Number of MMX Saturating Instructions Executed.	Available in Pentium II and Pentium III processors only.
	B2H	MMX_UOPS_EXEC	0FH	Number of MMX $\mu$ ops Executed.	Available in Pentium II and Pentium III processors only.
	B3H	MMX_INSTR_TYPE_EXEC	01H	MMX packed multiply instructions executed.	Available in Pentium II and Pentium III processors only.
			02H	MMX packed shift instructions executed.	
			04H	MMX pack operation instructions executed.	
			08H	MMX unpack operation instructions executed.	
			10H	MMX packed logical instructions executed.	
20H	MMX packed arithmetic instructions executed.				
CCH	FP_MMX_TRANS	00H	Transitions from MMX instruction to floating-point instructions.	Available in Pentium II and Pentium III processors only.	
		01H	Transitions from floating-point instructions to MMX instructions.		
CDH	MMX_ASSIST	00H	Number of MMX Assists (that is, the number of EMMS instructions executed).	Available in Pentium II and Pentium III processors only.	
CEH	MMX_INSTR_RET	00H	Number of MMX Instructions Retired.	Available in Pentium II processors only.	
Segment Register Renaming	D4H	SEG_RENAME_STALLS		Number of Segment Register Renaming Stalls:	Available in Pentium II and Pentium III processors only.

**Table 19-40. Events That Can Be Counted with the P6 Family Performance Monitoring Counters (Contd.)**

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
			02H 04H 08H 0FH	Segment register ES Segment register DS Segment register FS Segment register FS Segment registers ES + DS + FS + GS	
	D5H	SEG_REG_RENAMES	01H 02H 04H 08H 0FH	Number of Segment Register Renames: Segment register ES Segment register DS Segment register FS Segment register FS Segment registers ES + DS + FS + GS	Available in Pentium II and Pentium III processors only.
	D6H	RET_SEG_RENAMES	00H	Number of segment register rename events retired.	Available in Pentium II and Pentium III processors only.

**NOTES:**

- Several L2 cache events, where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. The lower 4 bits of the Unit Mask field are used in conjunction with L2 events to indicate the cache state or cache states involved.  
The P6 family processors identify cache states using the "MESI" protocol and consequently each bit in the Unit Mask field represents one of the four states: UMSK[3] = M (8H) state, UMSK[2] = E (4H) state, UMSK[1] = S (2H) state, and UMSK[0] = I (1H) state. UMSK[3:0] = MESI" (FH) should be used to collect data for all states; UMSK = 0H, for the applicable events, will result in nothing being counted.
- All of the external bus logic (EBL) events, except where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers.  
Bit 5 of the UMSK field is used in conjunction with the EBL events to indicate whether the processor should count transactions that are self-generated (UMSK[5] = 0) or transactions that result from any processor on the bus (UMSK[5] = 1).
- L2 cache locks, so it is possible to have a zero count.

## 19.21 PENTIUM PROCESSOR PERFORMANCE MONITORING EVENTS

Table 19-41 lists the events that can be counted with the performance monitoring counters for the Pentium processor. The Event Number column gives the hexadecimal code that identifies the event and that is entered in the ES0 or ES1 (event select) fields of the CESR MSR. The Mnemonic Event Name column gives the name of the event, and the Description and Comments columns give detailed descriptions of the events. Most events can be counted with either counter 0 or counter 1; however, some events can only be counted with only counter 0 or only counter 1 (as noted).

**NOTE**

The events in the table that are shaded are implemented only in the Pentium processor with MMX technology.

**Table 19-41. Events That Can Be Counted with Pentium Processor Performance Monitoring Counters**

Event Num.	Mnemonic Event Name	Description	Comments
00H	DATA_READ	Number of memory data reads (internal data cache hit and miss combined).	Split cycle reads are counted individually. Data Memory Reads that are part of TLB miss processing are not included. These events may occur at a maximum of two per clock. I/O is not included.
01H	DATA_WRITE	Number of memory data writes (internal data cache hit and miss combined); I/O not included.	Split cycle writes are counted individually. These events may occur at a maximum of two per clock. I/O is not included.
0H2	DATA_TLB_MISS	Number of misses to the data cache translation look-aside buffer.	
03H	DATA_READ_MISS	Number of memory read accesses that miss the internal data cache whether or not the access is cacheable or noncacheable.	Additional reads to the same cache line after the first BRDY# of the burst line fill is returned but before the final (fourth) BRDY# has been returned, will not cause the counter to be incremented additional times. Data accesses that are part of TLB miss processing are not included. Accesses directed to I/O space are not included.
04H	DATA WRITE MISS	Number of memory write accesses that miss the internal data cache whether or not the access is cacheable or noncacheable.	Data accesses that are part of TLB miss processing are not included. Accesses directed to I/O space are not included.
05H	WRITE_HIT_TO_M_OR_E_STATE_LINES	Number of write hits to exclusive or modified lines in the data cache.	These are the writes that may be held up if EWBE# is inactive. These events may occur a maximum of two per clock.
06H	DATA_CACHE_LINES_WRITTEN_BACK	Number of dirty lines (all) that are written back, regardless of the cause.	Replacements and internal and external snoops can all cause writeback and are counted.
07H	EXTERNAL_SNOOPS	Number of accepted external snoops whether they hit in the code cache or data cache or neither.	Assertions of EADS# outside of the sampling interval are not counted, and no internal snoops are counted.
08H	EXTERNAL_DATA_CACHE_SNOOP_HITS	Number of external snoops to the data cache.	Snoop hits to a valid line in either the data cache, the data line fill buffer, or one of the write back buffers are all counted as hits.
09H	MEMORY ACCESSES IN BOTH PIPES	Number of data memory reads or writes that are paired in both pipes of the pipeline.	These accesses are not necessarily run in parallel due to cache misses, bank conflicts, etc.
0AH	BANK CONFLICTS	Number of actual bank conflicts.	
0BH	MISALIGNED DATA MEMORY OR I/O REFERENCES	Number of memory or I/O reads or writes that are misaligned.	A 2- or 4-byte access is misaligned when it crosses a 4-byte boundary; an 8-byte access is misaligned when it crosses an 8-byte boundary. Ten byte accesses are treated as two separate accesses of 8 and 2 bytes each.
0CH	CODE READ	Number of instruction reads; whether the read is cacheable or noncacheable.	Individual 8-byte noncacheable instruction reads are counted.
0DH	CODE TLB MISS	Number of instruction reads that miss the code TLB whether the read is cacheable or noncacheable.	Individual 8-byte noncacheable instruction reads are counted.
0EH	CODE CACHE MISS	Number of instruction reads that miss the internal code cache; whether the read is cacheable or noncacheable.	Individual 8-byte noncacheable instruction reads are counted.

**Table 19-41. Events That Can Be Counted with Pentium Processor Performance Monitoring Counters (Contd.)**

Event Num.	Mnemonic Event Name	Description	Comments
0FH	ANY SEGMENT REGISTER LOADED	Number of writes into any segment register in real or protected mode including the LDTR, GDTR, IDTR, and TR.	Segment loads are caused by explicit segment register load instructions, far control transfers, and task switches. Far control transfers and task switches causing a privilege level change will signal this event twice. Interrupts and exceptions may initiate a far control transfer.
10H	Reserved		
11H	Reserved		
12H	Branches	Number of taken and not taken branches, including: conditional branches, jumps, calls, returns, software interrupts, and interrupt returns.	Also counted as taken branches are serializing instructions, VERR and VERW instructions, some segment descriptor loads, hardware interrupts (including FLUSH#), and programmatic exceptions that invoke a trap or fault handler. The pipe is not necessarily flushed. The number of branches actually executed is measured, not the number of predicted branches.
13H	BTB_HITS	Number of BTB hits that occur.	Hits are counted only for those instructions that are actually executed.
14H	TAKEN_BRANCH_OR_BTBT_HIT	Number of taken branches or BTB hits that occur.	This event type is a logical OR of taken branches and BTB hits. It represents an event that may cause a hit in the BTB. Specifically, it is either a candidate for a space in the BTB or it is already in the BTB.
15H	PIPELINE FLUSHES	Number of pipeline flushes that occur Pipeline flushes are caused by BTB misses on taken branches, mispredictions, exceptions, interrupts, and some segment descriptor loads.	The counter will not be incremented for serializing instructions (serializing instructions cause the prefetch queue to be flushed but will not trigger the Pipeline Flushed event counter) and software interrupts (software interrupts do not flush the pipeline).
16H	INSTRUCTIONS_EXECUTED	Number of instructions executed (up to two per clock).	Invocations of a fault handler are considered instructions. All hardware and software interrupts and exceptions will also cause the count to be incremented. Repeat prefixed string instructions will only increment this counter once despite the fact that the repeat loop executes the same instruction multiple times until the loop criteria is satisfied. This applies to all the Repeat string instruction prefixes (i.e., REP, REPE, REPZ, REPNE, and REPNZ). This counter will also only increment once per each HLT instruction executed regardless of how many cycles the processor remains in the HALT state.
17H	INSTRUCTIONS_EXECUTED_V PIPE	Number of instructions executed in the V_pipe. The event indicates the number of instructions that were paired.	This event is the same as the 16H event except it only counts the number of instructions actually executed in the V-pipe.
18H	BUS_CYCLE_DURATION	Number of clocks while a bus cycle is in progress. This event measures bus use.	The count includes HLDA, AHOLD, and BOFF# clocks.
19H	WRITE_BUFFER_FULL_STALL_DURATION	Number of clocks while the pipeline is stalled due to full write buffers.	Full write buffers stall data memory read misses, data memory write misses, and data memory write hits to S-state lines. Stalls on I/O accesses are not included.

**Table 19-41. Events That Can Be Counted with Pentium Processor Performance Monitoring Counters (Contd.)**

Event Num.	Mnemonic Event Name	Description	Comments
1AH	WAITING_FOR_DATA_MEMORY_READ_STALL_DURATION	Number of clocks while the pipeline is stalled while waiting for data memory reads.	Data TLB Miss processing is also included in the count. The pipeline stalls while a data memory read is in progress including attempts to read that are not bypassed while a line is being filled.
1BH	STALL ON WRITE TO AN E- OR M-STATE LINE	Number of stalls on writes to E- or M-state lines.	
1CH	LOCKED BUS CYCLE	Number of locked bus cycles that occur as the result of the LOCK prefix or LOCK instruction, page-table updates, and descriptor table updates.	Only the read portion of the locked read-modify-write is counted. Split locked cycles (SCYC active) count as two separate accesses. Cycles restarted due to BOFF# are not re-counted.
1DH	I/O READ OR WRITE CYCLE	Number of bus cycles directed to I/O space.	Misaligned I/O accesses will generate two bus cycles. Bus cycles restarted due to BOFF# are not re-counted.
1EH	NONCACHEABLE_MEMORY_READS	Number of noncacheable instruction or data memory read bus cycles. The count includes read cycles caused by TLB misses, but does not include read cycles to I/O space.	Cycles restarted due to BOFF# are not re-counted.
1FH	PIPELINE_AGI_STALLS	Number of address generation interlock (AGI) stalls. An AGI occurring in both the U- and V-pipelines in the same clock signals this event twice.	An AGI occurs when the instruction in the execute stage of either of U- or V-pipelines is writing to either the index or base address register of an instruction in the D2 (address generation) stage of either the U- or V- pipelines.
20H	Reserved		
21H	Reserved		
22H	FLOPS	Number of floating-point operations that occur.	Number of floating-point adds, subtracts, multiplies, divides, remainders, and square roots are counted. The transcendental instructions consist of multiple adds and multiplies and will signal this event multiple times. Instructions generating the divide-by-zero, negative square root, special operand, or stack exceptions will not be counted.  Instructions generating all other floating-point exceptions will be counted. The integer multiply instructions and other instructions which use the x87 FPU will be counted.
23H	BREAKPOINT MATCH ON DRO REGISTER	Number of matches on register DRO breakpoint.	The counters is incremented regardless if the breakpoints are enabled or not. However, if breakpoints are not enabled, code breakpoint matches will not be checked for instructions executed in the V-pipe and will not cause this counter to be incremented. (They are checked on instruction executed in the U-pipe only when breakpoints are not enabled.)  These events correspond to the signals driven on the BP[3:0] pins. Refer to Chapter 17, "Debug, Branch Profile, TSC, and Resource Monitoring Features" for more information.
24H	BREAKPOINT MATCH ON DR1 REGISTER	Number of matches on register DR1 breakpoint.	See comment for 23H event.

**Table 19-41. Events That Can Be Counted with Pentium Processor Performance Monitoring Counters (Contd.)**

Event Num.	Mnemonic Event Name	Description	Comments
25H	BREAKPOINT MATCH ON DR2 REGISTER	Number of matches on register DR2 breakpoint.	See comment for 23H event.
26H	BREAKPOINT MATCH ON DR3 REGISTER	Number of matches on register DR3 breakpoint.	See comment for 23H event.
27H	HARDWARE INTERRUPTS	Number of taken INTR and NMI interrupts.	
28H	DATA_READ_OR_WRITE	Number of memory data reads and/or writes (internal data cache hit and miss combined).	Split cycle reads and writes are counted individually. Data Memory Reads that are part of TLB miss processing are not included. These events may occur at a maximum of two per clock. I/O is not included.
29H	DATA_READ_MISS OR_WRITE MISS	Number of memory read and/or write accesses that miss the internal data cache, whether or not the access is cacheable or noncacheable.	Additional reads to the same cache line after the first BRDY# of the burst line fill is returned but before the final (fourth) BRDY# has been returned, will not cause the counter to be incremented additional times. Data accesses that are part of TLB miss processing are not included. Accesses directed to I/O space are not included.
2AH	BUS_OWNERSHIP_LATENCY (Counter 0)	The time from LRM bus ownership request to bus ownership granted (that is, the time from the earlier of a PBREQ (0), PHITM# or HITM# assertion to a PBGNT assertion)	The ratio of the 2AH events counted on counter 0 and counter 1 is the average stall time due to bus ownership conflict.
2AH	BUS OWNERSHIP TRANSFERS (Counter 1)	The number of buss ownership transfers (that is, the number of PBREQ (0) assertions	The ratio of the 2AH events counted on counter 0 and counter 1 is the average stall time due to bus ownership conflict.
2BH	MMX_INSTRUCTIONS_EXECUTED_U-PIPE (Counter 0)	Number of MMX instructions executed in the U-pipe	
2BH	MMX_INSTRUCTIONS_EXECUTED_V-PIPE (Counter 1)	Number of MMX instructions executed in the V-pipe	
2CH	CACHE_M-STATE_LINE_SHARING (Counter 0)	Number of times a processor identified a hit to a modified line due to a memory access in the other processor (PHITM (0))	If the average memory latencies of the system are known, this event enables the user to count the Write Backs on PHITM(0) penalty and the Latency on Hit Modified(l) penalty.
2CH	CACHE_LINE_SHARING (Counter 1)	Number of shared data lines in the L1 cache (PHIT (0))	
2DH	EMMS_INSTRUCTIONS_EXECUTED (Counter 0)	Number of EMMS instructions executed	



**Table 19-41. Events That Can Be Counted with Pentium Processor Performance Monitoring Counters (Contd.)**

Event Num.	Mnemonic Event Name	Description	Comments
2DH	TRANSITIONS_BETWEEN_MMX_AND_FP_INSTRUCTIONS (Counter 1)	Number of transitions between MMX and floating-point instructions or vice versa An even count indicates the processor is in MMX state. an odd count indicates it is in FP state.	This event counts the first floating-point instruction following an MMX instruction or first MMX instruction following a floating-point instruction. The count may be used to estimate the penalty in transitions between floating-point state and MMX state.
2EH	BUS_UTILIZATION_DUE_TO_PROCESSOR_ACTIVITY (Counter 0)	Number of clocks the bus is busy due to the processor's own activity (the bus activity that is caused by the processor)	
2EH	WRITES_TO_NONCACHEABLE_MEMORY (Counter 1)	Number of write accesses to noncacheable memory	The count includes write cycles caused by TLB misses and I/O write cycles. Cycles restarted due to BOFF# are not re-counted.
2FH	SATURATING_MMX_INSTRUCTIONS_EXECUTED (Counter 0)	Number of saturating MMX instructions executed, independently of whether they actually saturated.	
2FH	SATURATIONS_PERFORMED (Counter 1)	Number of MMX instructions that used saturating arithmetic when at least one of its results actually saturated	If an MMX instruction operating on 4 doublewords saturated in three out of the four results, the counter will be incremented by one only.
30H	NUMBER_OF_CYCLES_NOT_IN_HALT_STATE (Counter 0)	Number of cycles the processor is not idle due to HLT instruction	This event will enable the user to calculate "net CPI". Note that during the time that the processor is executing the HLT instruction, the Time-Stamp Counter is not disabled. Since this event is controlled by the Counter Controls CCO, CC1 it can be used to calculate the CPI at CPL=3, which the TSC cannot provide.
30H	DATA_CACHE_TLB_MISS_STALL_DURATION (Counter 1)	Number of clocks the pipeline is stalled due to a data cache translation look-aside buffer (TLB) miss	
31H	MMX_INSTRUCTION_DATA_READS (Counter 0)	Number of MMX instruction data reads	
31H	MMX_INSTRUCTION_DATA_READ_MISSES (Counter 1)	Number of MMX instruction data read misses	
32H	FLOATING_POINT_STALLS_DURATION (Counter 0)	Number of clocks while pipe is stalled due to a floating-point freeze	
32H	TAKEN_BRANCHES (Counter 1)	Number of taken branches	

**Table 19-41. Events That Can Be Counted with Pentium Processor Performance Monitoring Counters (Contd.)**

Event Num.	Mnemonic Event Name	Description	Comments
33H	D1_STARVATION_AND_FIFO_IS_EMPTY (Counter 0)	Number of times D1 stage cannot issue ANY instructions since the FIFO buffer is empty	The D1 stage can issue 0, 1, or 2 instructions per clock if those are available in an instructions FIFO buffer.
33H	D1_STARVATION_AND_ONLY_ONE_INSTRUCTION_IN_FIFO (Counter 1)	Number of times the D1 stage issues a single instruction (since the FIFO buffer had just one instruction ready)	The D1 stage can issue 0, 1, or 2 instructions per clock if those are available in an instructions FIFO buffer. When combined with the previously defined events, Instruction Executed (16H) and Instruction Executed in the V-pipe (17H), this event enables the user to calculate the numbers of time pairing rules prevented issuing of two instructions.
34H	MMX_INSTRUCTION_DATA_WRITES (Counter 0)	Number of data writes caused by MMX instructions	
34H	MMX_INSTRUCTION_DATA_WRITE_MISSES (Counter 1)	Number of data write misses caused by MMX instructions	
35H	PIPELINE_FLUSHES_DUE_TO_WRONG_BRANCH_PREDICTIONS (Counter 0)	Number of pipeline flushes due to wrong branch predictions resolved in either the E-stage or the WB-stage	The count includes any pipeline flush due to a branch that the pipeline did not follow correctly. It includes cases where a branch was not in the BTB, cases where a branch was in the BTB but was mispredicted, and cases where a branch was correctly predicted but to the wrong address. Branches are resolved in either the Execute stage (E-stage) or the Writeback stage (WB-stage). In the later case, the misprediction penalty is larger by one clock. The difference between the 35H event count in counter 0 and counter 1 is the number of E-stage resolved branches.
35H	PIPELINE_FLUSHES_DUE_TO_WRONG_BRANCH_PREDICTIONS_RESOLVED_IN_WB-STAGE (Counter 1)	Number of pipeline flushes due to wrong branch predictions resolved in the WB-stage	See note for event 35H (Counter 0).
36H	MISALIGNED_DATA_MEMORY_REFERENCE_ON_MMX_INSTRUCTIONS (Counter 0)	Number of misaligned data memory references when executing MMX instructions	
36H	PIPELINE_STALL_FOR_MMX_INSTRUCTION_DATA_MEMORY_READS (Counter 1)	Number clocks during pipeline stalls caused by waits form MMX instruction data memory reads	T3:

**Table 19-41. Events That Can Be Counted with Pentium Processor Performance Monitoring Counters (Contd.)**

Event Num.	Mnemonic Event Name	Description	Comments
37H	MISPREDICTED_OR_UNPREDICTED_RETURNS (Counter 1)	Number of returns predicted incorrectly or not predicted at all	The count is the difference between the total number of executed returns and the number of returns that were correctly predicted. Only RET instructions are counted (for example, IRET instructions are not counted).
37H	PREDICTED_RETURNS (Counter 1)	Number of predicted returns (whether they are predicted correctly and incorrectly)	Only RET instructions are counted (for example, IRET instructions are not counted).
38H	MMX_MULTIPLY_UNIT_INTERLOCK (Counter 0)	Number of clocks the pipe is stalled since the destination of previous MMX multiply instruction is not ready yet	The counter will not be incremented if there is another cause for a stall. For each occurrence of a multiply interlock, this event will be counted twice (if the stalled instruction comes on the next clock after the multiply) or by once (if the stalled instruction comes two clocks after the multiply).
38H	MOVD/MOVQ_STORE_STALL_DUE_TO_PREVIOUS_MMX_OPERATION (Counter 1)	Number of clocks a MOVD/MOVQ instruction store is stalled in D2 stage due to a previous MMX operation with a destination to be used in the store instruction.	
39H	RETURNS (Counter 0)	Number of returns executed.	Only RET instructions are counted; IRET instructions are not counted. Any exception taken on a RET instruction and any interrupt recognized by the processor on the instruction boundary prior to the execution of the RET instruction will also cause this counter to be incremented.
39H	Reserved		
3AH	BTB_FALSE_ENTRIES (Counter 0)	Number of false entries in the Branch Target Buffer	False entries are causes for misprediction other than a wrong prediction.
3AH	BTB_MISS_PREDICTION_ON_NOT-TAKEN_BRANCH (Counter 1)	Number of times the BTB predicted a not-taken branch as taken	
3BH	FULL_WRITE_BUFFER_STALL_DURATION_WHILE_EXECUTING_MMX_INSTRUCTIONS (Counter 0)	Number of clocks while the pipeline is stalled due to full write buffers while executing MMX instructions	
3BH	STALL_ON_MMX_INSTRUCTION_WRITE_TO_E-OR_M-STATE_LINE (Counter 1)	Number of clocks during stalls on MMX instructions writing to E- or M-state lines	

## 14. Updates to Chapter 22, Volume 3B

Change bars show changes to Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

-----  
Changes to this chapter: Added information on undefined opcodes.

Intel 64 and IA-32 processors are binary compatible. Compatibility means that, within limited constraints, programs that execute on previous generations of processors will produce identical results when executed on later processors. The compatibility constraints and any implementation differences between the Intel 64 and IA-32 processors are described in this chapter.

Each new processor has enhanced the software visible architecture from that found in earlier Intel 64 and IA-32 processors. Those enhancements have been defined with consideration for compatibility with previous and future processors. This chapter also summarizes the compatibility considerations for those extensions.

## 22.1 PROCESSOR FAMILIES AND CATEGORIES

IA-32 processors are referred to in several different ways in this chapter, depending on the type of compatibility information being related, as described in the following:

- **IA-32 Processors** — All the Intel processors based on the Intel IA-32 Architecture, which include the 8086/88, Intel 286, Intel386, Intel486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4, and Intel Xeon processors.
- **32-bit Processors** — All the IA-32 processors that use a 32-bit architecture, which include the Intel386, Intel486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4, and Intel Xeon processors.
- **16-bit Processors** — All the IA-32 processors that use a 16-bit architecture, which include the 8086/88 and Intel 286 processors.
- **P6 Family Processors** — All the IA-32 processors that are based on the P6 microarchitecture, which include the Pentium Pro, Pentium II, and Pentium III processors.
- **Pentium® 4 Processors** — A family of IA-32 and Intel 64 processors that are based on the Intel NetBurst® microarchitecture.
- **Intel® Pentium® M Processors** — A family of IA-32 processors that are based on the Intel Pentium M processor microarchitecture.
- **Intel® Core™ Duo and Solo Processors** — Families of IA-32 processors that are based on an improved Intel Pentium M processor microarchitecture.
- **Intel® Xeon® Processors** — A family of IA-32 and Intel 64 processors that are based on the Intel NetBurst microarchitecture. This family includes the Intel Xeon processor and the Intel Xeon processor MP based on the Intel NetBurst microarchitecture. Intel Xeon processors 3000, 3100, 3200, 3300, 3200, 5100, 5200, 5300, 5400, 7200, 7300 series are based on Intel Core microarchitectures and support Intel 64 architecture.
- **Pentium® D Processors** — A family of dual-core Intel 64 processors that provides two processor cores in a physical package. Each core is based on the Intel NetBurst microarchitecture.
- **Pentium® Processor Extreme Editions** — A family of dual-core Intel 64 processors that provides two processor cores in a physical package. Each core is based on the Intel NetBurst microarchitecture and supports Intel Hyper-Threading Technology.
- **Intel® Core™ 2 Processor family**— A family of Intel 64 processors that are based on the Intel Core microarchitecture. Intel Pentium Dual-Core processors are also based on the Intel Core microarchitecture.
- **Intel® Atom™ Processors** — A family of IA-32 and Intel 64 processors. 45 nm Intel Atom processors are based on the Intel Atom microarchitecture. 32 nm Intel Atom processors are based on newer microarchitectures including the Silvermont microarchitecture and the Airmont microarchitecture. Each generation of Intel Atom processors can be identified by the CPUID's DisplayFamily\_DisplayModel signature; see Table 2-1 "CPUID Signature Values of DisplayFamily\_DisplayModel" in Chapter 2, "Model-Specific Registers (MSRs)" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*.

## 22.2 RESERVED BITS

Throughout this manual, certain bits are marked as reserved in many register and memory layout descriptions. When bits are marked as undefined or reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown effect. Software should follow these guidelines in dealing with reserved bits:

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

Software written for existing IA-32 processor that handles reserved bits correctly will port to future IA-32 processors without generating protection exceptions.

## 22.3 ENABLING NEW FUNCTIONS AND MODES

Most of the new control functions defined for the P6 family and Pentium processors are enabled by new mode flags in the control registers (primarily register CR4). This register is undefined for IA-32 processors earlier than the Pentium processor. Attempting to access this register with an Intel486 or earlier IA-32 processor results in an invalid-opcode exception (#UD). Consequently, programs that execute correctly on the Intel486 or earlier IA-32 processor cannot erroneously enable these functions. Attempting to set a reserved bit in register CR4 to a value other than its original value results in a general-protection exception (#GP). So, programs that execute on the P6 family and Pentium processors cannot erroneously enable functions that may be implemented in future IA-32 processors.

The P6 family and Pentium processors do not check for attempts to set reserved bits in model-specific registers; however these bits may be checked on more recent processors. It is the obligation of the software writer to enforce this discipline. These reserved bits may be used in future Intel processors.

## 22.4 DETECTING THE PRESENCE OF NEW FEATURES THROUGH SOFTWARE

Software can check for the presence of new architectural features and extensions in either of two ways:

1. Test for the presence of the feature or extension. Software can test for the presence of new flags in the EFLAGS register and control registers. If these flags are reserved (meaning not present in the processor executing the test), an exception is generated. Likewise, software can attempt to execute a new instruction, which results in an invalid-opcode exception (#UD) being generated if it is not supported.
2. Execute the CPUID instruction. The CPUID instruction (added to the IA-32 in the Pentium processor) indicates the presence of new features directly.

See Chapter 19, "Processor Identification and Feature Determination," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for detailed information on detecting new processor features and extensions.

## 22.5 INTEL MMX TECHNOLOGY

The Pentium processor with MMX technology introduced the MMX technology and a set of MMX instructions to the IA-32. The MMX instructions are described in Chapter 9, "Programming with Intel® MMX™ Technology," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, and in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*. The MMX technology and MMX instructions are also included in the Pentium II, Pentium III, Pentium 4, and Intel Xeon processors.

## 22.6 STREAMING SIMD EXTENSIONS (SSE)

The Streaming SIMD Extensions (SSE) were introduced in the Pentium III processor. The SSE extensions consist of a new set of instructions and a new set of registers. The new registers include the eight 128-bit XMM registers and the 32-bit MXCSR control and status register. These instructions and registers are designed to allow SIMD computations to be made on single-precision floating-point numbers. Several of these new instructions also operate in the MMX registers. SSE instructions and registers are described in Section 10, "Programming with Streaming SIMD Extensions (SSE)," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, and in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*.

## 22.7 STREAMING SIMD EXTENSIONS 2 (SSE2)

The Streaming SIMD Extensions 2 (SSE2) were introduced in the Pentium 4 and Intel Xeon processors. They consist of a new set of instructions that operate on the XMM and MXCSR registers and perform SIMD operations on double-precision floating-point values and on integer values. Several of these new instructions also operate in the MMX registers. SSE2 instructions and registers are described in Chapter 11, "Programming with Streaming SIMD Extensions 2 (SSE2)," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, and in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*.

## 22.8 STREAMING SIMD EXTENSIONS 3 (SSE3)

The Streaming SIMD Extensions 3 (SSE3) were introduced in Pentium 4 processors supporting Intel Hyper-Threading Technology and Intel Xeon processors. SSE3 extensions include 13 instructions. Ten of these 13 instructions support the single instruction multiple data (SIMD) execution model used with SSE/SSE2 extensions. One SSE3 instruction accelerates x87 style programming for conversion to integer. The remaining two instructions (MONITOR and MWAIT) accelerate synchronization of threads. SSE3 instructions are described in Chapter 12, "Programming with SSE3, SSSE3 and SSE4," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, and in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*.

## 22.9 ADDITIONAL STREAMING SIMD EXTENSIONS

The Supplemental Streaming SIMD Extensions 3 (SSSE3) were introduced in the Intel Core 2 processor and Intel Xeon processor 5100 series. Streaming SIMD Extensions 4 provided 54 new instructions introduced in 45 nm Intel Xeon processors and Intel Core 2 processors. SSSE3, SSE4.1 and SSE4.2 instructions are described in Chapter 12, "Programming with SSE3, SSSE3 and SSE4," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, and in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*.

## 22.10 INTEL HYPER-THREADING TECHNOLOGY

Intel Hyper-Threading Technology provides two logical processors that can execute two separate code streams (called *threads*) concurrently by using shared resources in a single processor core or in a physical package.

This feature was introduced in the Intel Xeon processor MP and later steppings of the Intel Xeon processor, and Pentium 4 processors supporting Intel Hyper-Threading Technology. The feature is also found in the Pentium processor Extreme Edition. See also: Section 8.7, "Intel® Hyper-Threading Technology Architecture."

45 nm and 32 nm Intel Atom processors support Intel Hyper-Threading Technology.

Intel Atom processors based on Silvermont and Airmont microarchitectures do not support Intel Hyper-Threading Technology

## 22.11 MULTI-CORE TECHNOLOGY

The Pentium D processor and Pentium processor Extreme Edition provide two processor cores in each physical processor package. See also: Section 8.5, “Intel® Hyper-Threading Technology and Intel® Multi-Core Technology,” and Section 8.8, “Multi-Core Architecture.” Intel Core 2 Duo, Intel Pentium Dual-Core processors, Intel Xeon processors 3000, 3100, 5100, 5200 series provide two processor cores in each physical processor package. Intel Core 2 Extreme, Intel Core 2 Quad processors, Intel Xeon processors 3200, 3300, 5300, 5400, 7300 series provide two processor cores in each physical processor package.

## 22.12 SPECIFIC FEATURES OF DUAL-CORE PROCESSOR

Dual-core processors may have some processor-specific features. Use CPUID feature flags to detect the availability features. Note the following:

- **CPUID Brand String** — On Pentium processor Extreme Edition, the process will report the correct brand string only after the correct microcode updates are loaded.
- **Enhanced Intel SpeedStep Technology** — This feature is supported in Pentium D processor but not in Pentium processor Extreme Edition.

## 22.13 NEW INSTRUCTIONS IN THE PENTIUM AND LATER IA-32 PROCESSORS

Table 22-1 identifies the instructions introduced into the IA-32 in the Pentium processor and later IA-32 processors.

### 22.13.1 Instructions Added Prior to the Pentium Processor

The following instructions were added in the Intel486 processor:

- BSWAP (byte swap) instruction.
- XADD (exchange and add) instruction.
- CMPXCHG (compare and exchange) instruction.
- INVD (invalidate cache) instruction.
- WBINVD (write-back and invalidate cache) instruction.
- INVLPG (invalidate TLB entry) instruction.

**Table 22-1. New Instruction in the Pentium Processor and Later IA-32 Processors**

Instruction	CPUID Identification Bits	Introduced In
CMOV <sub>cc</sub> (conditional move)	EDX, Bit 15	Pentium Pro processor
FCMOV <sub>cc</sub> (floating-point conditional move)	EDX, Bits 0 and 15	
FCOMI (floating-point compare and set EFLAGS)	EDX, Bits 0 and 15	
RDPMC (read performance monitoring counters)	EAX, Bits 8-11, set to 6H; see Note 1	
UD2 (undefined)	EAX, Bits 8-11, set to 6H	
CMPXCHG8B (compare and exchange 8 bytes)	EDX, Bit 8	Pentium processor
CPUID (CPU identification)	None; see Note 2	
RDTSC (read time-stamp counter)	EDX, Bit 4	
RDMSR (read model-specific register)	EDX, Bit 5	
WRMSR (write model-specific register)	EDX, Bit 5	
MMX Instructions	EDX, Bit 23	



**Table 22-1. New Instruction in the Pentium Processor and Later IA-32 Processors (Contd.)**

Instruction	CPUID Identification Bits	Introduced In
-------------	---------------------------	---------------

**NOTES:**

1. The RDPNC instruction was introduced in the P6 family of processors and added to later model Pentium processors. This instruction is model specific in nature and not architectural.
2. The CPUID instruction is available in all Pentium and P6 family processors and in later models of the Intel486 processors. The ability to set and clear the ID flag (bit 21) in the EFLAGS register indicates the availability of the CPUID instruction.

The following instructions were added in the Intel386 processor:

- LSS, LFS, and LGS (load SS, FS, and GS registers).
- Long-displacement conditional jumps.
- Single-bit instructions.
- Bit scan instructions.
- Double-shift instructions.
- Byte set on condition instruction.
- Move with sign/zero extension.
- Generalized multiply instruction.
- MOV to and from control registers.
- MOV to and from test registers (now obsolete).
- MOV to and from debug registers.
- RSM (resume from SMM). This instruction was introduced in the Intel386 SL and Intel486 SL processors.

The following instructions were added in the Intel 387 math coprocessor:

- FPREM1.
- FUCOM, FUCOMP, and FUCOMPP.

## 22.14 OBSOLETE INSTRUCTIONS

The MOV to and from test registers instructions were removed from the Pentium processor and future IA-32 processors. Execution of these instructions generates an invalid-opcode exception (#UD).

## 22.15 UNDEFINED OPCODES

All new instructions defined for Intel 64 and IA-32 processors use binary encodings that were reserved on earlier-generation processors. Generally, attempting to execute a reserved opcode results in an invalid-opcode (#UD) exception being generated. Consequently, programs that execute correctly on earlier-generation processors cannot erroneously execute these instructions and thereby produce unexpected results when executed on later Intel 64 processors.

For compatibility with prior generations, there are a few reserved opcodes which do not result in a #UD but rather result in the same behavior as certain defined instructions. In the interest of standardization, it is recommended that software not use the opcodes given below but instead use those defined in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*.

The following items enumerate those reserved opcodes (referring in some cases to opcode groups as defined in Appendix A, "Opcode Map" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D*).

- **Immediate Group 1** - When not in 64-bit mode, instructions encoded with opcode 82H result in the behavior of the corresponding instructions encoded with opcode 80H. Depending on the Op/Reg field of the ModR/M Byte, these opcodes are the byte forms of ADD, OR, ADC, SBB, AND, SUB, XOR, CMP. (In 64-bit mode, these opcodes cause a #UD.)

- **Shift Group 2 /6** - Instructions encoded with opcodes C0H, C1H, D0H, D1H, D2H, and D3H with value 110B in the Op/Reg field (/6) of the ModR/M Byte result in the behavior of the corresponding instructions with value 100B in the Op/Reg field (/4). These are various forms of the SAL/SHL instruction.
- **Unary Group 3 /1** - Instructions encoded with opcodes F6H and F7H with value 001B in the Op/Reg field (/01) of the ModR/M Byte result in the behavior of the corresponding instructions with value 000B in the Op/Reg field (/0). These are various forms of the TEST instruction.
- **Reserved NOP** - Instructions encoded with the opcode 0F0DH or with the opcodes 0F18H through 0F1FH result in the behavior of the NOP (No Operation) instruction, except for those opcodes defined in the the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*. The opcodes not so defined are considered "Reserved NOP" and may be used for future instructions which have no defined impact on existing architectural state. These reserved NOP opcodes are decoded with a ModR/M byte and typical instruction prefix options but still result in the behavior of the NOP instruction.
- **x87 Opcodes** - There are several groups of x87 opcodes which provide the same behavior as other x87 instructions. See Section 22.18.9 for the complete list.

There are a few reserved opcodes that provide unique behavior but do not provide capabilities that are not already available in the main instructions defined in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*.

- **F1H** - INT1 has subtly different behavior from CD01H, Interrupt with vector 01.
- **D6H** - When not in 64-bit mode SALC - Set AL to Carry flag. IF (CF=1), AL=FF, ELSE, AL=0 (#UD in 64-bit mode)
- **x87 Opcodes** - There are a few x87 opcodes with subtly different behavior from existing x87 instructions. See Section 22.18.9 for details.

## 22.16 NEW FLAGS IN THE EFLAGS REGISTER

The section titled "EFLAGS Register" in Chapter 3, "Basic Execution Environment," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, shows the configuration of flags in the EFLAGS register for the P6 family processors. No new flags have been added to this register in the P6 family processors. The flags added to this register in the Pentium and Intel486 processors are described in the following sections.

The following flags were added to the EFLAGS register in the Pentium processor:

- VIF (virtual interrupt flag), bit 19.
- VIP (virtual interrupt pending), bit 20.
- ID (identification flag), bit 21.

The AC flag (bit 18) was added to the EFLAGS register in the Intel486 processor.

### 22.16.1 Using EFLAGS Flags to Distinguish Between 32-Bit IA-32 Processors

The following bits in the EFLAGS register that can be used to differentiate between the 32-bit IA-32 processors:

- Bit 18 (the AC flag) can be used to distinguish an Intel386 processor from the P6 family, Pentium, and Intel486 processors. Since it is not implemented on the Intel386 processor, it will always be clear.
- Bit 21 (the ID flag) indicates whether an application can execute the CPUID instruction. The ability to set and clear this bit indicates that the processor is a P6 family or Pentium processor. The CPUID instruction can then be used to determine which processor.
- Bits 19 (the VIF flag) and 20 (the VIP flag) will always be zero on processors that do not support virtual mode extensions, which includes all 32-bit processors prior to the Pentium processor.

See Chapter 19, "Processor Identification and Feature Determination," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for more information on identifying processors.

## 22.17 STACK OPERATIONS AND USER SOFTWARE

This section identifies the differences in stack implementation between the various IA-32 processors.

### 22.17.1 PUSH SP

The P6 family, Pentium, Intel486, Intel386, and Intel 286 processors push a different value on the stack for a PUSH SP instruction than the 8086 processor. The 32-bit processors push the value of the SP register before it is decremented as part of the push operation; the 8086 processor pushes the value of the SP register after it is decremented. If the value pushed is important, replace PUSH SP instructions with the following three instructions:

```
PUSH BP
MOV BP, SP
XCHG BP, [BP]
```

This code functions as the 8086 processor PUSH SP instruction on the P6 family, Pentium, Intel486, Intel386, and Intel 286 processors.

### 22.17.2 EFLAGS Pushed on the Stack

The setting of the stored values of bits 12 through 15 (which includes the IOPL field and the NT flag) in the EFLAGS register by the PUSHF instruction, by interrupts, and by exceptions is different with the 32-bit IA-32 processors than with the 8086 and Intel 286 processors. The differences are as follows:

- 8086 processor—bits 12 through 15 are always set.
- Intel 286 processor—bits 12 through 15 are always cleared in real-address mode.
- 32-bit processors in real-address mode—bit 15 (reserved) is always cleared, and bits 12 through 14 have the last value loaded into them.

## 22.18 X87 FPU

This section addresses the issues that must be faced when porting floating-point software designed to run on earlier IA-32 processors and math coprocessors to a Pentium 4, Intel Xeon, P6 family, or Pentium processor with integrated x87 FPU. To software, a Pentium 4, Intel Xeon, or P6 family processor looks very much like a Pentium processor. Floating-point software which runs on a Pentium or Intel486 DX processor, or on an Intel486 SX processor/Intel 487 SX math coprocessor system or an Intel386 processor/Intel 387 math coprocessor system, will run with at most minor modifications on a Pentium 4, Intel Xeon, or P6 family processor. To port code directly from an Intel 286 processor/Intel 287 math coprocessor system or an Intel 8086 processor/8087 math coprocessor system to a Pentium 4, Intel Xeon, P6 family, or Pentium processor, certain additional issues must be addressed.

In the following sections, the term “32-bit x87 FPUs” refers to the P6 family, Pentium, and Intel486 DX processors, and to the Intel 487 SX and Intel 387 math coprocessors; the term “16-bit IA-32 math coprocessors” refers to the Intel 287 and 8087 math coprocessors.

### 22.18.1 Control Register CR0 Flags

The ET, NE, and MP flags in control register CR0 control the interface between the integer unit of an IA-32 processor and either its internal x87 FPU or an external math coprocessor. The effect of these flags in the various IA-32 processors are described in the following paragraphs.

The ET (extension type) flag (bit 4 of the CR0 register) is used in the Intel386 processor to indicate whether the math coprocessor in the system is an Intel 287 math coprocessor (flag is clear) or an Intel 387 DX math coprocessor (flag is set). This bit is hardwired to 1 in the P6 family, Pentium, and Intel486 processors.

The NE (Numeric Exception) flag (bit 5 of the CR0 register) is used in the P6 family, Pentium, and Intel486 processors to determine whether unmasked floating-point exceptions are reported internally through interrupt vector 16

(flag is set) or externally through an external interrupt (flag is clear). On a hardware reset, the NE flag is initialized to 0, so software using the automatic internal error-reporting mechanism must set this flag to 1. This flag is nonexistent on the Intel386 processor.

As on the Intel 286 and Intel386 processors, the MP (monitor coprocessor) flag (bit 1 of register CR0) determines whether the WAIT/FWAIT instructions or waiting-type floating-point instructions trap when the context of the x87 FPU is different from that of the currently-executing task. If the MP and TS flag are set, then a WAIT/FWAIT instruction and waiting instructions will cause a device-not-available exception (interrupt vector 7). The MP flag is used on the Intel 286 and Intel386 processors to support the use of a WAIT/FWAIT instruction to wait on a device other than a math coprocessor. The device reports its status through the BUSY# pin. Since the P6 family, Pentium, and Intel486 processors do not have such a pin, the MP flag has no relevant use and should be set to 1 for normal operation.

## 22.18.2 x87 FPU Status Word

This section identifies differences to the x87 FPU status word for the different IA-32 processors and math coprocessors, the reason for the differences, and their impact on software.

### 22.18.2.1 Condition Code Flags (C0 through C3)

The following information pertains to differences in the use of the condition code flags (C0 through C3) located in bits 8, 9, 10, and 14 of the x87 FPU status word.

After execution of a FINIT instruction or a hardware reset on a 32-bit x87 FPU, the condition code flags are set to 0. The same operations on a 16-bit IA-32 math coprocessor leave these flags intact (they contain their prior value). This difference in operation has no impact on software and provides a consistent state after reset.

Transcendental instruction results in the core range of the P6 family and Pentium processors may differ from the Intel486 DX processor and Intel 487 SX math coprocessor by 2 to 3 units in the last place (ulps)—(see “Transcendental Instruction Accuracy” in Chapter 8, “Programming with the x87 FPU,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*). As a result, the value saved in the C1 flag may also differ.

After an incomplete FPREM/FPREM1 instruction, the C0, C1, and C3 flags are set to 0 on the 32-bit x87 FPUs. After the same operation on a 16-bit IA-32 math coprocessor, these flags are left intact.

On the 32-bit x87 FPUs, the C2 flag serves as an incomplete flag for the FTAN instruction. On the 16-bit IA-32 math coprocessors, the C2 flag is undefined for the FPTAN instruction. This difference has no impact on software, because Intel 287 or 8087 programs do not check C2 after an FPTAN instruction. The use of this flag on later processors allows fast checking of operand range.

### 22.18.2.2 Stack Fault Flag

When unmasked stack overflow or underflow occurs on a 32-bit x87 FPU, the IE flag (bit 0) and the SF flag (bit 6) of the x87 FPU status word are set to indicate a stack fault and condition code flag C1 is set or cleared to indicate overflow or underflow, respectively. When unmasked stack overflow or underflow occurs on a 16-bit IA-32 math coprocessor, only the IE flag is set. Bit 6 is reserved on these processors. The addition of the SF flag on a 32-bit x87 FPU has no impact on software. Existing exception handlers need not change, but may be upgraded to take advantage of the additional information.

## 22.18.3 x87 FPU Control Word

Only affine closure is supported for infinity control on a 32-bit x87 FPU. The infinity control flag (bit 12 of the x87 FPU control word) remains programmable on these processors, but has no effect. This change was made to conform to the IEEE Standard 754 for Binary Floating-Point Arithmetic. On a 16-bit IA-32 math coprocessor, both affine and projective closures are supported, as determined by the setting of bit 12. After a hardware reset, the default value of bit 12 is projective. Software that requires projective infinity arithmetic may give different results.

## 22.18.4 x87 FPU Tag Word

When loading the tag word of a 32-bit x87 FPU, using an `FLDENV`, `FRSTOR`, or `FXRSTOR` (Pentium III processor only) instruction, the processor examines the incoming tag and classifies the location only as empty or non-empty. Thus, tag values of 00, 01, and 10 are interpreted by the processor to indicate a non-empty location. The tag value of 11 is interpreted by the processor to indicate an empty location. Subsequent operations on a non-empty register always examine the value in the register, not the value in its tag. The `FSTENV`, `FSAVE`, and `FXSAVE` (Pentium III processor only) instructions examine the non-empty registers and put the correct values in the tags before storing the tag word.

The corresponding tag for a 16-bit IA-32 math coprocessor is checked before each register access to determine the class of operand in the register; the tag is updated after every change to a register so that the tag always reflects the most recent status of the register. Software can load a tag with a value that disagrees with the contents of a register (for example, the register contains a valid value, but the tag says special). Here, the 16-bit IA-32 math coprocessors honor the tag and do not examine the register.

Software written to run on a 16-bit IA-32 math coprocessor may not operate correctly on a 16-bit x87 FPU, if it uses the `FLDENV`, `FRSTOR`, or `FXRSTOR` instructions to change tags to values (other than to empty) that are different from actual register contents.

The encoding in the tag word for the 32-bit x87 FPUs for unsupported data formats (including pseudo-zero and unnormal) is special (10B), to comply with IEEE Standard 754. The encoding in the 16-bit IA-32 math coprocessors for pseudo-zero and unnormal is valid (00B) and the encoding for other unsupported data formats is special (10B). Code that recognizes the pseudo-zero or unnormal format as valid must therefore be changed if it is ported to a 32-bit x87 FPU.

## 22.18.5 Data Types

This section discusses the differences of data types for the various x87 FPUs and math coprocessors.

### 22.18.5.1 NaNs

The 32-bit x87 FPUs distinguish between signaling NaNs (SNaNs) and quiet NaNs (QNaNs). These x87 FPUs only generate QNaNs and normally do not generate an exception upon encountering a QNaN. An invalid-operation exception (`#I`) is generated only upon encountering a SNaN, except for the `FCOM`, `FIST`, and `FBSTP` instructions, which also generates an invalid-operation exceptions for a QNaNs. This behavior matches IEEE Standard 754.

The 16-bit IA-32 math coprocessors only generate one kind of NaN (the equivalent of a QNaN), but the raise an invalid-operation exception upon encountering any kind of NaN.

When porting software written to run on a 16-bit IA-32 math coprocessor to a 32-bit x87 FPU, uninitialized memory locations that contain QNaNs should be changed to SNaNs to cause the x87 FPU or math coprocessor to fault when uninitialized memory locations are referenced.

### 22.18.5.2 Pseudo-zero, Pseudo-NaN, Pseudo-infinity, and Unnormal Formats

The 32-bit x87 FPUs neither generate nor support the pseudo-zero, pseudo-NaN, pseudo-infinity, and unnormal formats. Whenever they encounter them in an arithmetic operation, they raise an invalid-operation exception. The 16-bit IA-32 math coprocessors define and support special handling for these formats. Support for these formats was dropped to conform with IEEE Standard 754 for Binary Floating-Point Arithmetic.

This change should not impact software ported from 16-bit IA-32 math coprocessors to 32-bit x87 FPUs. The 32-bit x87 FPUs do not generate these formats, and therefore will not encounter them unless software explicitly loads them in the data registers. The only affect may be in how software handles the tags in the tag word (see also: Section 22.18.4, "x87 FPU Tag Word").

## 22.18.6 Floating-Point Exceptions

This section identifies the implementation differences in exception handling for floating-point instructions in the various x87 FPUs and math coprocessors.

### 22.18.6.1 Denormal Operand Exception (#D)

When the denormal operand exception is masked, the 32-bit x87 FPUs automatically normalize denormalized numbers when possible; whereas, the 16-bit IA-32 math coprocessors return a denormal result. A program written to run on a 16-bit IA-32 math coprocessor that uses the denormal exception solely to normalize denormalized operands is redundant when run on the 32-bit x87 FPUs. If such a program is run on 32-bit x87 FPUs, performance can be improved by masking the denormal exception. Floating-point programs run faster when the FPU performs normalization of denormalized operands.

The denormal operand exception is not raised for transcendental instructions and the FEXTRACT instruction on the 16-bit IA-32 math coprocessors. This exception is raised for these instructions on the 32-bit x87 FPUs. The exception handlers ported to these latter processors need to be changed only if the handlers give special treatment to different opcodes.

### 22.18.6.2 Numeric Overflow Exception (#O)

On the 32-bit x87 FPUs, when the numeric overflow exception is masked and the rounding mode is set to chop (toward 0), the result is the largest positive or smallest negative number. The 16-bit IA-32 math coprocessors do not signal the overflow exception when the masked response is not  $\infty$ ; that is, they signal overflow only when the rounding control is not set to round to 0. If rounding is set to chop (toward 0), the result is positive or negative  $\infty$ . Under the most common rounding modes, this difference has no impact on existing software.

If rounding is toward 0 (chop), a program on a 32-bit x87 FPU produces, under overflow conditions, a result that is different in the least significant bit of the significand, compared to the result on a 16-bit IA-32 math coprocessor. The reason for this difference is IEEE Standard 754 compatibility.

When the overflow exception is not masked, the precision exception is flagged on the 32-bit x87 FPUs. When the result is stored in the stack, the significand is rounded according to the precision control (PC) field of the FPU control word or according to the opcode. On the 16-bit IA-32 math coprocessors, the precision exception is not flagged and the significand is not rounded. The impact on existing software is that if the result is stored on the stack, a program running on a 32-bit x87 FPU produces a different result under overflow conditions than on a 16-bit IA-32 math coprocessor. The difference is apparent only to the exception handler. This difference is for IEEE Standard 754 compatibility.

### 22.18.6.3 Numeric Underflow Exception (#U)

When the underflow exception is masked on the 32-bit x87 FPUs, the underflow exception is signaled when the result is tiny and inexact (see Section 4.9.1.5, "Numeric Underflow Exception (#U)" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*). When the underflow exception is unmasked and the instruction is supposed to store the result on the stack, the significand is rounded to the appropriate precision (according to the PC flag in the FPU control word, for those instructions controlled by PC, otherwise to extended precision), after adjusting the exponent.

### 22.18.6.4 Exception Precedence

There is no difference in the precedence of the denormal-operand exception on the 32-bit x87 FPUs, whether it be masked or not. When the denormal-operand exception is not masked on the 16-bit IA-32 math coprocessors, it takes precedence over all other exceptions. This difference causes no impact on existing software, but some unneeded normalization of denormalized operands is prevented on the Intel486 processor and Intel 387 math coprocessor.

### 22.18.6.5 CS and EIP For FPU Exceptions

On the Intel 32-bit x87 FPUs, the values from the CS and EIP registers saved for floating-point exceptions point to any prefixes that come before the floating-point instruction. On the 8087 math coprocessor, the saved CS and IP registers points to the floating-point instruction.



### 22.18.6.6 FPU Error Signals

The floating-point error signals to the P6 family, Pentium, and Intel486 processors do not pass through an interrupt controller; an INT# signal from an Intel 387, Intel 287 or 8087 math coprocessors does. If an 8086 processor uses another exception for the 8087 interrupt, both exception vectors should call the floating-point-error exception handler. Some instructions in a floating-point-error exception handler may need to be deleted if they use the interrupt controller. The P6 family, Pentium, and Intel486 processors have signals that, with the addition of external logic, support reporting for emulation of the interrupt mechanism used in many personal computers.

On the P6 family, Pentium, and Intel486 processors, an undefined floating-point opcode will cause an invalid-opcode exception (#UD, interrupt vector 6). Undefined floating-point opcodes, like legal floating-point opcodes, cause a device not available exception (#NM, interrupt vector 7) when either the TS or EM flag in control register CR0 is set. The P6 family, Pentium, and Intel486 processors do not check for floating-point error conditions on encountering an undefined floating-point opcode.

### 22.18.6.7 Assertion of the FERR# Pin

When using the MS-DOS compatibility mode for handling floating-point exceptions, the FERR# pin must be connected to an input to an external interrupt controller. An external interrupt is then generated when the FERR# output drives the input to the interrupt controller and the interrupt controller in turn drives the INTR pin on the processor.

For the P6 family and Intel386 processors, an unmasked floating-point exception always causes the FERR# pin to be asserted upon completion of the instruction that caused the exception. For the Pentium and Intel486 processors, an unmasked floating-point exception may cause the FERR# pin to be asserted either at the end of the instruction causing the exception or immediately before execution of the next floating-point instruction. (Note that the next floating-point instruction would not be executed until the pending unmasked exception has been handled.) See Appendix D, "Guidelines for Writing x87 FPU Extension Handlers," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a complete description of the required mechanism for handling floating-point exceptions using the MS-DOS compatibility mode.

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

### 22.18.6.8 Invalid Operation Exception On Denormals

An invalid-operation exception is not generated on the 32-bit x87 FPUs upon encountering a denormal value when executing a FSQRT, FDIV, or FPREM instruction or upon conversion to BCD or to integer. The operation proceeds by first normalizing the value. On the 16-bit IA-32 math coprocessors, upon encountering this situation, the invalid-operation exception is generated. This difference has no impact on existing software. Software running on the 32-bit x87 FPUs continues to execute in cases where the 16-bit IA-32 math coprocessors trap. The reason for this change was to eliminate an exception from being raised.

### 22.18.6.9 Alignment Check Exceptions (#AC)

If alignment checking is enabled, a misaligned data operand on the P6 family, Pentium, and Intel486 processors causes an alignment check exception (#AC) when a program or procedure is running at privilege-level 3, except for the stack portion of the FSAVE/FNSAVE, FXSAVE, FRSTOR, and FXRSTOR instructions.

### 22.18.6.10 Segment Not Present Exception During FLDENV

On the Intel486 processor, when a segment not present exception (#NP) occurs in the middle of an FLDENV instruction, it can happen that part of the environment is loaded and part not. In such cases, the FPU control word is left with a value of 007FH. The P6 family and Pentium processors ensure the internal state is correct at all times by attempting to read the first and last bytes of the environment before updating the internal state.

### 22.18.6.11 Device Not Available Exception (#NM)

The device-not-available exception (#NM, interrupt 7) will occur in the P6 family, Pentium, and Intel486 processors as described in Section 2.5, "Control Registers," Table 2-2, and Chapter 6, "Interrupt 7—Device Not Available Exception (#NM)."

### 22.18.6.12 Coprocessor Segment Overrun Exception

The coprocessor segment overrun exception (interrupt 9) does not occur in the P6 family, Pentium, and Intel486 processors. In situations where the Intel 387 math coprocessor would cause an interrupt 9, the P6 family, Pentium, and Intel486 processors simply abort the instruction. To avoid undetected segment overruns, it is recommended that the floating-point save area be placed in the same page as the TSS. This placement will prevent the FPU environment from being lost if a page fault occurs during the execution of an FLDENV, FRSTOR, or FXRSTOR instruction while the operating system is performing a task switch.

### 22.18.6.13 General Protection Exception (#GP)

A general-protection exception (#GP, interrupt 13) occurs if the starting address of a floating-point operand falls outside a segment's size. An exception handler should be included to report these programming errors.

### 22.18.6.14 Floating-Point Error Exception (#MF)

In real mode and protected mode (not including virtual-8086 mode), interrupt vector 16 must point to the floating-point exception handler. In virtual-8086 mode, the virtual-8086 monitor can be programmed to accommodate a different location of the interrupt vector for floating-point exceptions.

## 22.18.7 Changes to Floating-Point Instructions

This section identifies the differences in floating-point instructions for the various Intel FPU and math coprocessor architectures, the reason for the differences, and their impact on software.

### 22.18.7.1 FDIV, FPREM, and FSQRT Instructions

The 32-bit x87 FPUs support operations on denormalized operands and, when detected, an underflow exception can occur, for compatibility with the IEEE Standard 754. The 16-bit IA-32 math coprocessors do not operate on denormalized operands or return underflow results. Instead, they generate an invalid-operation exception when they detect an underflow condition. An existing underflow exception handler will require change only if it gives different treatment to different opcodes. Also, it is possible that fewer invalid-operation exceptions will occur.

### 22.18.7.2 FSCALE Instruction

With the 32-bit x87 FPUs, the range of the scaling operand is not restricted. If  $(0 < |ST(1)| < 1)$ , the scaling factor is 0; therefore,  $ST(0)$  remains unchanged. If the rounded result is not exact or if there was a loss of accuracy (masked underflow), the precision exception is signaled. With the 16-bit IA-32 math coprocessors, the range of the scaling operand is restricted. If  $(0 < |ST(1)| < 1)$ , the result is undefined and no exception is signaled. The impact of this difference on existing software is that different results are delivered on the 32-bit and 16-bit FPUs and math coprocessors when  $(0 < |ST(1)| < 1)$ .

### 22.18.7.3 FPREM1 Instruction

The 32-bit x87 FPUs compute a partial remainder according to IEEE Standard 754. This instruction does not exist on the 16-bit IA-32 math coprocessors. The availability of the FPREM1 instruction has no impact on existing software.



#### 22.18.7.4 FPREM Instruction

On the 32-bit x87 FPUs, the condition code flags C0, C3, C1 in the status word correctly reflect the three low-order bits of the quotient following execution of the FPREM instruction. On the 16-bit IA-32 math coprocessors, the quotient bits are incorrect when performing a reduction of  $(64^N + M)$  when  $(N \geq 1)$  and M is 1 or 2. This difference does not affect existing software; software that works around the bug should not be affected.

#### 22.18.7.5 FUCOM, FUCOMP, and FUCOMPP Instructions

When executing the FUCOM, FUCOMP, and FUCOMPP instructions, the 32-bit x87 FPUs perform unordered compare according to IEEE Standard 754. These instructions do not exist on the 16-bit IA-32 math coprocessors. The availability of these new instructions has no impact on existing software.

#### 22.18.7.6 FPTAN Instruction

On the 32-bit x87 FPUs, the range of the operand for the FPTAN instruction is much less restricted ( $|ST(0)| < 2^{63}$ ) than on earlier math coprocessors. The instruction reduces the operand internally using an internal  $\pi/4$  constant that is more accurate. The range of the operand is restricted to  $(|ST(0)| < \pi/4)$  on the 16-bit IA-32 math coprocessors; the operand must be reduced to this range using FPREM. This change has no impact on existing software. See also sections 8.3.8 and section 8.3.10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* for more information on the accuracy of the FPTAN instruction.

#### 22.18.7.7 Stack Overflow

On the 32-bit x87 FPUs, if an FPU stack overflow occurs when the invalid-operation exception is masked, the FPU returns the real, integer, or BCD-integer indefinite value to the destination operand, depending on the instruction being executed. On the 16-bit IA-32 math coprocessors, the original operand remains unchanged following a stack overflow, but it is loaded into register ST(1). This difference has no impact on existing software.

#### 22.18.7.8 FSIN, FCOS, and FSINCOS Instructions

On the 32-bit x87 FPUs, these instructions perform three common trigonometric functions. These instructions do not exist on the 16-bit IA-32 math coprocessors. The availability of these instructions has no impact on existing software, but using them provides a performance upgrade. See also sections 8.3.8 and section 8.3.10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* for more information on the accuracy of the FSIN, FCOS, and FSINCOS instructions.

#### 22.18.7.9 FPATAN Instruction

On the 32-bit x87 FPUs, the range of operands for the FPATAN instruction is unrestricted. On the 16-bit IA-32 math coprocessors, the absolute value of the operand in register ST(0) must be smaller than the absolute value of the operand in register ST(1). This difference has impact on existing software.

#### 22.18.7.10 F2XM1 Instruction

The 32-bit x87 FPUs support a wider range of operands  $(-1 < ST(0) < +1)$  for the F2XM1 instruction. The supported operand range for the 16-bit IA-32 math coprocessors is  $(0 \leq ST(0) \leq 0.5)$ . This difference has no impact on existing software.

#### 22.18.7.11 FLD Instruction

On the 32-bit x87 FPUs, when using the FLD instruction to load an extended-real value, a denormal-operand exception is not generated because the instruction is not arithmetic. The 16-bit IA-32 math coprocessors do report a denormal-operand exception in this situation. This difference does not affect existing software.

On the 32-bit x87 FPUs, loading a denormal value that is in single- or double-real format causes the value to be converted to extended-real format. Loading a denormal value on the 16-bit IA-32 math coprocessors causes the

value to be converted to an unnormal. If the next instruction is FXTRACT or FXAM, the 32-bit x87 FPUs will give a different result than the 16-bit IA-32 math coprocessors. This change was made for IEEE Standard 754 compatibility.

On the 32-bit x87 FPUs, loading an SNaN that is in single- or double-real format causes the FPU to generate an invalid-operation exception. The 16-bit IA-32 math coprocessors do not raise an exception when loading a signaling NaN. The invalid-operation exception handler for 16-bit math coprocessor software needs to be updated to handle this condition when porting software to 32-bit FPUs. This change was made for IEEE Standard 754 compatibility.

### 22.18.7.12 FXTRACT Instruction

On the 32-bit x87 FPUs, if the operand is 0 for the FXTRACT instruction, the divide-by-zero exception is reported and  $-\infty$  is delivered to register ST(1). If the operand is  $+\infty$ , no exception is reported. If the operand is 0 on the 16-bit IA-32 math coprocessors, 0 is delivered to register ST(1) and no exception is reported. If the operand is  $+\infty$ , the invalid-operation exception is reported. These differences have no impact on existing software. Software usually bypasses 0 and  $\infty$ . This change is due to the IEEE Standard 754 recommendation to fully support the “logb” function.

### 22.18.7.13 Load Constant Instructions

On 32-bit x87 FPUs, rounding control is in effect for the load constant instructions. Rounding control is not in effect for the 16-bit IA-32 math coprocessors. Results for the FLDPI, FLDLN2, FLDLG2, and FLDL2E instructions are the same as for the 16-bit IA-32 math coprocessors when rounding control is set to round to nearest or round to  $+\infty$ . They are the same for the FLDL2T instruction when rounding control is set to round to nearest, round to  $-\infty$ , or round to zero. Results are different from the 16-bit IA-32 math coprocessors in the least significant bit of the mantissa if rounding control is set to round to  $-\infty$  or round to 0 for the FLDPI, FLDLN2, FLDLG2, and FLDL2E instructions; they are different for the FLDL2T instruction if round to  $+\infty$  is specified. These changes were implemented for compatibility with IEEE Standard 754 for Floating-Point Arithmetic recommendations.

### 22.18.7.14 FXAM Instruction

With the 32-bit x87 FPUs, if the FPU encounters an empty register when executing the FXAM instruction, it not generate combinations of C0 through C3 equal to 1101 or 1111. The 16-bit IA-32 math coprocessors may generate these combinations, among others. This difference has no impact on existing software; it provides a performance upgrade to provide repeatable results.

### 22.18.7.15 FSAVE and FSTENV Instructions

With the 32-bit x87 FPUs, the address of a memory operand pointer stored by FSAVE or FSTENV is undefined if the previous floating-point instruction did not refer to memory

## 22.18.8 Transcendental Instructions

The floating-point results of the P6 family and Pentium processors for transcendental instructions in the core range may differ from the Intel486 processors by about 2 or 3 ulps (see “Transcendental Instruction Accuracy” in Chapter 8, “Programming with the x87 FPU,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*). Condition code flag C1 of the status word may differ as a result. The exact threshold for underflow and overflow will vary by a few ulps. The P6 family and Pentium processors’ results will have a worst case error of less than 1 ulp when rounding to the nearest-even and less than 1.5 ulps when rounding in other modes. The transcendental instructions are guaranteed to be monotonic, with respect to the input operands, throughout the domain supported by the instruction.

Transcendental instructions may generate different results in the round-up flag (C1) on the 32-bit x87 FPUs. The round-up flag is undefined for these instructions on the 16-bit IA-32 math coprocessors. This difference has no impact on existing software.

## 22.18.9 Obsolete Instructions and Undefined Opcodes

The 8087 math coprocessor instructions FENI and FDISI, and the Intel 287 math coprocessor instruction FSETPM are treated as integer NOP instructions in the 32-bit x87 FPU. If these opcodes are detected in the instruction stream, no specific operation is performed and no internal states are affected. FSETPM informed the Intel 287 math coprocessor that the processor was in protected mode. The 32-bit x87 FPU handles all addressing and exception-pointer information, whether in protected mode or not.

For compatibility with prior generations there are a few reserved x87 opcodes which do not result in an invalid-opcode (#UD) exception, but rather result in the same behavior as existing defined x87 instructions. In the interest of standardization, it is recommended that the opcodes defined in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D* be used for these operations for standardization.

- DCD0H through DCD7H - Behaves the same as FCOM, D8D0H through D8D7H.
- DCD8H through DCDFH - Behaves the same as FCOMP, D8D8H through D8DFH.
- D0C8H through D0CFH - Behaves the same as FXCH, D9C8H through D9CFH.
- DED0H through DED7H - Behaves the same as FCOMP, D8D8H through D8DFH.
- DFD0H through DFD7H - Behaves the same as FSTP, DDD8H through DDDFH.
- DFC8H through DFCFH - Behaves the same as FXCH, D9C8H through D9CFH.
- DFD8H through DDFH - Behaves the same as FSTP, DDD8H through DDDFH.

There are a few reserved x87 opcodes which provide unique behavior but do not provide capabilities which are not already available in the main instructions defined in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*.

- D9D8H through D9DFH - Behaves the same as FSTP (DDD8H through DDDFH) but won't cause a stack underflow exception.
- DFC0H through DFC7H - Behaves the same as FFREE (DDC0H through DDD7H) with the addition of an x87 stack POP.

## 22.18.10 WAIT/FWAIT Prefix Differences

On the Intel486 processor, when a WAIT/FWAIT instruction precedes a floating-point instruction (one which itself automatically synchronizes with the previous floating-point instruction), the WAIT/FWAIT instruction is treated as a no-op. Pending floating-point exceptions from a previous floating-point instruction are processed not on the WAIT/FWAIT instruction but on the floating-point instruction following the WAIT/FWAIT instruction. In such a case, the report of a floating-point exception may appear one instruction later on the Intel486 processor than on a P6 family or Pentium FPU, or on Intel 387 math coprocessor.

## 22.18.11 Operands Split Across Segments and/or Pages

On the P6 family, Pentium, and Intel486 processor FPUs, when the first half of an operand to be written is inside a page or segment and the second half is outside, a memory fault can cause the first half to be stored but not the second half. In this situation, the Intel 387 math coprocessor stores nothing.

## 22.18.12 FPU Instruction Synchronization

On the 32-bit x87 FPU, all floating-point instructions are automatically synchronized; that is, the processor automatically waits until the previous floating-point instruction has completed before completing the next floating-point instruction. No explicit WAIT/FWAIT instructions are required to assure this synchronization. For the 8087 math coprocessors, explicit waits are required before each floating-point instruction to ensure synchronization. Although 8087 programs having explicit WAIT instructions execute perfectly on the 32-bit IA-32 processors without reassembly, these WAIT instructions are unnecessary.

## 22.19 SERIALIZING INSTRUCTIONS

Certain instructions have been defined to serialize instruction execution to ensure that modifications to flags, registers and memory are completed before the next instruction is executed (or in P6 family processor terminology “committed to machine state”). Because the P6 family processors use branch-prediction and out-of-order execution techniques to improve performance, instruction execution is not generally serialized until the results of an executed instruction are committed to machine state (see Chapter 2, “Intel® 64 and IA-32 Architectures,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).

As a result, at places in a program or task where it is critical to have execution completed for all previous instructions before executing the next instruction (for example, at a branch, at the end of a procedure, or in multiprocessor dependent code), it is useful to add a serializing instruction. See Section 8.3, “Serializing Instructions,” for more information on serializing instructions.

## 22.20 FPU AND MATH COPROCESSOR INITIALIZATION

Table 9-1 shows the states of the FPUs in the P6 family, Pentium, Intel486 processors and of the Intel 387 math coprocessor and Intel 287 coprocessor following a power-up, reset, or INIT, or following the execution of an FINIT/FNINIT instruction. The following is some additional compatibility information concerning the initialization of x87 FPUs and math coprocessors.

### 22.20.1 Intel® 387 and Intel® 287 Math Coprocessor Initialization

Following an Intel386 processor reset, the processor identifies its coprocessor type (Intel® 287 or Intel® 387 DX math coprocessor) by sampling its ERROR# input some time after the falling edge of RESET# signal and before execution of the first floating-point instruction. The Intel 287 coprocessor keeps its ERROR# output in inactive state after hardware reset; the Intel 387 coprocessor keeps its ERROR# output in active state after hardware reset.

Upon hardware reset or execution of the FINIT/FNINIT instruction, the Intel 387 math coprocessor signals an error condition. The P6 family, Pentium, and Intel486 processors, like the Intel 287 coprocessor, do not.

### 22.20.2 Intel486 SX Processor and Intel 487 SX Math Coprocessor Initialization

When initializing an Intel486 SX processor and an Intel 487 SX math coprocessor, the initialization routine should check the presence of the math coprocessor and should set the FPU related flags (EM, MP, and NE) in control register CR0 accordingly (see Section 2.5, “Control Registers,” for a complete description of these flags). Table 22-2 gives the recommended settings for these flags when the math coprocessor is present. The FSTCW instruction will give a value of FFFFH for the Intel486 SX microprocessor and 037FH for the Intel 487 SX math coprocessor.

**Table 22-2. Recommended Values of the EM, MP, and NE Flags for Intel486 SX Microprocessor/Intel 487 SX Math Coprocessor System**

CR0 Flags	Intel486 SX Processor Only	Intel 487 SX Math Coprocessor Present
EM	1	0
MP	0	1
NE	1	0, for MS-DOS* systems 1, for user-defined exception handler

The EM and MP flags in register CR0 are interpreted as shown in Table 22-3.

**Table 22-3. EM and MP Flag Interpretation**

EM	MP	Interpretation
0	0	Floating-point instructions are passed to FPU; WAIT/FWAIT and other waiting-type instructions ignore TS.
0	1	Floating-point instructions are passed to FPU; WAIT/FWAIT and other waiting-type instructions test TS.
1	0	Floating-point instructions trap to emulator; WAIT/FWAIT and other waiting-type instructions ignore TS.
1	1	Floating-point instructions trap to emulator; WAIT/FWAIT and other waiting-type instructions test TS.

Following is an example code sequence to initialize the system and check for the presence of Intel486 SX processor/Intel 487 SX math coprocessor.

```
fninit
fstcw mem_loc
mov ax, mem_loc
cmp ax, 037fh
jz Intel487_SX_Math_CoProcessor_present ;ax=037fh
jmp Intel486_SX_microprocessor_present ;ax=ffffh
```

If the Intel 487 SX math coprocessor is not present, the following code can be run to set the CR0 register for the Intel486 SX processor.

```
mov eax, cr0
and eax, ffffffffh ;make MP=0
or eax, 0024h ;make EM=1, NE=1
mov cr0, eax
```

This initialization will cause any floating-point instruction to generate a device not available exception (#NH), interrupt 7. The software emulation will then take control to execute these instructions. This code is not required if an Intel 487 SX math coprocessor is present in the system. In that case, the typical initialization routine for the Intel486 SX microprocessor will be adequate.

Also, when designing an Intel486 SX processor based system with an Intel 487 SX math coprocessor, timing loops should be independent of frequency and clocks per instruction. One way to attain this is to implement these loops in hardware and not in software (for example, BIOS).

## 22.21 CONTROL REGISTERS

The following sections identify the new control registers and control register flags and fields that were introduced to the 32-bit IA-32 in various processor families. See Figure 2-7 for the location of these flags and fields in the control registers.

The Pentium III processor introduced one new control flag in control register CR4:

- OSXMMEXCPT (bit 10) — The OS will set this bit if it supports unmasked SIMD floating-point exceptions.

The Pentium II processor introduced one new control flag in control register CR4:

- OSFXSR (bit 9) — The OS supports saving and restoring the Pentium III processor state during context switches.

The Pentium Pro processor introduced three new control flags in control register CR4:

- PAE (bit 5) — Physical address extension. Enables paging mechanism to reference extended physical addresses when set; restricts physical addresses to 32 bits when clear (see also: Section 22.22.1.1, “Physical Memory Addressing Extension”).
- PGE (bit 7) — Page global enable. Inhibits flushing of frequently-used or shared pages on CR3 writes (see also: Section 22.22.1.2, “Global Pages”).
- PCE (bit 8) — Performance-monitoring counter enable. Enables execution of the RDPMC instruction at any protection level.

The content of CR4 is 0H following a hardware reset.

Control register CR4 was introduced in the Pentium processor. This register contains flags that enable certain new extensions provided in the Pentium processor:

- VME — Virtual-8086 mode extensions. Enables support for a virtual interrupt flag in virtual-8086 mode (see Section 20.3, “Interrupt and Exception Handling in Virtual-8086 Mode”).
- PVI — Protected-mode virtual interrupts. Enables support for a virtual interrupt flag in protected mode (see Section 20.4, “Protected-Mode Virtual Interrupts”).
- TSD — Time-stamp disable. Restricts the execution of the RDTSC instruction to procedures running at privileged level 0.
- DE — Debugging extensions. Causes an undefined opcode (#UD) exception to be generated when debug registers DR4 and DR5 are references for improved performance (see Section 22.23.3, “Debug Registers DR4 and DR5”).
- PSE — Page size extensions. Enables 4-MByte pages with 32-bit paging when set (see Section 4.3, “32-Bit Paging”).
- MCE — Machine-check enable. Enables the machine-check exception, allowing exception handling for certain hardware error conditions (see Chapter 15, “Machine-Check Architecture”).

The Intel486 processor introduced five new flags in control register CR0:

- NE — Numeric error. Enables the normal mechanism for reporting floating-point numeric errors.
- WP — Write protect. Write-protects read-only pages against supervisor-mode accesses.
- AM — Alignment mask. Controls whether alignment checking is performed. Operates in conjunction with the AC (Alignment Check) flag.
- NW — Not write-through. Enables write-throughs and cache invalidation cycles when clear and disables invalidation cycles and write-throughs that hit in the cache when set.
- CD — Cache disable. Enables the internal cache when clear and disables the cache when set.

The Intel486 processor introduced two new flags in control register CR3:

- PCD — Page-level cache disable. The state of this flag is driven on the PCD# pin during bus cycles that are not paged, such as interrupt acknowledge cycles, when paging is enabled. The PCD# pin is used to control caching in an external cache on a cycle-by-cycle basis.
- PWT — Page-level write-through. The state of this flag is driven on the PWT# pin during bus cycles that are not paged, such as interrupt acknowledge cycles, when paging is enabled. The PWT# pin is used to control write through in an external cache on a cycle-by-cycle basis.

## 22.22 MEMORY MANAGEMENT FACILITIES

The following sections describe the new memory management facilities available in the various IA-32 processors and some compatibility differences.

### 22.22.1 New Memory Management Control Flags

The Pentium Pro processor introduced three new memory management features: physical memory addressing extension, the global bit in page-table entries, and general support for larger page sizes. These features are only available when operating in protected mode.

#### 22.22.1.1 Physical Memory Addressing Extension

The new PAE (physical address extension) flag in control register CR4, bit 5, may enable additional address lines on the processor, allowing extended physical addresses. This option can only be used when paging is enabled, using a new page-table mechanism provided to support the larger physical address range (see Section 4.1, "Paging Modes and Control Bits").

#### 22.22.1.2 Global Pages

The new PGE (page global enable) flag in control register CR4, bit 7, provides a mechanism for preventing frequently used pages from being flushed from the translation lookaside buffer (TLB). When this flag is set, frequently used pages (such as pages containing kernel procedures or common data tables) can be marked global by setting the global flag in a page-directory or page-table entry.

On a task switch or a write to control register CR3 (which normally causes the TLBs to be flushed), the entries in the TLB marked global are not flushed. Marking pages global in this manner prevents unnecessary reloading of the TLB due to TLB misses on frequently used pages. See Section 4.10, "Caching Translation Information" for a detailed description of this mechanism.

#### 22.22.1.3 Larger Page Sizes

The P6 family processors support large page sizes. For 32-bit paging, this facility is enabled with the PSE (page size extension) flag in control register CR4, bit 4. When this flag is set, the processor supports either 4-KByte or 4-MByte page sizes. PAE paging and 4-level paging<sup>1</sup> support 2-MByte pages regardless of the value of CR4.PSE (see Section 4.4, "PAE Paging" and Section 4.5, "4-Level Paging"). See Chapter 4, "Paging," for more information about large page sizes.

### 22.22.2 CD and NW Cache Control Flags

The CD and NW flags in control register CR0 were introduced in the Intel486 processor. In the P6 family and Pentium processors, these flags are used to implement a writeback strategy for the data cache; in the Intel486 processor, they implement a write-through strategy. See Table 11-5 for a comparison of these bits on the P6 family, Pentium, and Intel486 processors. For complete information on caching, see Chapter 11, "Memory Cache Control."

### 22.22.3 Descriptor Types and Contents

Operating-system code that manages space in descriptor tables often contains an invalid value in the access-rights field of descriptor-table entries to identify unused entries. Access rights values of 80H and 00H remain invalid for the P6 family, Pentium, Intel486, Intel386, and Intel 286 processors. Other values that were invalid on the Intel 286 processor may be valid on the 32-bit processors because uses for these bits have been defined.

---

1. Earlier versions of this manual used the term "IA-32e paging" to identify 4-level paging.



## 22.22.4 Changes in Segment Descriptor Loads

On the Intel386 processor, loading a segment descriptor always causes a locked read and write to set the accessed bit of the descriptor. On the P6 family, Pentium, and Intel486 processors, the locked read and write occur only if the bit is not already set.

## 22.23 DEBUG FACILITIES

The P6 family and Pentium processors include extensions to the Intel486 processor debugging support for breakpoints. To use the new breakpoint features, it is necessary to set the DE flag in control register CR4.

### 22.23.1 Differences in Debug Register DR6

It is not possible to write a 1 to reserved bit 12 in debug status register DR6 on the P6 family and Pentium processors; however, it is possible to write a 1 in this bit on the Intel486 processor. See Table 9-1 for the different setting of this register following a power-up or hardware reset.

### 22.23.2 Differences in Debug Register DR7

The P6 family and Pentium processors determines the type of breakpoint access by the R/W0 through R/W3 fields in debug control register DR7 as follows:

- 00 Break on instruction execution only.
- 01 Break on data writes only.
- 10 Undefined if the DE flag in control register CR4 is cleared; break on I/O reads or writes but not instruction fetches if the DE flag in control register CR4 is set.
- 11 Break on data reads or writes but not instruction fetches.

On the P6 family and Pentium processors, reserved bits 11, 12, 14 and 15 are hard-wired to 0. On the Intel486 processor, however, bit 12 can be set. See Table 9-1 for the different settings of this register following a power-up or hardware reset.

### 22.23.3 Debug Registers DR4 and DR5

Although the DR4 and DR5 registers are documented as reserved, previous generations of processors aliased references to these registers to debug registers DR6 and DR7, respectively. When debug extensions are not enabled (the DE flag in control register CR4 is cleared), the P6 family and Pentium processors remain compatible with existing software by allowing these aliased references. When debug extensions are enabled (the DE flag is set), attempts to reference registers DR4 or DR5 will result in an invalid-opcode exception (#UD).

## 22.24 RECOGNITION OF BREAKPOINTS

For the Pentium processor, it is recommended that debuggers execute the LGDT instruction before returning to the program being debugged to ensure that breakpoints are detected. This operation does not need to be performed on the P6 family, Intel486, or Intel386 processors.

The implementation of test registers on the Intel486 processor used for testing the cache and TLB has been redesigned using MSRs on the P6 family and Pentium processors. (Note that MSRs used for this function are different on the P6 family and Pentium processors.) The MOV to and from test register instructions generate invalid-opcode exceptions (#UD) on the P6 family processors.



## 22.25 EXCEPTIONS AND/OR EXCEPTION CONDITIONS

This section describes the new exceptions and exception conditions added to the 32-bit IA-32 processors and implementation differences in existing exception handling. See Chapter 6, "Interrupt and Exception Handling," for a detailed description of the IA-32 exceptions.

The Pentium III processor introduced new state with the XMM registers. Computations involving data in these registers can produce exceptions. A new MXCSR control/status register is used to determine which exception or exceptions have occurred. When an exception associated with the XMM registers occurs, an interrupt is generated.

- SIMD floating-point exception (#XM, interrupt 19) — New exceptions associated with the SIMD floating-point registers and resulting computations.

No new exceptions were added with the Pentium Pro and Pentium II processors. The set of available exceptions is the same as for the Pentium processor. However, the following exception condition was added to the IA-32 with the Pentium Pro processor:

- Machine-check exception (#MC, interrupt 18) — New exception conditions. Many exception conditions have been added to the machine-check exception and a new architecture has been added for handling and reporting on hardware errors. See Chapter 15, "Machine-Check Architecture," for a detailed description of the new conditions.

The following exceptions and/or exception conditions were added to the IA-32 with the Pentium processor:

- Machine-check exception (#MC, interrupt 18) — New exception. This exception reports parity and other hardware errors. It is a model-specific exception and may not be implemented or implemented differently in future processors. The MCE flag in control register CR4 enables the machine-check exception. When this bit is clear (which it is at reset), the processor inhibits generation of the machine-check exception.
- General-protection exception (#GP, interrupt 13) — New exception condition added. An attempt to write a 1 to a reserved bit position of a special register causes a general-protection exception to be generated.
- Page-fault exception (#PF, interrupt 14) — New exception condition added. When a 1 is detected in any of the reserved bit positions of a page-table entry, page-directory entry, or page-directory pointer during address translation, a page-fault exception is generated.

The following exception was added to the Intel486 processor:

- Alignment-check exception (#AC, interrupt 17) — New exception. Reports unaligned memory references when alignment checking is being performed.

The following exceptions and/or exception conditions were added to the Intel386 processor:

- Divide-error exception (#DE, interrupt 0)
  - Change in exception handling. Divide-error exceptions on the Intel386 processors always leave the saved CS:IP value pointing to the instruction that failed. On the 8086 processor, the CS:IP value points to the next instruction.
  - Change in exception handling. The Intel386 processors can generate the largest negative number as a quotient for the IDIV instruction (80H and 8000H). The 8086 processor generates a divide-error exception instead.
- Invalid-opcode exception (#UD, interrupt 6) — New exception condition added. Improper use of the LOCK instruction prefix can generate an invalid-opcode exception.
- Page-fault exception (#PF, interrupt 14) — New exception condition added. If paging is enabled in a 16-bit program, a page-fault exception can be generated as follows. Paging can be used in a system with 16-bit tasks if all tasks use the same page directory. Because there is no place in a 16-bit TSS to store the PDBR register, switching to a 16-bit task does not change the value of the PDBR register. Tasks ported from the Intel 286 processor should be given 32-bit TSSs so they can make full use of paging.
- General-protection exception (#GP, interrupt 13) — New exception condition added. The Intel386 processor sets a limit of 15 bytes on instruction length. The only way to violate this limit is by putting redundant prefixes before an instruction. A general-protection exception is generated if the limit on instruction length is violated. The 8086 processor has no instruction length limit.

## 22.25.1 Machine-Check Architecture

The Pentium Pro processor introduced a new architecture to the IA-32 for handling and reporting on machine-check exceptions. This machine-check architecture (described in detail in Chapter 15, “Machine-Check Architecture”) greatly expands the ability of the processor to report on internal hardware errors.

## 22.25.2 Priority of Exceptions

The priority of exceptions are broken down into several major categories:

1. Traps on the previous instruction
2. External interrupts
3. Faults on fetching the next instruction
4. Faults in decoding the next instruction
5. Faults on executing an instruction

There are no changes in the priority of these major categories between the different processors, however, exceptions within these categories are implementation dependent and may change from processor to processor.

## 22.25.3 Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers

MMX instructions and a subset of SSE, SSE2, SSSE3 instructions operate on MMX registers. The exception conditions of these instructions are described in the following tables.

**Table 22-4. Exception Conditions for Legacy SIMD/MMX Instructions with FP Exception and 16-Byte Alignment**

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
	X	X	X	X	If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.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
			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
#PF(fault-code)		X	X	X	For a page fault
#XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1
Applicable Instructions	CVTPD2PI, CVTTPD2PI				

Table 22-5. Exception Conditions for Legacy SIMD/MMX Instructions with XMM and FP Exception

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
	X	X	X	X	If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (FOH)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.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		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
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1
Applicable Instructions	CVTPI2PS, CVTPS2PI, CVTTPS2PI				

**Table 22-6. Exception Conditions for Legacy SIMD/MMX Instructions with XMM and without FP Exception**

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CRO.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF <sup>1</sup>	X	X	X	X	If there is a pending X87 FPU exception
#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		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
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	CVTPI2PD				

**NOTES:**

1. Applies to "CVTPI2PD xmm, mm" but not "CVTPI2PD xmm, m64".

Table 22-7. Exception Conditions for SIMD/MMX Instructions with Memory Reference

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1.
	X	X	X	X	If preceded by a LOCK prefix (FOH)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.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		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
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	PABSB, PABSD, PABSW, PACKSSWB, PACKSSDW, PACKUSWB, PADDB, PADD, PADDQ, PADDW, PADDSB, PADDSD, PADDUSB, PADDUSW, PALIGNR, PAND, PANDN, PAVGB, PAVGW, PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW, PHADD, PHADDW, PHADDSW, PHSUBD, PHSUBW, PHSUBSW, PINSRW, PMADDUSWB, PMADDWD, PMAXS, PMAXUB, PMINSW, PMINUB, PMULHRW, PMULHUW, PMULHW, PMULLW, PMULUDQ, PSADB, PSHUFB, PSHUFW, PSIGNB, PSIGND, PSIGNW, PSLLW, PSLLD, PSLLQ, PSRAD, PSRAW, PSRLW, PSRLD, PSRLQ, PSUBB, PSUBD, PSUBQ, PSUBW, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ, PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ, PXOR				

**Table 22-8. Exception Conditions for Legacy SIMD/MMX Instructions without FP Exception**

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CRO.EM[bit 2] = 1. If ModR/M.mod ≠ 11b <sup>1</sup>
	X	X	X	X	If preceded by a LOCK prefix (FOH)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#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
#GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If the destination operand is in a non-writable segment. <sup>2</sup> If the DS, ES, FS, or GS register contains a NULL segment selector. <sup>3</sup>
				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
#PF(fault-code)		X	X	X	For a page fault
#AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	MASKMOVQ, MOVNTQ, "MOVQ (mmreg)"				

**NOTES:**

- 1. Applies to MASKMOVQ only.
- 2. Applies to MASKMOVQ and MOVQ (mmreg) only.
- 3. Applies to MASKMOVQ only.

Table 22-9. Exception Conditions for Legacy SIMD/MMX Instructions without Memory Reference

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1.
	X	X	X	X	If preceded by a LOCK prefix (FOH)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM			X	X	If CR0.TS[bit 3]=1
Applicable Instructions	PEXTRW, PMOVMASKB				

## 22.26 INTERRUPTS

The following differences in handling interrupts are found among the IA-32 processors.

### 22.26.1 Interrupt Propagation Delay

External hardware interrupts may be recognized on different instruction boundaries on the P6 family, Pentium, Intel486, and Intel386 processors, due to the superscaler designs of the P6 family and Pentium processors. Therefore, the EIP pushed onto the stack when servicing an interrupt may be different for the P6 family, Pentium, Intel486, and Intel386 processors.

### 22.26.2 NMI Interrupts

After an NMI interrupt is recognized by the P6 family, Pentium, Intel486, Intel386, and Intel 286 processors, the NMI interrupt is masked until the first IRET instruction is executed, unlike the 8086 processor.

### 22.26.3 IDT Limit

The LIDT instruction can be used to set a limit on the size of the IDT. A double-fault exception (#DF) is generated if an interrupt or exception attempts to read a vector beyond the limit. Shutdown then occurs on the 32-bit IA-32 processors if the double-fault handler vector is beyond the limit. (The 8086 processor does not have a shutdown mode nor a limit.)

## 22.27 ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER (APIC)

The Advanced Programmable Interrupt Controller (APIC), referred to in this book as the **local APIC**, was introduced into the IA-32 processors with the Pentium processor (beginning with the 735/90 and 815/100 models) and is included in the Pentium 4, Intel Xeon, and P6 family processors. The features and functions of the local APIC are derived from the Intel 82489DX external APIC, which was used with the Intel486 and early Pentium processors. Additional refinements of the local APIC architecture were incorporated in the Pentium 4 and Intel Xeon processors.

### 22.27.1 Software Visible Differences Between the Local APIC and the 82489DX

The following features in the local APIC features differ from those found in the 82489DX external APIC:

- When the local APIC is disabled by clearing the APIC software enable/disable flag in the spurious-interrupt vector MSR, the state of its internal registers are unaffected, except that the mask bits in the LVT are all set to block local interrupts to the processor. Also, the local APIC ceases accepting IPIs except for INIT, SMI, NMI, and start-up IPIs. In the 82489DX, when the local unit is disabled, all the internal registers including the IRR, ISR and TMR are cleared and the mask bits in the LVT are set. In this state, the 82489DX local unit will accept only the reset deassert message.
- In the local APIC, NMI and INIT (except for INIT deassert) are always treated as edge triggered interrupts, even if programmed otherwise. In the 82489DX, these interrupts are always level triggered.
- In the local APIC, IPIs generated through the ICR are always treated as edge triggered (except INIT Deassert). In the 82489DX, the ICR can be used to generate either edge or level triggered IPIs.
- In the local APIC, the logical destination register supports 8 bits; in the 82489DX, it supports 32 bits.
- In the local APIC, the APIC ID register is 4 bits wide; in the 82489DX, it is 8 bits wide.
- The remote read delivery mode provided in the 82489DX and local APIC for Pentium processors is not supported in the local APIC in the Pentium 4, Intel Xeon, and P6 family processors.
- For the 82489DX, in the lowest priority delivery mode, all the target local APICs specified by the destination field participate in the lowest priority arbitration. For the local APIC, only those local APICs which have free interrupt slots will participate in the lowest priority arbitration.

### 22.27.2 New Features Incorporated in the Local APIC for the P6 Family and Pentium Processors

The local APIC in the Pentium and P6 family processors have the following new features not found in the 82489DX external APIC.

- Cluster addressing is supported in logical destination mode.
- Focus processor checking can be enabled/disabled.
- Interrupt input signal polarity can be programmed for the LINT0 and LINT1 pins.
- An SMI IPI is supported through the ICR and I/O redirection table.
- An error status register is incorporated into the LVT to log and report APIC errors.

In the P6 family processors, the local APIC incorporates an additional LVT register to handle performance monitoring counter interrupts.

### 22.27.3 New Features Incorporated in the Local APIC of the Pentium 4 and Intel Xeon Processors

The local APIC in the Pentium 4 and Intel Xeon processors has the following new features not found in the P6 family and Pentium processors and in the 82489DX.

- The local APIC ID is extended to 8 bits.
- An thermal sensor register is incorporated into the LVT to handle thermal sensor interrupts.
- The the ability to deliver lowest-priority interrupts to a focus processor is no longer supported.
- The flat cluster logical destination mode is not supported.

## 22.28 TASK SWITCHING AND TSS

This section identifies the implementation differences of task switching, additions to the TSS and the handling of TSSs and TSS segment selectors.



## 22.28.1 P6 Family and Pentium Processor TSS

When the virtual mode extensions are enabled (by setting the VME flag in control register CR4), the TSS in the P6 family and Pentium processors contain an interrupt redirection bit map, which is used in virtual-8086 mode to redirect interrupts back to an 8086 program.

## 22.28.2 TSS Selector Writes

During task state saves, the Intel486 processor writes 2-byte segment selectors into a 32-bit TSS, leaving the upper 16 bits undefined. For performance reasons, the P6 family and Pentium processors write 4-byte segment selectors into the TSS, with the upper 2 bytes being 0. For compatibility reasons, code should not depend on the value of the upper 16 bits of the selector in the TSS.

## 22.28.3 Order of Reads/Writes to the TSS

The order of reads and writes into the TSS is processor dependent. The P6 family and Pentium processors may generate different page-fault addresses in control register CR2 in the same TSS area than the Intel486 and Intel386 processors, if a TSS crosses a page boundary (which is not recommended).

## 22.28.4 Using A 16-Bit TSS with 32-Bit Constructs

Task switches using 16-bit TSSs should be used only for pure 16-bit code. Any new code written using 32-bit constructs (operands, addressing, or the upper word of the EFLAGS register) should use only 32-bit TSSs. This is due to the fact that the 32-bit processors do not save the upper 16 bits of EFLAGS to a 16-bit TSS. A task switch back to a 16-bit task that was executing in virtual mode will never re-enable the virtual mode, as this flag was not saved in the upper half of the EFLAGS value in the TSS. Therefore, it is strongly recommended that any code using 32-bit constructs use a 32-bit TSS to ensure correct behavior in a multitasking environment.

## 22.28.5 Differences in I/O Map Base Addresses

The Intel486 processor considers the TSS segment to be a 16-bit segment and wraps around the 64K boundary. Any I/O accesses check for permission to access this I/O address at the I/O base address plus the I/O offset. If the I/O map base address exceeds the specified limit of 0DFFFH, an I/O access will wrap around and obtain the permission for the I/O address at an incorrect location within the TSS. A TSS limit violation does not occur in this situation on the Intel486 processor. However, the P6 family and Pentium processors consider the TSS to be a 32-bit segment and a limit violation occurs when the I/O base address plus the I/O offset is greater than the TSS limit. By following the recommended specification for the I/O base address to be less than 0DFFFH, the Intel486 processor will not wrap around and access incorrect locations within the TSS for I/O port validation and the P6 family and Pentium processors will not experience general-protection exceptions (#GP). Figure 22-1 demonstrates the different areas accessed by the Intel486 and the P6 family and Pentium processors.

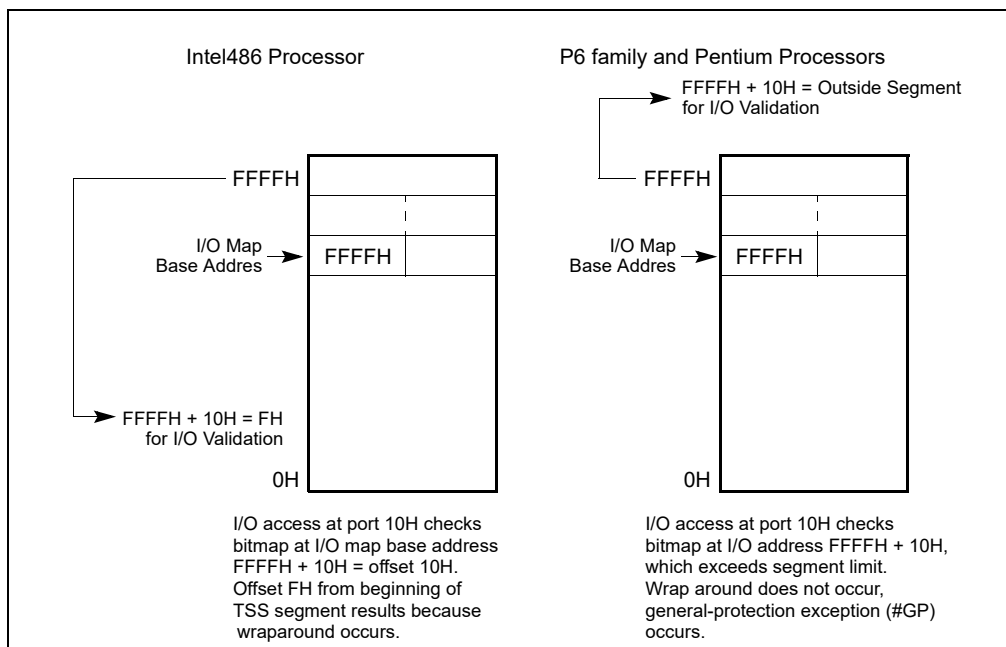


Figure 22-1. I/O Map Base Address Differences

## 22.29 CACHE MANAGEMENT

The P6 family processors include two levels of internal caches: L1 (level 1) and L2 (level 2). The L1 cache is divided into an instruction cache and a data cache; the L2 cache is a general-purpose cache. See Section 11.1, "Internal Caches, TLBs, and Buffers," for a description of these caches. (Note that although the Pentium II processor L2 cache is physically located on a separate chip in the cassette, it is considered an internal cache.)

The Pentium processor includes separate level 1 instruction and data caches. The data cache supports a writeback (or alternatively write-through, on a line by line basis) policy for memory updates.

The Intel486 processor includes a single level 1 cache for both instructions and data.

The meaning of the CD and NW flags in control register CR0 have been redefined for the P6 family and Pentium processors. For these processors, the recommended value (00B) enables writeback for the data cache of the Pentium processor and for the L1 data cache and L2 cache of the P6 family processors. In the Intel486 processor, setting these flags to (00B) enables write-through for the cache.

External system hardware can force the Pentium processor to disable caching or to use the write-through cache policy should that be required. In the P6 family processors, the MTRRs can be used to override the CD and NW flags (see Table 11-6).

The P6 family and Pentium processors support page-level cache management in the same manner as the Intel486 processor by using the PCD and PWT flags in control register CR3, the page-directory entries, and the page-table entries. The Intel486 processor, however, is not affected by the state of the PWT flag since the internal cache of the Intel486 processor is a write-through cache.

### 22.29.1 Self-Modifying Code with Cache Enabled

On the Intel486 processor, a write to an instruction in the cache will modify it in both the cache and memory. If the instruction was prefetched before the write, however, the old version of the instruction could be the one executed. To prevent this problem, it is necessary to flush the instruction prefetch unit of the Intel486 processor by coding a jump instruction immediately after any write that modifies an instruction. The P6 family and Pentium processors, however, check whether a write may modify an instruction that has been prefetched for execution. This check is based on the linear address of the instruction. If the linear address of an instruction is found to be present in the

prefetch queue, the P6 family and Pentium processors flush the prefetch queue, eliminating the need to code a jump instruction after any writes that modify an instruction.

Because the linear address of the write is checked against the linear address of the instructions that have been prefetched, special care must be taken for self-modifying code to work correctly when the physical addresses of the instruction and the written data are the same, but the linear addresses differ. In such cases, it is necessary to execute a serializing operation to flush the prefetch queue after the write and before executing the modified instruction. See Section 8.3, “Serializing Instructions,” for more information on serializing instructions.

### NOTE

The check on linear addresses described above is not in practice a concern for compatibility. Applications that include self-modifying code use the same linear address for modifying and fetching the instruction. System software, such as a debugger, that might possibly modify an instruction using a different linear address than that used to fetch the instruction must execute a serializing operation, such as IRET, before the modified instruction is executed.

## 22.29.2 Disabling the L3 Cache

A unified third-level (L3) cache in processors based on Intel NetBurst microarchitecture (see Section 11.1, “Internal Caches, TLBs, and Buffers”) provides the third-level cache disable flag, bit 6 of the IA32\_MISC\_ENABLE MSR. The third-level cache disable flag allows the L3 cache to be disabled and enabled, independently of the L1 and L2 caches (see Section 11.5.4, “Disabling and Enabling the L3 Cache”). The third-level cache disable flag applies only to processors based on Intel NetBurst microarchitecture. Processors with L3 and based on other microarchitectures do not support the third-level cache disable flag.

## 22.30 PAGING

This section identifies enhancements made to the paging mechanism and implementation differences in the paging mechanism for various IA-32 processors.

### 22.30.1 Large Pages

The Pentium processor extended the memory management/paging facilities of the IA-32 to allow large (4 MBytes) pages sizes (see Section 4.3, “32-Bit Paging”). The first P6 family processor (the Pentium Pro processor) added a 2 MByte page size to the IA-32 in conjunction with the physical address extension (PAE) feature (see Section 4.4, “PAE Paging”).

The availability of large pages with 32-bit paging on any IA-32 processor can be determined via feature bit 3 (PSE) of register EDX after the CPUID instruction has been execution with an argument of 1. (Large pages are always available with PAE paging and 4-level paging.) Intel processors that do not support the CPUID instruction support only 32-bit paging and do not support page size enhancements. (See “CPUID—CPU Identification” in Chapter 3, “Instruction Set Reference, A-L,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A* for more information on the CPUID instruction.)

### 22.30.2 PCD and PWT Flags

The PCD and PWT flags were introduced to the IA-32 in the Intel486 processor to control the caching of pages:

- PCD (page-level cache disable) flag—Controls caching on a page-by-page basis.
- PWT (page-level write-through) flag—Controls the write-through/writeback caching policy on a page-by-page basis. Since the internal cache of the Intel486 processor is a write-through cache, it is not affected by the state of the PWT flag.

### 22.30.3 Enabling and Disabling Paging

Paging is enabled and disabled by loading a value into control register CR0 that modifies the PG flag. For backward and forward compatibility with all IA-32 processors, Intel recommends that the following operations be performed when enabling or disabling paging:

1. Execute a MOV CR0, REG instruction to either set (enable paging) or clear (disable paging) the PG flag.
2. Execute a near JMP instruction.

The sequence bounded by the MOV and JMP instructions should be identity mapped (that is, the instructions should reside on a page whose linear and physical addresses are identical).

For the P6 family processors, the MOV CR0, REG instruction is serializing, so the jump operation is not required. However, for backwards compatibility, the JMP instruction should still be included.

## 22.31 STACK OPERATIONS AND SUPERVISOR SOFTWARE

This section identifies the differences in the stack mechanism for the various IA-32 processors.

### 22.31.1 Selector Pushes and Pops

When pushing a segment selector onto the stack, the Pentium 4, Intel Xeon, P6 family, and Intel486 processors decrement the ESP register by the operand size and then write 2 bytes. If the operand size is 32-bits, the upper two bytes of the write are not modified. The Pentium processor decrements the ESP register by the operand size and determines the size of the write by the operand size. If the operand size is 32-bits, the upper two bytes are written as 0s.

When popping a segment selector from the stack, the Pentium 4, Intel Xeon, P6 family, and Intel486 processors read 2 bytes and increment the ESP register by the operand size of the instruction. The Pentium processor determines the size of the read from the operand size and increments the ESP register by the operand size.

It is possible to align a 32-bit selector push or pop such that the operation generates an exception on a Pentium processor and not on an Pentium 4, Intel Xeon, P6 family, or Intel486 processor. This could occur if the third and/or fourth byte of the operation lies beyond the limit of the segment or if the third and/or fourth byte of the operation is locate on a non-present or inaccessible page.

For a POP-to-memory instruction that meets the following conditions:

- The stack segment size is 16-bit.
- Any 32-bit addressing form with the SIB byte specifying ESP as the base register.
- The initial stack pointer is FFFCH (32-bit operand) or FFFEh (16-bit operand) and will wrap around to 0H as a result of the POP operation.

The result of the memory write is implementation-specific. For example, in P6 family processors, the result of the memory write is SS:0H plus any scaled index and displacement. In Pentium processors, the result of the memory write may be either a stack fault (real mode or protected mode with stack segment size of 64 KByte), or write to SS:10000H plus any scaled index and displacement (protected mode and stack segment size exceeds 64 KByte).

### 22.31.2 Error Code Pushes

The Intel486 processor implements the error code pushed on the stack as a 16-bit value. When pushed onto a 32-bit stack, the Intel486 processor only pushes 2 bytes and updates ESP by 4. The P6 family and Pentium processors' error code is a full 32 bits with the upper 16 bits set to zero. The P6 family and Pentium processors, therefore, push 4 bytes and update ESP by 4. Any code that relies on the state of the upper 16 bits may produce inconsistent results.

### 22.31.3 Fault Handling Effects on the Stack

During the handling of certain instructions, such as CALL and PUSH, faults may occur in different sequences for the different processors. For example, during far calls, the Intel486 processor pushes the old CS and EIP before a possible branch fault is resolved. A branch fault is a fault from a branch instruction occurring from a segment limit or access rights violation. If a branch fault is taken, the Intel486 and P6 family processors will have corrupted memory below the stack pointer. However, the ESP register is backed up to make the instruction restartable. The P6 family processors issue the branch before the pushes. Therefore, if a branch fault does occur, these processors do not corrupt memory below the stack pointer. This implementation difference, however, does not constitute a compatibility problem, as only values at or above the stack pointer are considered to be valid. Other operations that encounter faults may also corrupt memory below the stack pointer and this behavior may vary on different implementations.

### 22.31.4 Interlevel RET/IRET From a 16-Bit Interrupt or Call Gate

If a call or interrupt is made from a 32-bit stack environment through a 16-bit gate, only 16 bits of the old ESP can be pushed onto the stack. On the subsequent RET/IRET, the 16-bit ESP is popped but the full 32-bit ESP is updated since control is being resumed in a 32-bit stack environment. The Intel486 processor writes the SS selector into the upper 16 bits of ESP. The P6 family and Pentium processors write zeros into the upper 16 bits.

## 22.32 MIXING 16- AND 32-BIT SEGMENTS

The features of the 16-bit Intel 286 processor are an object-code compatible subset of those of the 32-bit IA-32 processors. The D (default operation size) flag in segment descriptors indicates whether the processor treats a code or data segment as a 16-bit or 32-bit segment; the B (default stack size) flag in segment descriptors indicates whether the processor treats a stack segment as a 16-bit or 32-bit segment.

The segment descriptors used by the Intel 286 processor are supported by the 32-bit IA-32 processors if the Intel-reserved word (highest word) of the descriptor is clear. On the 32-bit IA-32 processors, this word includes the upper bits of the base address and the segment limit.

The segment descriptors for data segments, code segments, local descriptor tables (there are no descriptors for global descriptor tables), and task gates are the same for the 16- and 32-bit processors. Other 16-bit descriptors (TSS segment, call gate, interrupt gate, and trap gate) are supported by the 32-bit processors.

The 32-bit processors also have descriptors for TSS segments, call gates, interrupt gates, and trap gates that support the 32-bit architecture. Both kinds of descriptors can be used in the same system.

For those segment descriptors common to both 16- and 32-bit processors, clear bits in the reserved word cause the 32-bit processors to interpret these descriptors exactly as an Intel 286 processor does, that is:

- Base Address — The upper 8 bits of the 32-bit base address are clear, which limits base addresses to 24 bits.
- Limit — The upper 4 bits of the limit field are clear, restricting the value of the limit field to 64 KBytes.
- Granularity bit — The G (granularity) flag is clear, indicating the value of the 16-bit limit is interpreted in units of 1 byte.
- Big bit — In a data-segment descriptor, the B flag is clear in the segment descriptor used by the 32-bit processors, indicating the segment is no larger than 64 KBytes.
- Default bit — In a code-segment descriptor, the D flag is clear, indicating 16-bit addressing and operands are the default. In a stack-segment descriptor, the D flag is clear, indicating use of the SP register (instead of the ESP register) and a 64-KByte maximum segment limit.

For information on mixing 16- and 32-bit code in applications, see Chapter 21, "Mixing 16-Bit and 32-Bit Code."

## 22.33 SEGMENT AND ADDRESS WRAPAROUND

This section discusses differences in segment and address wraparound between the P6 family, Pentium, Intel486, Intel386, Intel 286, and 8086 processors.

### 22.33.1 Segment Wraparound

On the 8086 processor, an attempt to access a memory operand that crosses offset 65,535 or 0FFFFH or offset 0 (for example, moving a word to offset 65,535 or pushing a word when the stack pointer is set to 1) causes the offset to wrap around modulo 65,536 or 010000H. With the Intel 286 processor, any base and offset combination that addresses beyond 16 MBytes wraps around to the 1 MByte of the address space. The P6 family, Pentium, Intel486, and Intel386 processors in real-address mode generate an exception in these cases:

- A general-protection exception (#GP) if the segment is a data segment (that is, if the CS, DS, ES, FS, or GS register is being used to address the segment).
- A stack-fault exception (#SS) if the segment is a stack segment (that is, if the SS register is being used).

An exception to this behavior occurs when a stack access is data aligned, and the stack pointer is pointing to the last aligned piece of data that size at the top of the stack (ESP is FFFFFFFCH). When this data is popped, no segment limit violation occurs and the stack pointer will wrap around to 0.

The address space of the P6 family, Pentium, and Intel486 processors may wraparound at 1 MByte in real-address mode. An external A20M# pin forces wraparound if enabled. On Intel 8086 processors, it is possible to specify addresses greater than 1 MByte. For example, with a selector value FFFFH and an offset of FFFFH, the effective address would be 10FFE7H (1 MByte plus 65519 bytes). The 8086 processor, which can form addresses up to 20 bits long, truncates the uppermost bit, which “wraps” this address to FFE7H. However, the P6 family, Pentium, and Intel486 processors do not truncate this bit if A20M# is not enabled.

If a stack operation wraps around the address limit, shutdown occurs. (The 8086 processor does not have a shutdown mode or a limit.)

The behavior when executing near the limit of a 4-GByte selector (limit = FFFFFFFFH) is different between the Pentium Pro and the Pentium 4 family of processors. On the Pentium Pro, instructions which cross the limit -- for example, a two byte instruction such as INC EAX that is encoded as FFH C0H starting exactly at the limit faults for a segment violation (a one byte instruction at FFFFFFFFH does not cause an exception). Using the Pentium 4 micro-processor family, neither of these situations causes a fault.

Segment wraparound and the functionality of A20M# is used primarily by older operating systems and not used by modern operating systems. On newer Intel 64 processors, A20M# may be absent.

## 22.34 STORE BUFFERS AND MEMORY ORDERING

The Pentium 4, Intel Xeon, and P6 family processors provide a store buffer for temporary storage of writes (stores) to memory (see Section 11.10, “Store Buffer”). Writes stored in the store buffer(s) are always written to memory in program order, with the exception of “fast string” store operations (see Section 8.2.4, “Fast-String Operation and Out-of-Order Stores”).

The Pentium processor has two store buffers, one corresponding to each of the pipelines. Writes in these buffers are always written to memory in the order they were generated by the processor core.

It should be noted that only memory writes are buffered and I/O writes are not. The Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors do not synchronize the completion of memory writes on the bus and instruction execution after a write. An I/O, locked, or serializing instruction needs to be executed to synchronize writes with the next instruction (see Section 8.3, “Serializing Instructions”).

The Pentium 4, Intel Xeon, and P6 family processors use processor ordering to maintain consistency in the order that data is read (loaded) and written (stored) in a program and the order the processor actually carries out the reads and writes. With this type of ordering, reads can be carried out speculatively and in any order, reads can pass buffered writes, and writes to memory are always carried out in program order. (See Section 8.2, “Memory Ordering,” for more information about processor ordering.) The Pentium III processor introduced a new instruction to serialize writes and make them globally visible. Memory ordering issues can arise between a producer and a consumer of data. The SFENCE instruction provides a performance-efficient way of ensuring ordering between routines that produce weakly-ordered results and routines that consume this data.

No re-ordering of reads occurs on the Pentium processor, except under the condition noted in Section 8.2.1, “Memory Ordering in the Intel® Pentium® and Intel486™ Processors,” and in the following paragraph describing the Intel486 processor.

Specifically, the store buffers are flushed before the IN instruction is executed. No reads (as a result of cache miss) are reordered around previously generated writes sitting in the store buffers. The implication of this is that the store buffers will be flushed or emptied before a subsequent bus cycle is run on the external bus.

On both the Intel486 and Pentium processors, under certain conditions, a memory read will go onto the external bus before the pending memory writes in the buffer even though the writes occurred earlier in the program execution. A memory read will only be reordered in front of all writes pending in the buffers if all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions, the Intel486 and Pentium processors will not read from an external memory location that needs to be updated by one of the pending writes.

During a locked bus cycle, the Intel486 processor will always access external memory, it will never look for the location in the on-chip cache. All data pending in the Intel486 processor's store buffers will be written to memory before a locked cycle is allowed to proceed to the external bus. Thus, the locked bus cycle can be used for eliminating the possibility of reordering read cycles on the Intel486 processor. The Pentium processor does check its cache on a read-modify-write access and, if the cache line has been modified, writes the contents back to memory before locking the bus. The P6 family processors write to their cache on a read-modify-write operation (if the access does not split across a cache line) and does not write back to system memory. If the access does split across a cache line, it locks the bus and accesses system memory.

I/O reads are never reordered in front of buffered memory writes on an IA-32 processor. This ensures an update of all memory locations before reading the status from an I/O device.

## 22.35 BUS LOCKING

The Intel 286 processor performs the bus locking differently than the Intel P6 family, Pentium, Intel486, and Intel386 processors. Programs that use forms of memory locking specific to the Intel 286 processor may not run properly when run on later processors.

A locked instruction is guaranteed to lock only the area of memory defined by the destination operand, but may lock a larger memory area. For example, typical 8086 and Intel 286 configurations lock the entire physical memory space. Programmers should not depend on this.

On the Intel 286 processor, the LOCK prefix is sensitive to IOPL. If the CPL is greater than the IOPL, a general-protection exception (#GP) is generated. On the Intel386 DX, Intel486, and Pentium, and P6 family processors, no check against IOPL is performed.

The Pentium processor automatically asserts the LOCK# signal when acknowledging external interrupts. After signaling an interrupt request, an external interrupt controller may use the data bus to send the interrupt vector to the processor. After receiving the interrupt request signal, the processor asserts LOCK# to insure that no other data appears on the data bus until the interrupt vector is received. This bus locking does not occur on the P6 family processors.

## 22.36 BUS HOLD

Unlike the 8086 and Intel 286 processors, but like the Intel386 and Intel486 processors, the P6 family and Pentium processors respond to requests for control of the bus from other potential bus masters, such as DMA controllers, between transfers of parts of an unaligned operand, such as two words which form a doubleword. Unlike the Intel386 processor, the P6 family, Pentium and Intel486 processors respond to bus hold during reset initialization.

## 22.37 MODEL-SPECIFIC EXTENSIONS TO THE IA-32

Certain extensions to the IA-32 are specific to a processor or family of IA-32 processors and may not be implemented or implemented in the same way in future processors. The following sections describe these model-specific extensions. The CPUID instruction indicates the availability of some of the model-specific features.



## 22.37.1 Model-Specific Registers

The Pentium processor introduced a set of model-specific registers (MSRs) for use in controlling hardware functions and performance monitoring. To access these MSRs, two new instructions were added to the IA-32 architecture: read MSR (RDMSR) and write MSR (WRMSR). The MSRs in the Pentium processor are not guaranteed to be duplicated or provided in the next generation IA-32 processors.

The P6 family processors greatly increased the number of MSRs available to software. See Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* for a complete list of the available MSRs. The new registers control the debug extensions, the performance counters, the machine-check exception capability, the machine-check architecture, and the MTRRs. These registers are accessible using the RDMSR and WRMSR instructions. Specific information on some of these new MSRs is provided in the following sections. As with the Pentium processor MSR, the P6 family processor MSRs are not guaranteed to be duplicated or provided in the next generation IA-32 processors.

## 22.37.2 RDMSR and WRMSR Instructions

The RDMSR (read model-specific register) and WRMSR (write model-specific register) instructions recognize a much larger number of model-specific registers in the P6 family processors. (See “RDMSR—Read from Model Specific Register” and “WRMSR—Write to Model Specific Register” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C & 2D* for more information.)

## 22.37.3 Memory Type Range Registers

Memory type range registers (MTRRs) are a new feature introduced into the IA-32 in the Pentium Pro processor. MTRRs allow the processor to optimize memory operations for different types of memory, such as RAM, ROM, frame buffer memory, and memory-mapped I/O.

MTRRs are MSRs that contain an internal map of how physical address ranges are mapped to various types of memory. The processor uses this internal memory map to determine the cacheability of various physical memory locations and the optimal method of accessing memory locations. For example, if a memory location is specified in an MTRR as write-through memory, the processor handles accesses to this location as follows. It reads data from that location in lines and caches the read data or maps all writes to that location to the bus and updates the cache to maintain cache coherency. In mapping the physical address space with MTRRs, the processor recognizes five types of memory: uncacheable (UC), uncacheable, speculatable, write-combining (WC), write-through (WT), write-protected (WP), and writeback (WB).

Earlier IA-32 processors (such as the Intel486 and Pentium processors) used the KEN# (cache enable) pin and external logic to maintain an external memory map and signal cacheable accesses to the processor. The MTRR mechanism simplifies hardware designs by eliminating the KEN# pin and the external logic required to drive it.

See Chapter 9, “Processor Management and Initialization,” and Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* for more information on the MTRRs.

## 22.37.4 Machine-Check Exception and Architecture

The Pentium processor introduced a new exception called the machine-check exception (#MC, interrupt 18). This exception is used to detect hardware-related errors, such as a parity error on a read cycle.

The P6 family processors extend the types of errors that can be detected and that generate a machine-check exception. It also provides a new machine-check architecture for recording information about a machine-check error and provides extended recovery capability.

The machine-check architecture provides several banks of reporting registers for recording machine-check errors. Each bank of registers is associated with a specific hardware unit in the processor. The primary focus of the machine checks is on bus and interconnect operations; however, checks are also made of translation lookaside buffer (TLB) and cache operations.



The machine-check architecture can correct some errors automatically and allow for reliable restart of instruction execution. It also collects sufficient information for software to use in correcting other machine errors not corrected by hardware.

See Chapter 15, “Machine-Check Architecture,” for more information on the machine-check exception and the machine-check architecture.

### 22.37.5 Performance-Monitoring Counters

The P6 family and Pentium processors provide two performance-monitoring counters for use in monitoring internal hardware operations. The number of performance monitoring counters and associated programming interfaces may be implementation specific for Pentium 4 processors, Pentium M processors. Later processors may have implemented these as part of an architectural performance monitoring feature. The architectural and non-architectural performance monitoring interfaces for different processor families are described in Chapter 18, “Performance Monitoring,”. Chapter 19, “Performance Monitoring Events,” lists all the events that can be counted for architectural performance monitoring events and non-architectural events. The counters are set up, started, and stopped using two MSRs and the RDMSR and WRMSR instructions. For the P6 family processors, the current count for a particular counter can be read using the new RDPMC instruction.

The performance-monitoring counters are useful for debugging programs, optimizing code, diagnosing system failures, or refining hardware designs. See Chapter 18, “Performance Monitoring,” for more information on these counters.

## 22.38 TWO WAYS TO RUN INTEL 286 PROCESSOR TASKS

When porting 16-bit programs to run on 32-bit IA-32 processors, there are two approaches to consider:

- Porting an entire 16-bit software system to a 32-bit processor, complete with the old operating system, loader, and system builder. Here, all tasks will have 16-bit TSSs. The 32-bit processor is being used as if it were a faster version of the 16-bit processor.
- Porting selected 16-bit applications to run in a 32-bit processor environment with a 32-bit operating system, loader, and system builder. Here, the TSSs used to represent 286 tasks should be changed to 32-bit TSSs. It is possible to mix 16 and 32-bit TSSs, but the benefits are small and the problems are great. All tasks in a 32-bit software system should have 32-bit TSSs. It is not necessary to change the 16-bit object modules themselves; TSSs are usually constructed by the operating system, by the loader, or by the system builder. See Chapter 21, “Mixing 16-Bit and 32-Bit Code,” for more detailed information about mixing 16-bit and 32-bit code.

Because the 32-bit processors use the contents of the reserved word of 16-bit segment descriptors, 16-bit programs that place values in this word may not run correctly on the 32-bit processors.

## 22.39 INITIAL STATE OF PENTIUM, PENTIUM PRO AND PENTIUM 4 PROCESSORS

Table 22-10 shows the state of the flags and other registers following power-up for the Pentium, Pentium Pro and Pentium 4 processors. The state of control register CR0 is 60000010H (see Figure 9-1 “Contents of CR0 Register after Reset” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). This places the processor in real-address mode with paging disabled.

**Table 22-10. Processor State Following Power-up/Reset/INIT for Pentium, Pentium Pro and Pentium 4 Processors**

Register	Pentium 4 Processor	Pentium Pro Processor	Pentium Processor
EFLAGS <sup>1</sup>	00000002H	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H	0000FFF0H
CR0	60000010H <sup>2</sup>	60000010H <sup>2</sup>	60000010H <sup>2</sup>
CR2, CR3, CR4	00000000H	00000000H	00000000H

**Table 22-10. Processor State Following Power-up/Reset/INIT for Pentium, Pentium Pro and Pentium 4 Processors**

Register	Pentium 4 Processor	Pentium Pro Processor	Pentium Processor
CS	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed
SS, DS, ES, FS, GS	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed
EDX	00000FxxH	000n06xxH <sup>3</sup>	000005xxH
EAX	0 <sup>4</sup>	0 <sup>4</sup>	0 <sup>4</sup>
EBX, ECX, ESI, EDI, EBP, ESP	00000000H	00000000H	00000000H
ST0 through ST7 <sup>5</sup>	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged
x87 FPU Control Word <sup>5</sup>	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH
x87 FPU Status Word <sup>5</sup>	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H
x87 FPU Tag Word <sup>5</sup>	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH
x87 FPU Data Operand and CS Seg. Selectors <sup>5</sup>	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H
x87 FPU Data Operand and Inst. Pointers <sup>5</sup>	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H
MM0 through MM7 <sup>5</sup>	Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged	Pentium II and Pentium III Processors Only— Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged	Pentium with MMX Technology Only— Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged
XMM0 through XMM7	Pwr up or Reset: 0H INIT: Unchanged	If CPUID.01H:SSE is 1 — Pwr up or Reset: 0H INIT: Unchanged	NA
MXCSR	Pwr up or Reset: 1F80H INIT: Unchanged	Pentium III processor only- Pwr up or Reset: 1F80H INIT: Unchanged	NA
GDTR, IDTR	Base = 00000000H Limit = FFFFH AR = Present, R/W	Base = 00000000H Limit = FFFFH AR = Present, R/W	Base = 00000000H Limit = FFFFH AR = Present, R/W
LDTR, Task Register	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W
DR0, DR1, DR2, DR3	00000000H	00000000H	00000000H
DR6	FFFF0FF0H	FFFF0FF0H	FFFF0FF0H

**Table 22-10. Processor State Following Power-up/Reset/INIT for Pentium, Pentium Pro and Pentium 4 Processors**

Register	Pentium 4 Processor	Pentium Pro Processor	Pentium Processor
DR7	00000400H	00000400H	00000400H
Time-Stamp Counter	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged
Perf. Counters and Event Select	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged
All Other MSRs	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged
Data and Code Cache, TLBs	Invalid <sup>6</sup>	Invalid <sup>6</sup>	Invalid <sup>6</sup>
Fixed MTRRs	Pwr up or Reset: Disabled INIT: Unchanged	Pwr up or Reset: Disabled INIT: Unchanged	Not Implemented
Variable MTRRs	Pwr up or Reset: Disabled INIT: Unchanged	Pwr up or Reset: Disabled INIT: Unchanged	Not Implemented
Machine-Check Architecture	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged	Not Implemented
APIC	Pwr up or Reset: Enabled INIT: Unchanged	Pwr up or Reset: Enabled INIT: Unchanged	Pwr up or Reset: Enabled INIT: Unchanged
R8-R15 <sup>7</sup>	0000000000000000H	0000000000000000H	N.A.
XMM8-XMM15 <sup>7</sup>	Pwr up or Reset: 0H INIT: Unchanged	Pwr up or Reset: 0H INIT: Unchanged	N.A.

**NOTES:**

1. The 10 most-significant bits of the EFLAGS register are undefined following a reset. Software should not depend on the states of any of these bits.
2. The CD and NW flags are unchanged, bit 4 is set to 1, all other bits are cleared.
3. Where “n” is the Extended Model Value for the respective processor.
4. If Built-In Self-Test (BIST) is invoked on power up or reset, EAX is 0 only if all tests passed. (BIST cannot be invoked during an INIT.)
5. The state of the x87 FPU and MMX registers is not changed by the execution of an INIT.
6. Internal caches are invalid after power-up and RESET, but left unchanged with an INIT.
7. If the processor supports IA-32e mode.



## 15. 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*.

-----

Changes to this chapter: Minor typo correction.

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 27.4), EPT violation, EPT misconfiguration, or page-modification log-full event 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 L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
  - If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT<sup>1</sup>; 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 MSR 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.<sup>2</sup> If an unblocked event causes a VM exit directly, a return to the active state occurs only after the VM exit completes.<sup>3</sup> 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.
    - 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.

---

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

2. If a VM exit takes the processor from an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

3. An exception is made if the logical processor had been inactive due to execution of MWAIT; in this case, it is considered to have become active before the VM exit.

- If a VM exit results from a fault, EPT violation, EPT misconfiguration, or page-modification log-full event is encountered during execution of IRET and the “NMI exiting” VM-execution control is 0, any blocking by NMI is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the VM-exit interruption-information field; see Section 27.2.1 and Section 27.2.2.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, or page-modification log-full event is encountered during execution of IRET and the “virtual NMIs” VM-execution control is 1, virtual-NMI blocking is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the VM-exit interruption-information field; see Section 27.2.1 and 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, EPT misconfiguration, or page-modification log-full event is encountered while executing an instruction, data breakpoints due to that instruction may have been recognized and information about them may be saved in the pending debug exceptions field (unless the VM exit clears that field; see Section 27.3.4).
- The following VM exits are considered to happen after an instruction is executed:
  - VM exits resulting from debug traps (single-step, I/O breakpoints, and data breakpoints).
  - VM exits resulting from debug exceptions whose recognition was delayed by blocking by MOV SS.
  - VM exits resulting from some machine-check exceptions.
  - Trap-like VM exits due to execution of MOV to CR8 when the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1 (see Section 29.3). (Such VM exits can occur only from 64-bit mode and thus only on processors that support Intel 64 architecture.)
  - Trap-like VM exits due to execution of WRMSR when the “use MSR bitmaps” VM-execution control is 1; the value of ECX is in the range 800H–8FFH; and the bit corresponding to the ECX value in write bitmap for low MSRs is 0; and the “virtualize x2APIC mode” VM-execution control is 1. See Section 29.5.
  - VM exits caused by APIC-write emulation (see Section 29.4.3.2) that result from APIC accesses as part of instruction execution.

For these VM exits, the instruction’s modifications to architectural state complete before the VM exit occurs. Such modifications include those to the logical processor’s interruptibility state (see Table 24-3). If there had been blocking by MOV SS, POP SS, or STI before the instruction executed, such blocking is no longer in effect.

A VM exit that occurs in enclave mode sets bit 27 of the exit-reason field and bit 4 of the guest interruptibility-state field. Before such a VM exit is delivered, an Asynchronous Enclave Exit (AEX) occurs (see Chapter 39, “Enclave Exiting Events”). An AEX modifies architectural state (Section 39.3). In particular, the processor establishes the following architectural state as indicated:

- The following bits in RFLAGS are cleared: CF, PF, AF, ZF, SF, OF, and RF.
- FS and GS are restored to the values they had prior to the most recent enclave entry.
- RIP is loaded with the AEP of interrupted enclave thread.
- RSP is loaded from the URSP field in the enclave’s state-save area (SSA).



## 27.2 RECORDING VM-EXIT INFORMATION AND UPDATING VM-ENTRY CONTROL FIELDS

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. Section 27.2.1 to Section 27.2.4 detail the use of these fields.

In addition to updating the VM-exit information fields, the valid bit (bit 31) is cleared in the VM-entry interruption-information field. If bit 5 of the IA32\_VMX\_MISC MSR (index 485H) is read as 1 (see Appendix A.6), the value of IA32\_EFER.LMA is stored into the “IA-32e mode guest” VM-entry control.<sup>1</sup>

### 27.2.1 Basic VM-Exit Information

Section 24.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
  - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix C lists the numbers used and their meaning.
  - Bit 27 of this field is set to 1 if the VM exit occurred while the logical processor was in enclave mode. Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode. A VM exit also sets this bit if it is incident to delivery of an event injected by VM entry and the guest interruptibility-state field indicates an enclave interrupt (bit 4 of the field is 1).
  - The remainder of the field (bits 31:28 and bits 26:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 34.15.2.3).<sup>2</sup>
- **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; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 29.4); EPT violations; EOI virtualization (see Section 29.1.4); APIC-write emulation (see Section 29.4.3.3); and page-modification log full (see Section 28.2.5). For all other VM exits, this field is cleared. The following items provide details:
  - For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 27-1.

**Table 27-1. Exit Qualification for Debug Exceptions**

Bit Position(s)	Contents
3:0	B3 - B0. When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if its corresponding enabling bit in DR7 is not set.
12:4	Reserved (cleared to 0).
13	BD. When set, this bit indicates that the cause of the debug exception is “debug register access detected.”
14	BS. When set, this bit indicates that the cause of the debug exception is either the execution of a single instruction (if RFLAGS.TF = 1 and IA32_DEBUGCTL.BTF = 0) or a taken branch (if RFLAGS.TF = DEBUGCTL.BTF = 1).
63:15	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

1. Bit 5 of the IA32\_VMX\_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.  
 2. Bit 31 of this field is set on certain VM-entry failures; see Section 26.7.

- For a page-fault exception, the exit qualification contains the linear address that caused the page fault. On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the page-fault exception occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

- For a start-up IPI (SIPI), the exit qualification contains the SIPI vector information in bits 7:0. Bits 63:8 of the exit qualification are cleared to 0.
- For a task switch, the exit qualification contains details about the task switch, encoded as shown in Table 27-2.
- For INVLPG, the exit qualification contains the linear-address operand of the instruction.
  - On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
  - If the INVLPG source operand specifies an unusable segment, the linear address specified in the exit qualification will match the linear address that the INVLPG would have used if no VM exit occurred. This address is not architecturally defined and may be implementation-specific.

**Table 27-2. Exit Qualification for Task Switch**

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Reserved (cleared to 0)
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

- For INVEPT, INVPCID, INVVPID, LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, and XSAVES, the exit qualification receives the value of the instruction's displacement field, which is sign-extended to 64 bits if necessary (32 bits on processors that do not support Intel 64 architecture). If the instruction has no displacement (for example, has a register operand), zero is stored into the exit qualification.

On processors that support Intel 64 architecture, an exception is made for RIP-relative addressing (used only in 64-bit mode). Such addressing causes an instruction to use an address that is the sum of the displacement field and the value of RIP that references the following instruction. In this case, the exit qualification is loaded with the sum of the displacement field and the appropriate RIP value.

In all cases, bits of this field beyond the instruction's address size are undefined. For example, suppose that the address-size field in the VM-exit instruction-information field (see Section 24.9.4 and Section 27.2.4) reports an  $n$ -bit address size. Then bits 63: $n$  (bits 31: $n$  on processors that do not support Intel 64 architecture) of the instruction displacement are undefined.

- For a control-register access, the exit qualification contains information about the access and has the format given in Table 27-3.
- For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 27-4.
- For an I/O instruction, the exit qualification contains information about the instruction and has the format given in Table 27-5.

- For MWAIT, the exit qualification contains a value that indicates whether address-range monitoring hardware was armed. The exit qualification is set either to 0 (if address-range monitoring hardware is not armed) or to 1 (if address-range monitoring hardware is armed).
- 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.<sup>1</sup>

If the access to the APIC-access page occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

Such a VM exit that set bits 15:12 of the exit qualification to 0000b (data read during instruction execution) or 0001b (data write during instruction execution) set bit 12—which distinguishes data read from data write—to that which would have been stored in bit 1—W/R—of the page-fault error code had the access caused a page fault instead of an APIC-access VM exit. This implies the following:

- For an APIC-access VM exit caused by the CLFLUSH and CLFLUSHOPT instructions, the access type is “data read during instruction execution.”
- For an APIC-access VM exit caused by the ENTER instruction, the access type is “data write during instruction execution.”

**Table 27-3. Exit Qualification for Control-Register Accesses**

Bit Positions	Contents
3:0	Number of control register (0 for CLTS and LMSW). Bit 3 is always 0 on processors that do not support Intel 64 architecture as they do not support CR8.
5:4	Access type: 0 = MOV to CR 1 = MOV from CR 2 = CLTS 3 = LMSW
6	LMSW operand type: 0 = register 1 = memory  For CLTS and MOV CR, cleared to 0
7	Reserved (cleared to 0)
11:8	For MOV CR, the general-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture)  For CLTS and LMSW, cleared to 0
15:12	Reserved (cleared to 0)

1. The exit qualification is undefined if the access was part of the logging of a branch record or a processor-event-based-sampling (PEBS) record to the DS save area. It is recommended that software configure the paging structures so that no address in the DS save area translates to an address on the APIC-access page.

**Table 27-3. Exit Qualification for Control-Register Accesses (Contd.)**

Bit Positions	Contents
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

- For an APIC-access VM exit caused by the MASKMOVQ instruction or the MASKMOVDQU instruction, the access type is “data write during instruction execution.”
- For an APIC-access VM exit caused by the MONITOR instruction, the access type is “data read during instruction execution.”

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 to the APIC-access page (see Section 29.4.6), the exit qualification is undefined.

- For an EPT violation, the exit qualification contains information about the access causing the EPT violation and has the format given in Table 27-7.

As noted in that table, the format and meaning of the exit qualification depends on the setting of the “mode-based execute control for EPT” VM-execution control and whether the processor supports advanced VM-exit information for EPT violations.<sup>1</sup>

An EPT violation that occurs during as a result of execution of a read-modify-write operation sets bit 1 (data write). Whether it also sets bit 0 (data read) is implementation-specific and, for a given implementation, may differ for different kinds of read-modify-write operations.

**Table 27-4. Exit Qualification for MOV DR**

Bit Position(s)	Contents
2:0	Number of debug register
3	Reserved (cleared to 0)
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Reserved (cleared to 0)
11:8	General-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8 - 15 = R8 - R15, respectively
63:12	Reserved (cleared to 0)

1. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32\_VMX\_EPT\_VPID\_CAP (see Appendix A.10).

**Table 27-5. Exit Qualification for I/O Instructions**

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte  Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in DX or in an immediate operand)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

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.

**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"> <li>▪ If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page.</li> <li>▪ Undefined if the APIC-access VM exit is due a guest-physical access</li> </ul>
15:12	Access type: 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 10 = guest-physical access during event delivery 15 = guest-physical access for an instruction fetch or during instruction execution  Other values not used
63:16	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

- 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.<sup>1</sup> Bits above bit 11 are cleared.
  - For a VM exit due to a page-modification log-full event (Section 28.2.5), only bit 12 of the exit qualification is defined, and only in some cases. It is undefined in 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, it is defined as follows:

- If the “virtual NMIs” VM-execution control is 0, the page-modification log-full event 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 page-modification log-full event 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 these VM exits, all bits other than bit 12 are undefined.

- **Guest-linear address.** For some VM exits, this field receives a linear address that pertains to the VM exit. The field is set for different VM exits as follows:
  - VM exits due to attempts to execute LMSW with a memory operand. In these cases, this field receives the linear address of that operand. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
  - VM exits due to attempts to execute INS or OUTS for which the relevant segment is usable (if the relevant segment is not usable, the value is undefined). (ES is always the relevant segment for INS; for OUTS, the relevant segment is DS unless overridden by an instruction prefix.) The linear address is the base address of relevant segment plus (E)DI (for INS) or (E)SI (for OUTS). Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

**Table 27-7. Exit Qualification for EPT Violations**

Bit Position(s)	Contents
0	Set if the access causing the EPT violation was a data read. <sup>1</sup>
1	Set if the access causing the EPT violation was a data write. <sup>1</sup>
2	Set if the access causing the EPT violation was an instruction fetch.
3	The logical-AND of bit 0 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was readable). <sup>2</sup>
4	The logical-AND of bit 1 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was writeable).

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 3FOH.

Table 27-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
5	The logical-AND of bit 2 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. If the “mode-based execute control for EPT” VM-execution control is 0, this indicates whether the guest-physical address was executable. If that control is 1, this indicates whether the guest-physical address was executable for supervisor-mode linear addresses.
6	If the “mode-based execute control” VM-execution control is 0, the value of this bit is undefined. If that control is 1, this bit is the logical-AND of bit 10 in the EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation. In this case, it indicates whether the guest-physical address was executable for user-mode linear addresses.
7	Set if the guest linear-address field is valid. The guest linear-address field is valid for all EPT violations except those resulting from an attempt to load the guest PDPTes as part of the execution of the MOV CR instruction.
8	If bit 7 is 1: <ul style="list-style-type: none"> <li>▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address.</li> <li>▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit.</li> </ul> Reserved if bit 7 is 0 (cleared to 0).
9	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, <sup>3</sup> this bit is 0 if the linear address is a supervisor-mode linear address and 1 if it is a user-mode linear address. (If CRO.PG = 0, the translation of every linear address is a user-mode linear address and thus this bit will be 1.) Otherwise, this bit is undefined.
10	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, <sup>3</sup> this bit is 0 if paging translates the linear address to a read-only page and 1 if it translates to a read/write page. (If CRO.PG = 0, every linear address is read/write and thus this bit will be 1.) Otherwise, this bit is undefined.
11	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, <sup>3</sup> this bit is 0 if paging translates the linear address to an executable page and 1 if it translates to an execute-disable page. (If CRO.PG = 0, CR4.PAE = 0, or IA32_EFER.NXE = 0, every linear address is executable and thus this bit will be 0.) Otherwise, this bit is undefined.
12	NMI unblocking due to IRET
63:13	Reserved (cleared to 0).

**NOTES:**

1. If accessed and dirty flags for EPT are enabled, processor accesses to guest paging-structure entries are treated as writes with regard to EPT violations (see Section 28.2.3.2). If such an access causes an EPT violation, the processor sets both bit 0 and bit 1 of the exit qualification.
2. Bits 5:3 are cleared to 0 if any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present (see Section 28.2.2).
3. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32\_VMX\_EPT\_VPID\_CAP (see Appendix A.10).
  - VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 27-7; these are all EPT violations except those resulting from an attempt to load the PDPTes as of execution of the MOV CR instruction). The linear address may translate to the guest-physical address whose access caused the EPT violation. Alternatively, translation of the linear address may reference a paging-structure entry whose access caused the EPT violation. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the EPT violation occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

- For all other VM exits, the field is undefined.
- **Guest-physical address.** For a VM exit due to an EPT violation or an EPT misconfiguration, this field receives the guest-physical address that caused the EPT violation or EPT misconfiguration. For all other VM exits, the field is undefined.

If the EPT violation or EPT misconfiguration occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

## 27.2.2 Information for VM Exits Due to Vectored Events

Section 24.9.2 defines fields containing information for VM exits due to the following events: exceptions (including those generated by the instructions INT3, INTO, BOUND, and UD); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs). Such VM exits include those that occur on an attempt at a task switch that causes an exception before generating the VM exit due to the task switch that causes the VM exit.

The following items detail the use of these fields:

- **VM-exit interruption information** (format given in Table 24-15). The following items detail how this field is established for VM exits due to these events:
  - For an exception, bits 7:0 receive the exception vector (at most 31). For an NMI, bits 7:0 are set to 2. For an external interrupt, bits 7:0 receive the vector.
  - Bits 10:8 are set to 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), or 6 (software exception). Hardware exceptions comprise all exceptions except breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.) BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD are hardware exceptions.
  - Bit 11 is set to 1 if the VM exit is caused by a hardware exception that would have delivered an error code on the stack. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).<sup>1</sup> If bit 11 is set to 1, the error code is placed in the VM-exit interruption error code (see below).
  - 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).
    - If the VM exit is due to a double fault (the interruption type is hardware exception and the vector is 8).

Otherwise, bit 12 is defined as follows:

- If the “virtual NMIs” VM-execution control is 0, the VM exit is due to a fault on the IRET instruction (other than a debug exception for an instruction breakpoint), 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 VM exit is due to a fault on the IRET instruction (other than a debug exception for an instruction breakpoint), 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.<sup>2</sup>

- Bits 30:13 are always set to 0.

1. If the capability MSR IA32\_VMX\_CRO\_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

2. The conditions imply that, if the “NMI exiting” VM-execution control is 0 or the “virtual NMIs” VM-execution control is 1, bit 12 is always cleared to 0 by VM exits due to debug exceptions.



- Bit 31 is always set to 1.

For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- VM-exit interruption error code.
  - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the VM-exit interruption-information field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally through the IDT. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the IDT-vectoring information field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).
  - For other VM exits, the value of this field is undefined.

### 27.2.3 Information 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:<sup>1</sup>

- A fault occurs during event delivery and causes a VM exit (because the bit associated with the fault is set to 1 in the exception bitmap).
- A task switch is invoked through a task gate in the IDT. The VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 25.4.2).
- Event delivery causes an APIC-access VM exit (see Section 29.4).
- An EPT violation, EPT misconfiguration, or page-modification log-full event that occurs during event delivery.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 26.5.1.2).

A VM exit is not considered to occur during event delivery in any of the following circumstances:

- The original event causes the VM exit directly (for example, because the original event is a non-maskable interrupt (NMI) and the “NMI exiting” VM-execution control is 1).
- The original event results in a double-fault exception that causes the VM exit directly.
- The VM exit occurred as a result of fetching the first instruction of the handler invoked by the event delivery.
- The VM exit is caused by a triple fault.

The following items detail the use of these fields:

- IDT-vectoring information (format given in Table 24-16). The following items detail how this field is established for VM exits that occur during event delivery:
  - If the VM exit occurred during delivery of an exception, bits 7:0 receive the exception vector (at most 31). If the VM exit occurred during delivery of an NMI, bits 7:0 are set to 2. If the VM exit occurred during delivery of an external interrupt, bits 7:0 receive the vector.
  - Bits 10:8 are set to indicate the type of event that was being delivered when the VM exit occurred: 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 4 (software interrupt), 5 (privileged software interrupt), or 6 (software exception).

Hardware exceptions comprise all exceptions except breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.) BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD are hardware exceptions.

Bits 10:8 may indicate privileged software interrupt if such an event was injected as part of VM entry.

---

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).

- Bit 11 is set to 1 if the VM exit occurred during delivery of a hardware exception that would have delivered an error code on the stack. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).<sup>1</sup> If bit 11 is set to 1, the error code is placed in the IDT-vectoring error code (see below).
- Bit 12 is undefined.
- Bits 30:13 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- IDT-vectoring error code.
  - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the IDT-vectoring information field, this field receives the error code that would have been pushed on the stack by the event that was being delivered through the IDT at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
  - For other VM exits, the value of this field is undefined.

## 27.2.4 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, ENCLS, GETSEC, HLT, IN, INS, INVVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, RDMSR, RDPMSR, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, SGDT, SIDT, SLDT, STR, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WRMSR, XRSTORS, XSETBV, and XSAVES.<sup>2</sup>
  - 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 and for VM exits caused by EPT violations and page-modification log-full events encountered during delivery of a software interrupt, privileged software exception, or software exception.<sup>3</sup>
  - For VM exits due to executions of VMFUNC that fail because one of the following is true:

---

1. If the capability MSR IA32\_VMX\_CRO\_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

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.

- EAX indicates a VM function that is not enabled (the bit at position EAX is 0 in the VM-function controls; see Section 25.5.5.2).
- EAX = 0 and either ECX ≥ 512 or the value of ECX selects an invalid tentative EPTP value (see Section 25.5.5.3).

In all the above cases, this field receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit (see the next paragraph for one exception).

The cases of VM exits encountered during delivery of a software interrupt, privileged software exception, or software exception include those encountered during delivery of events injected as part of VM entry (see Section 26.5.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

If the VM exit occurred in enclave mode, this field is cleared (none of the previous items apply).

**Table 27-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS**

Bit Position(s)	Content
6:0	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
14:10	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for VM exits due to execution of INS.
31:18	Undefined.

- **VM-exit instruction information.** For VM exits due to attempts to execute INS, INVEPT, INVPCID, INVVPID, LIDT, LGDT, LLDT, LTR, OUTS, RDRAND, RDSEED, SIDT, SGDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, or XSAVES, this field receives information about the instruction that caused the VM exit. The format of the field depends on the identity of the instruction causing the VM exit:
  - For VM exits due to attempts to execute INS or OUTS, the field has the format is given in Table 27-8.<sup>1</sup>
  - For VM exits due to attempts to execute INVEPT, INVPCID, or INVVPID, the field has the format is given in Table 27-9.
  - For VM exits due to attempts to execute LIDT, LGDT, SIDT, or SGDT, the field has the format is given in Table 27-10.
  - For VM exits due to attempts to execute LLDT, LTR, SLDT, or STR, the field has the format is given in Table 27-11.
  - For VM exits due to attempts to execute RDRAND or RDSEED, the field has the format is given in Table 27-12.

1. The format of the field was undefined for these VM exits on the first processors to support the virtual-machine extensions. Software can determine whether the format specified in Table 27-8 is used by consulting the VMX capability MSR IA32\_VMX\_BASIC (see Appendix A.1).

- For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, or XSAVES, the field has the format is given in Table 27-13.
- For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 27-14.

For all other VM exits, the field is undefined, unless the VM exit occurred in enclave mode, in which case the field is cleared.

- **I/O RCX, I/O RSI, I/O RDI, I/O RIP.** These fields are undefined except for SMM VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions. See Section 34.15.2.3. Note that, if the VM exit occurred in enclave mode, these fields are all cleared.

**Table 27-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID**

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for memory instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Reg2 (same encoding as IndexReg above)

**Table 27-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT**

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
11	Operand size: 0: 16-bit 1: 32-bit Undefined for VM exits from 64-bit mode.
14:12	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
29:28	Instruction identity: 0: SGDT 1: SIDT 2: LGDT 3: LIDT

**Table 27-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT (Contd.)**

Bit Position(s)	Content
31:30	Undefined.

**Table 27-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR**

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).

**Table 27-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR (Contd.)**

Bit Position(s)	Content
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
29:28	Instruction identity: 0: SLDT 1: STR 2: LLDT 3: LTR
31:30	Undefined.

**Table 27-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND and RDSEED**

Bit Position(s)	Content
2:0	Undefined.
6:3	Destination register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture)
10:7	Undefined.
12:11	Operand size: 0: 16-bit 1: 32-bit 2: 64-bit The value 3 is not used.
31:13	Undefined.

**Table 27-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES**

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.

**Table 27-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES (Contd.)**

Bit Position(s)	Content
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS  Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Undefined.

**Table 27-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE**

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.



**Table 27-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE (Contd.)**

Bit Position(s)	Content
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
31:28	Reg2 (same encoding as Reg1 above)

## 27.3 SAVING GUEST STATE

Each field in the guest-state area of the VMCS (see Section 24.4) is written with the corresponding component of processor state. On processors that support Intel 64 architecture, the full values of each natural-width field (see Section 24.11.2) is saved regardless of the mode of the logical processor before and after the VM exit.

In general, the state saved is that which was in the logical processor at the time the VM exit commences. See Section 27.1 for a discussion of which architectural updates occur at that time.

Section 27.3.1 through Section 27.3.4 provide details for how certain components of processor state are saved. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

### 27.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs is saved as follows:

- The contents of CR0, CR3, CR4, and the IA32\_SYSENTER\_CS, IA32\_SYSENTER\_ESP, and IA32\_SYSENTER\_EIP MSRs are saved into the corresponding fields. Bits 63:32 of the IA32\_SYSENTER\_CS MSR are not saved. On processors that do not support Intel 64 architecture, bits 63:32 of the IA32\_SYSENTER\_ESP and IA32\_SYSENTER\_EIP MSRs are not saved.
- If the “save debug controls” VM-exit control is 1, the contents of DR7 and the IA32\_DEBUGCTL MSR are saved into the corresponding fields. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always saved data into these fields.
- If the “save IA32\_PAT” VM-exit control is 1, the contents of the IA32\_PAT MSR are saved into the corresponding field.
- If the “save IA32\_EFER” VM-exit control is 1, the contents of the IA32\_EFER MSR are saved into the corresponding field.
- If the processor supports either the 1-setting of the “load IA32\_BNDCFGS” VM-entry control or that of the “clear IA32\_BNDCFGS” VM-exit control, the contents of the IA32\_BNDCFGS MSR are saved into the corresponding field.
- The value of the SMBASE field is undefined after all VM exits except SMM VM exits. See Section 34.15.2.

### 27.3.2 Saving Segment Registers and Descriptor-Table Registers

For each segment register (CS, SS, DS, ES, FS, GS, LDTR, or TR), the values saved for the base-address, segment-limit, and access rights are based on whether the register was unusable (see Section 24.4.1) before the VM exit:

- If the register was unusable, the values saved into the following fields are undefined: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in the access-rights field. The following exceptions apply:
  - CS.
    - The base-address and segment-limit fields are saved.
    - The L, D, and G bits are saved in the access-rights field.
  - SS.
    - DPL is saved in the access-rights field.
    - On processors that support Intel 64 architecture, bits 63:32 of the value saved for the base address are always zero.
  - DS and ES. On processors that support Intel 64 architecture, bits 63:32 of the values saved for the base addresses are always zero.
  - FS and GS. The base-address field is saved.
  - LDTR. The value saved for the base address is always canonical.
- If the register was not unusable, the values saved into the following fields are those which were in the register before the VM exit: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in access rights.
- Bits 31:17 and 11:8 in the access-rights field are always cleared. Bit 16 is set to 1 if and only if the segment is unusable.

The contents of the GDTR and IDTR registers are saved into the corresponding base-address and limit fields.

### 27.3.3 Saving RIP, RSP, 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 occurred in enclave mode, the value saved is the AEP of interrupted enclave thread (the remaining items do not apply).

- If the VM exit occurs due to by an attempt to execute an instruction that causes VM exits unconditionally or that has been configured to cause a VM exit via the VM-execution controls, the value saved references that instruction.
- If the VM exit is caused by an occurrence of an INIT signal, a start-up IPI (SIPI), or system-management interrupt (SMI), the value saved is that which was in RIP before the event occurred.
- If the VM exit occurs due to the 1-setting of either the “interrupt-window exiting” VM-execution control or the “NMI-window exiting” VM-execution control, the value saved is that which would be in the register had the VM exit not occurred.
- If the VM exit is due to an external interrupt, non-maskable interrupt (NMI), or hardware exception (as defined in Section 27.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,<sup>1</sup> 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 occurred in enclave mode, the value saved is 0 (the remaining items do not apply).
  - If the VM exit is caused directly by an event that would normally be delivered through the IDT, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate<sup>2</sup> or into the old task-state segment had the event been delivered through a task gate) had the event been delivered through the IDT. See below for VM exits due to task switches caused by task gates in the IDT.
  - If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
  - If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.

---

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.

- 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.<sup>1</sup>
- For APIC-access VM exits and for VM exits caused by EPT violations, EPT misconfigurations, and page-modification log-full events, the value saved depends on whether the VM exit occurred during delivery of an event through the IDT:
  - If the VM exit stored 0 for bit 31 for IDT-vectoring information field (because the VM exit did not occur during delivery of an event through the IDT; see Section 27.2.3), 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.<sup>2</sup> 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.
  - Bit 4 (enclave interruption) is set to 1 if the VM exit occurred while the logical processor was in enclave mode.
 

Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit that is incident to delivery of an event injected by VM entry leaves this bit unmodified.
- The pending debug exceptions field is saved as clear for all VM exits except the following:
  - A VM exit caused by an INIT signal, a machine-check exception, or a system-management interrupt (SMI).
  - A VM exit with basic exit reason "TPR below threshold",<sup>3</sup> "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:

- 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.

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).

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 in any of the following cases:
  - If there was at least one matched data or I/O breakpoint that was enabled in DR7.
  - If it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 26.6.3) and the VM exit occurred before those exceptions were either delivered or lost.
  - If the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled (see Section 16.3.7, “RTM-Enabled Debugger Support,” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*). (This does not apply to VM exits with basic exit reason “monitor trap flag.”)

In other cases, bit 12 is cleared to 0.

- Bit 14 (BS) is set if RFLAGS.TF = 1 in either of the following cases:
  - IA32\_DEBUGCTL.BTF = 0 and the cause of a pending debug exception was the execution of a single instruction.
  - IA32\_DEBUGCTL.BTF = 1 and the cause of a pending debug exception was a taken branch.
- Bit 16 (RTM) is set if a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions had been enabled. (This does not apply to VM exits with basic exit reason “monitor trap flag.”)

- 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:<sup>1</sup>
    - 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 CRO.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.

## 27.4 SAVING MSRS

After processor state is saved to the guest-state area, values of MSRs may be stored into the VM-exit MSR-store area (see Section 24.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-store count) is processed in order by storing the value of the MSR indexed by bits 31:0 (as they would be read by RDMSR) into bits 127:64. Processing of an entry fails in either of the following cases:

- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32\_SMBASE is an MSR that can be read only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be saved on VM exits for model-specific reasons. A processor may prevent certain MSRs (based on the value of bits 31:0) from being stored on VM exits, even if they can normally be read by RDMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4*.
- Bits 63:32 of the entry are not all 0.
- An attempt to read the MSR indexed by bits 31:0 would cause a general-protection exception if executed via RDMSR with CPL = 0.

A VMX abort occurs if processing fails for any entry. See Section 27.7.

## 27.5 LOADING HOST STATE

Processor state is updated on VM exits in the following ways:

- Some state is loaded from or otherwise determined by the contents of the host-state area.
- Some state is determined by VM-exit controls.
- Some state is established in the same way on every VM exit.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order.

On processors that support Intel 64 architecture, the full values of each 64-bit field loaded (for example, the base address for GDTR) is loaded regardless of the mode of the logical processor before and after the VM exit.

The loading of host state is detailed in Section 27.5.1 to Section 27.5.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

A logical processor is in IA-32e mode after a VM exit only if the “host address-space size” VM-exit control is 1. If the logical processor was in IA-32e mode before the VM exit and this control is 0, a VMX abort occurs. See Section 27.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 27.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 27.6). This loading occurs only after the state loading described in this section.

### 27.5.1 Loading Host Control Registers, Debug Registers, MSRs

VM exits load new values for controls registers, debug registers, and some MSRs:

- CR0, CR3, and CR4 are loaded from the CR0 field, the CR3 field, and the CR4 field, respectively, with the following exceptions:
  - The following bits are not modified:

- For CR0, ET, CD, NW; bits 63:32 (on processors that support Intel 64 architecture), 28:19, 17, and 15:6; and any bits that are fixed in VMX operation (see Section 23.8).<sup>1</sup>
  - For CR3, bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width (they are cleared to 0).<sup>2</sup> (This item applies only to processors that support Intel 64 architecture.)
  - For CR4, any bits that are fixed in VMX operation (see Section 23.8).
- CR4.PAE is set to 1 if the "host address-space size" VM-exit control is 1.
  - CR4.PCIDE is set to 0 if the "host address-space size" VM-exit control is 0.
- DR7 is set to 400H.
  - The following MSRs are established as follows:
    - The IA32\_DEBUGCTL MSR is cleared to 00000000\_00000000H.
    - The IA32\_SYSENTER\_CS MSR is loaded from the IA32\_SYSENTER\_CS field. Since that field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
    - IA32\_SYSENTER\_ESP MSR and IA32\_SYSENTER\_EIP MSR are loaded from the IA32\_SYSENTER\_ESP field and the IA32\_SYSENTER\_EIP field, respectively.

If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

If the processor does support the Intel 64 architecture and the processor supports  $N < 64$  linear-address bits, each of bits 63:N is set to the value of bit  $N-1$ .<sup>3</sup>

    - The following steps are performed on processors that support Intel 64 architecture:
      - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 27.5.2).
      - The LMA and LME bits in the IA32\_EFER MSR are each loaded with the setting of the "host address-space size" VM-exit control.
    - If the "load IA32\_PERF\_GLOBAL\_CTRL" VM-exit control is 1, the IA32\_PERF\_GLOBAL\_CTRL MSR is loaded from the IA32\_PERF\_GLOBAL\_CTRL field. Bits that are reserved in that MSR are maintained with their reserved values.
    - If the "load IA32\_PAT" VM-exit control is 1, the IA32\_PAT MSR is loaded from the IA32\_PAT field. Bits that are reserved in that MSR are maintained with their reserved values.
    - If the "load IA32\_EFER" VM-exit control is 1, the IA32\_EFER MSR is loaded from the IA32\_EFER field. Bits that are reserved in that MSR are maintained with their reserved values.
    - If the "clear IA32\_BNDCFGS" VM-exit control is 1, the IA32\_BNDCFGS MSR is cleared to 00000000\_00000000H; otherwise, it is not modified.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-exit MSR-load area. See Section 27.6.

## 27.5.2 Loading Host Segment and Descriptor-Table Registers

Each of the registers CS, SS, DS, ES, FS, GS, and TR is loaded as follows (see below for the treatment of LDTR):

- The selector is loaded from the selector field. The segment is unusable if its selector is loaded with zero. The checks specified Section 26.3.1.2 limit the selector values that may be loaded. In particular, CS and TR are never loaded with zero and are thus never unusable. SS can be loaded with zero only on processors that

- 
1. Bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.
  2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
  3. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.



support Intel 64 architecture and only if the VM exit is to 64-bit mode (64-bit mode allows use of segments marked unusable).

- The base address is set as follows:
  - CS. Cleared to zero.
  - SS, DS, and ES. Undefined if the segment is unusable; otherwise, cleared to zero.
  - FS and GS. Undefined (but, on processors that support Intel 64 architecture, canonical) if the segment is unusable and the VM exit is not to 64-bit mode; otherwise, loaded from the base-address field.
 

If the processor supports the Intel 64 architecture and the processor supports  $N < 64$  linear-address bits, each of bits 63:N is set to the value of bit  $N-1$ .<sup>1</sup> The values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
  - TR. Loaded from the host-state area. If the processor supports the Intel 64 architecture and the processor supports  $N < 64$  linear-address bits, each of bits 63:N is set to the value of bit  $N-1$ .
- The segment limit is set as follows:
  - CS. Set to FFFFFFFFH (corresponding to a descriptor limit of FFFFFFFH and a G-bit setting of 1).
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to FFFFFFFFH.
  - TR. Set to 00000067H.
- The type field and S bit are set as follows:
  - CS. Type set to 11 and S set to 1 (execute/read, accessed, non-conforming code segment).
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, type set to 3 and S set to 1 (read/write, accessed, expand-up data segment).
  - TR. Type set to 11 and S set to 0 (busy 32-bit task-state segment).
- The DPL is set as follows:
  - CS, SS, and TR. Set to 0. The current privilege level (CPL) will be 0 after the VM exit completes.
  - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 0.
- The P bit is set as follows:
  - CS, TR. Set to 1.
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because the value of this control is also loaded into IA32\_EFER.LMA (see Section 27.5.1), no VM exit is ever to compatibility mode (which requires IA32\_EFER.LMA = 1 and CS.L = 0).
- D/B.
  - CS. Loaded with the inverse of the setting of the “host address-space size” VM-exit control. For example, if that control is 0, indicating a 32-bit guest, CS.D/B is set to 1.
  - SS. Set to 1.
  - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
  - TR. Set to 0.
- G.
  - CS. Set to 1.
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
  - TR. Set to 0.

---

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.



The host-state area does not contain a selector field for LDTR. LDTR is established as follows on all VM exits: the selector is cleared to 0000H, the segment is marked unusable and is otherwise undefined (although the base address is always canonical).

The base addresses for GDTR and IDTR are loaded from the GDTR base-address field and the IDTR base-address field, respectively. If the processor supports the Intel 64 architecture and the processor supports  $N < 64$  linear-address bits, each of bits 63:N of each base address is set to the value of bit N-1 of that base address. The GDTR and IDTR limits are each set to FFFFH.

### 27.5.3 Loading Host RIP, RSP, and RFLAGS

RIP and RSP are loaded from the RIP field and the RSP field, respectively. RFLAGS is cleared, except bit 1, which is always set.

### 27.5.4 Checking and Loading Host Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32\_EFER.LMA = 0, the logical processor uses **PAE paging**. See Section 4.4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.<sup>1</sup> When in PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTEs and, if they are valid, loads them into the processor (into internal, non-architectural registers).

A VM exit is to a VMM that uses PAE paging if (1) bit 5 (corresponding to CR4.PAE) is set in the CR4 field in the host-state area of the VMCS; and (2) the "host address-space size" VM-exit control is 0. Such a VM exit may check the validity of the PDPTEs referenced by the CR3 field in the host-state area of the VMCS. Such a VM exit must check their validity if either (1) PAE paging was not in use before the VM exit; or (2) the value of CR3 is changing as a result of the VM exit. A VM exit to a VMM that does not use PAE paging must not check the validity of the PDPTEs.

A VM exit that checks the validity of the PDPTEs uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use. If MOV to CR3 would cause a general-protection exception due to the PDPTEs that would be loaded (e.g., because a reserved bit is set), a VMX abort occurs (see Section 27.7). If a VM exit to a VMM that uses PAE does not cause a VMX abort, the PDPTEs are loaded into the processor as would MOV to CR3, using the value of CR3 being load by the VM exit.

### 27.5.5 Updating Non-Register State

VM exits affect the non-register state of a logical processor as follows:

- A logical processor is always in the active state after a VM exit.
- Event blocking is affected as follows:
  - There is no blocking by STI or by MOV SS after a VM exit.
  - VM exits caused directly by non-maskable interrupts (NMIs) cause blocking by NMI (see Table 24-3). Other VM exits do not affect blocking by NMI. (See Section 27.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 28.3 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 exits invalidate cached mappings:

- If the "enable VPID" VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).

---

1. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

- VM exits are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

## 27.5.6 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 8.10.4 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. VM exits clear any address-range monitoring that may be in effect.

## 27.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 24.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32\_FS\_BASE MSR) or C0000101H (the IA32\_GS\_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32\_SMM\_MONITOR\_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM exits for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4*.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.<sup>1</sup>

If processing fails for any entry, a VMX abort occurs. See Section 27.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM exit, the logical processor does not use any translations that were cached before the transition.

## 27.7 VMX ABORTS

A problem encountered during a VM exit leads to a **VMX abort**. A VMX abort takes a logical processor into a shut-down state as described below.

A VMX abort does not modify the VMCS data in the VMCS region of any active VMCS. The contents of these data are thus suspect after the VMX abort.

On a VMX abort, a logical processor saves a nonzero 32-bit VMX-abort indicator field at byte offset 4 in the VMCS region of the VMCS whose misconfiguration caused the failure (see Section 24.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 27.4).
2. Host checking of the page-directory-pointer-table entries (PDPTes) failed (see Section 27.5.4).
3. The current VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the logical processor cannot complete the VM exit properly.

---

1. Note the following about processors that support Intel 64 architecture. If CRO.PG = 1, WRMSR to the IA32\_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CRO.PG is always 1 in VMX operation, the IA32\_EFER MSR should not be included in the VM-exit MSR-load area for the purpose of modifying the LME bit.

4. There was a failure on loading host MSR (see Section 27.6).
5. There was a machine-check event during VM exit (see Section 27.8).
6. The logical processor was in IA-32e mode before the VM exit and the “host address-space size” VM-entry control was 0 (see Section 27.5).

Some of these causes correspond to failures during the loading of state from the host-state area. Because the loading of such state may be done in any order (see Section 27.5) a VM exit that might lead to a VMX abort for multiple reasons (for example, the current VMCS may be corrupt and the host PDPTs might not be properly configured). In such cases, the VMX-abort indicator could correspond to any one of those reasons.

A logical processor never reads the VMX-abort indicator in a VMCS region and writes it only with one of the non-zero values mentioned above. The VMX-abort indicator allows software on one logical processor to diagnose the VMX-abort on another. For this reason, it is recommended that software running in VMX root operation zero the VMX-abort indicator in the VMCS region of any VMCS that it uses.

After saving the VMX-abort indicator, operation of a logical processor experiencing a VMX abort depends on whether the logical processor is in SMX operation:<sup>1</sup>

- If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000DH, indicating “VMX abort.” See *Intel® Trusted Execution Technology Measured Launched Environment Programming Guide*.
- If the logical processor is outside SMX operation, it issues a special bus cycle (to notify the chipset) and enters the VMX-abort shutdown state. RESET is the only event that wakes a logical processor from the VMX-abort shutdown state. The following events do not affect a logical processor in this state: machine-check events; INIT signals; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

## 27.8 MACHINE-CHECK EVENTS DURING VM EXIT

If a machine-check event occurs during VM exit, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM exit:
  - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:<sup>2</sup>
    - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
    - If the logical processor is outside SMX operation, it goes to the shutdown state.
  - If CR4.MCE = 1, a machine-check exception (#MC) is generated:
    - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
    - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- The machine-check event is handled after VM exit completes:
  - If the VM exit ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
    - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).

---

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

2. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

- If the logical processor is outside SMX operation, it goes to the shutdown state.
  - If the VM exit ends with CR4.MCE = 1, a machine-check exception (#MC) is delivered through the host IDT.
- A VMX abort is generated (see Section 27.7). The logical processor blocks events as done normally in VMX abort. The VMX abort indicator is 5, for “machine-check event during VM exit.”

The first option is not used if the machine-check event occurs after any host state has been loaded. The second option is used only if VM entry is able to load all host state.



## 16. 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*.

-----  
Changes to this chapter: SMRAM typo corrections in Section 34.4.2 "SMRAM Caching".

This chapter describes aspects of IA-64 and IA-32 architecture used in system management mode (SMM).

SMM provides an alternate operating environment that can be used to monitor and manage various system resources for more efficient energy usage, to control system hardware, and/or to run proprietary code. It was introduced into the IA-32 architecture in the Intel386 SL processor (a mobile specialized version of the Intel386 processor). It is also available in the Pentium M, Pentium 4, Intel Xeon, P6 family, and Pentium and Intel486 processors (beginning with the enhanced versions of the Intel486 SL and Intel486 processors).

### 34.1 SYSTEM MANAGEMENT MODE OVERVIEW

SMM is a special-purpose operating mode provided for handling system-wide functions like power management, system hardware control, or proprietary OEM-designed code. It is intended for use only by system firmware, not by applications software or general-purpose systems software. The main benefit of SMM is that it offers a distinct and easily isolated processor environment that operates transparently to the operating system or executive and software applications.

When SMM is invoked through a system management interrupt (SMI), the processor saves the current state of the processor (the processor's context), then switches to a separate operating environment defined by a new address space. The system management software executive (SMI handler) starts execution in that environment, and the critical code and data of the SMI handler reside in a physical memory region (SMRAM) within that address space. While in SMM, the processor executes SMI handler code to perform operations such as powering down unused disk drives or monitors, executing proprietary code, or placing the whole system in a suspended state. When the SMI handler has completed its operations, it executes a resume (RSM) instruction. This instruction causes the processor to reload the saved context of the processor, switch back to protected or real mode, and resume executing the interrupted application or operating-system program or task.

The following SMM mechanisms make it transparent to applications programs and operating systems:

- The only way to enter SMM is by means of an SMI.
- The processor executes SMM code in a separate address space that can be made inaccessible from the other operating modes.
- Upon entering SMM, the processor saves the context of the interrupted program or task.
- All interrupts normally handled by the operating system are disabled upon entry into SMM.
- The RSM instruction can be executed only in SMM.

Section 34.3 describes transitions into and out of SMM. The execution environment after entering SMM is in real-address mode with paging disabled ( $CR0.PE = CR0.PG = 0$ ). In this initial execution environment, the SMI handler can address up to 4 GBytes of memory and can execute all I/O and system instructions. Section 34.5 describes in detail the initial SMM execution environment for an SMI handler and operation within that environment. The SMI handler may subsequently switch to other operating modes while remaining in SMM.

#### NOTES

Software developers should be aware that, even if a logical processor was using the physical-address extension (PAE) mechanism (introduced in the P6 family processors) or was in IA-32e mode before an SMI, this will not be the case after the SMI is delivered. This is because delivery of an SMI disables paging (see Table 34-4). (This does not apply if the dual-monitor treatment of SMIs and SMM is active; see Section 34.15.)

#### 34.1.1 System Management Mode and VMX Operation

Traditionally, SMM services system management interrupts and then resumes program execution (back to the software stack consisting of executive and application software; see Section 34.2 through Section 34.13).

A virtual machine monitor (VMM) using VMX can act as a host to multiple virtual machines and each virtual machine can support its own software stack of executive and application software. On processors that support VMX, virtual-machine extensions may use system-management interrupts (SMIs) and system-management mode (SMM) in one of two ways:

- **Default treatment.** System firmware handles SMIs. The processor saves architectural states and critical states relevant to VMX operation upon entering SMM. When the firmware completes servicing SMIs, it uses RSM to resume VMX operation.
- **Dual-monitor treatment.** Two VM monitors collaborate to control the servicing of SMIs: one VMM operates outside of SMM to provide basic virtualization in support for guests; the other VMM operates inside SMM (while in VMX operation) to support system-management functions. The former is referred to as **executive monitor**, the latter **SMM-transfer monitor (STM)**.<sup>1</sup>

The default treatment is described in Section 34.14, “Default Treatment of SMIs and SMM with VMX Operation and SMX Operation”. Dual-monitor treatment of SMM is described in Section 34.15, “Dual-Monitor Treatment of SMIs and SMM”.

## 34.2 SYSTEM MANAGEMENT INTERRUPT (SMI)

The only way to enter SMM is by signaling an SMI through the SMI# pin on the processor or through an SMI message received through the APIC bus. The SMI is a nonmaskable external interrupt that operates independently from the processor’s interrupt- and exception-handling mechanism and the local APIC. The SMI takes precedence over an NMI and a maskable interrupt. SMM is non-reentrant; that is, the SMI is disabled while the processor is in SMM.

### NOTES

In the Pentium 4, Intel Xeon, and P6 family processors, when a processor that is designated as an application processor during an MP initialization sequence is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked. However if a SMI is received while an application processor is in the wait for SIPI mode, the SMI will be pended. The processor then responds on receipt of a SIPI by immediately servicing the pended SMI and going into SMM before handling the SIPI.

An SMI may be blocked for one instruction following execution of STI, MOV to SS, or POP into SS.

## 34.3 SWITCHING BETWEEN SMM AND THE OTHER PROCESSOR OPERATING MODES

Figure 2-3 shows how the processor moves between SMM and the other processor operating modes (protected, real-address, and virtual-8086). Signaling an SMI while the processor is in real-address, protected, or virtual-8086 modes always causes the processor to switch to SMM. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

### 34.3.1 Entering SMM

The processor always handles an SMI on an architecturally defined “interruptible” point in program execution (which is commonly at an IA-32 architecture instruction boundary). When the processor receives an SMI, it waits for all instructions to retire and for all stores to complete. The processor then saves its current context in SMRAM (see Section 34.4), enters SMM, and begins to execute the SMI handler.

Upon entering SMM, the processor signals external hardware that SMI handling has begun. The signaling mechanism used is implementation dependent. For the P6 family processors, an SMI acknowledge transaction is gener-

---

1. The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32\_VMX\_BASIC (see Appendix A.1) to determine whether it is supported.



ated on the system bus and the multiplexed status signal EXF4 is asserted each time a bus transaction is generated while the processor is in SMM. For the Pentium and Intel486 processors, the SMIACK# pin is asserted.

An SMI has a greater priority than debug exceptions and external interrupts. Thus, if an NMI, maskable hardware interrupt, or a debug exception occurs at an instruction boundary along with an SMI, only the SMI is handled. Subsequent SMI requests are not acknowledged while the processor is in SMM. The first SMI interrupt request that occurs while the processor is in SMM (that is, after SMM has been acknowledged to external hardware) is latched and serviced when the processor exits SMM with the RSM instruction. The processor will latch only one SMI while in SMM.

See Section 34.5 for a detailed description of the execution environment when in SMM.

### 34.3.2 Exiting From SMM

The only way to exit SMM is to execute the RSM instruction. The RSM instruction is only available to the SMI handler; if the processor is not in SMM, attempts to execute the RSM instruction result in an invalid-opcode exception (#UD) being generated.

The RSM instruction restores the processor's context by loading the state save image from SMRAM back into the processor's registers. The processor then returns an SMIACK transaction on the system bus and returns program control back to the interrupted program.

Upon successful completion of the RSM instruction, the processor signals external hardware that SMM has been exited. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is no longer generated on bus cycles. For the Pentium and Intel486 processors, the SMIACK# pin is deserted.

If the processor detects invalid state information saved in the SMRAM, it enters the shutdown state and generates a special bus cycle to indicate it has entered shutdown state. Shutdown happens only in the following situations:

- A reserved bit in control register CR4 is set to 1 on a write to CR4. This error should not happen unless SMI handler code modifies reserved areas of the SMRAM saved state map (see Section 34.4.1). CR4 is saved in the state map in a reserved location and cannot be read or modified in its saved state.
- An illegal combination of bits is written to control register CR0, in particular PG set to 1 and PE set to 0, or NW set to 1 and CD set to 0.
- CR4.PCIDE would be set to 1 and IA32\_EFER.LMA to 0.
- (For the Pentium and Intel486 processors only.) If the address stored in the SMBASE register when an RSM instruction is executed is not aligned on a 32-KByte boundary. This restriction does not apply to the P6 family processors.

In the shutdown state, Intel processors stop executing instructions until a RESET#, INIT# or NMI# is asserted. While Pentium family processors recognize the SMI# signal in shutdown state, P6 family and Intel486 processors do not. Intel does not support using SMI# to recover from shutdown states for any processor family; the response of processors in this circumstance is not well defined. On Pentium 4 and later processors, shutdown will inhibit INTR and A20M but will not change any of the other inhibits. On these processors, NMIs will be inhibited if no action is taken in the SMI handler to uninhibit them (see Section 34.8).

If the processor is in the HALT state when the SMI is received, the processor handles the return from SMM slightly differently (see Section 34.10). Also, the SMBASE address can be changed on a return from SMM (see Section 34.11).

## 34.4 SMRAM

Upon entering SMM, the processor switches to a new address space. Because paging is disabled upon entering SMM, this initial address space maps all memory accesses to the low 4 GBytes of the processor's physical address space. The SMI handler's critical code and data reside in a memory region referred to as system-management RAM (SMRAM). The processor uses a pre-defined region within SMRAM to save the processor's pre-SMI context. SMRAM can also be used to store system management information (such as the system configuration and specific information about powered-down devices) and OEM-specific information.

The default SMRAM size is 64 KBytes beginning at a base physical address in physical memory called the SMBASE (see Figure 34-1). The SMBASE default value following a hardware reset is 30000H. The processor looks for the first instruction of the SMI handler at the address [SMBASE + 8000H]. It stores the processor's state in the area from [SMBASE + FE00H] to [SMBASE + FFFFH]. See Section 34.4.1 for a description of the mapping of the state save area.

The system logic is minimally required to decode the physical address range for the SMRAM from [SMBASE + 8000H] to [SMBASE + FFFFH]. A larger area can be decoded if needed. The size of this SMRAM can be between 32 KBytes and 4 GBytes.

The location of the SMRAM can be changed by changing the SMBASE value (see Section 34.11). It should be noted that all processors in a multiple-processor system are initialized with the same SMBASE value (30000H). Initialization software must sequentially place each processor in SMM and change its SMBASE so that it does not overlap those of other processors.

The actual physical location of the SMRAM can be in system memory or in a separate RAM memory. The processor generates an SMI acknowledge transaction (P6 family processors) or asserts the SMIACK# pin (Pentium and Intel486 processors) when the processor receives an SMI (see Section 34.3.1).

System logic can use the SMI acknowledge transaction or the assertion of the SMIACK# pin to decode accesses to the SMRAM and redirect them (if desired) to specific SMRAM memory. If a separate RAM memory is used for SMRAM, system logic should provide a programmable method of mapping the SMRAM into system memory space when the processor is not in SMM. This mechanism will enable start-up procedures to initialize the SMRAM space (that is, load the SMI handler) before executing the SMI handler during SMM.

### 34.4.1 SMRAM State Save Map

When an IA-32 processor that does not support Intel 64 architecture initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area begins at [SMBASE + 8000H + 7FFFH] and extends down to [SMBASE + 8000H + 7E00H]. Table 34-1 shows the state save map. The offset in column 1 is relative to the SMBASE value plus 8000H. Reserved spaces should not be used by software.

Some of the registers in the SMRAM state save area (marked YES in column 3) may be read and changed by the SMI handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). An SMI handler should not rely on any values stored in an area that is marked as reserved.

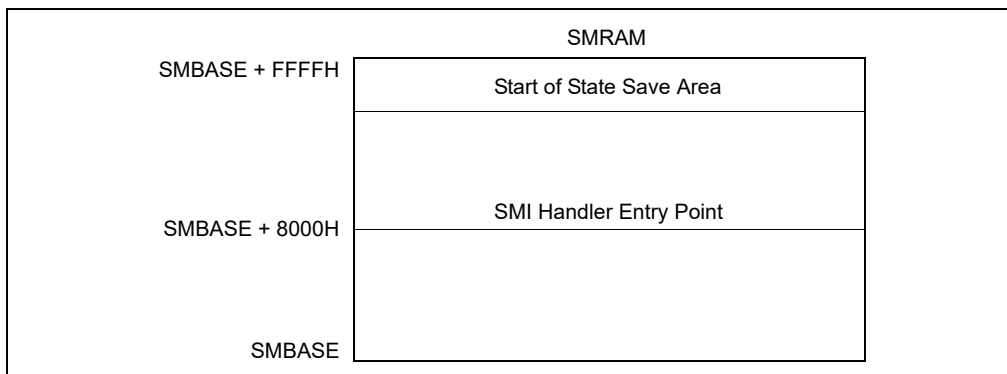


Figure 34-1. SMRAM Usage

Table 34-1. SMRAM State Save Map

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FFCH	CR0	No
7FF8H	CR3	No
7FF4H	EFLAGS	Yes
7FF0H	EIP	Yes
7FECH	EDI	Yes
7FE8H	ESI	Yes
7FE4H	EBP	Yes
7FE0H	ESP	Yes
7FDCH	EBX	Yes
7FD8H	EDX	Yes
7FD4H	ECX	Yes
7FD0H	EAX	Yes
7FCCH	DR6	No
7FC8H	DR7	No
7FC4H	TR <sup>1</sup>	No
7FC0H	Reserved	No
7FBCH	GS <sup>1</sup>	No
7FB8H	FS <sup>1</sup>	No
7FB4H	DS <sup>1</sup>	No
7FB0H	SS <sup>1</sup>	No
7FACH	CS <sup>1</sup>	No
7FA8H	ES <sup>1</sup>	No
7FA4H	I/O State Field, see Section 34.7	No
7FA0H	I/O Memory Address Field, see Section 34.7	No
7F9FH-7F03H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7E00H	Reserved	No

**NOTE:**

1. The two most significant bytes are reserved.

The following registers are saved (but not readable) and restored upon exiting SMM:

- Control register CR4. (This register is cleared to all 0s when entering SMM).
- The hidden segment descriptor information stored in segment registers CS, DS, ES, FS, GS, and SS.

If an SMI request is issued for the purpose of powering down the processor, the values of all reserved locations in the SMM state save must be saved to nonvolatile memory.

The following state is not automatically saved and restored following an SMI and the RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The x87 FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium processors) or test registers TR3 through TR7 (for the Pentium and Intel486 processors).
- The state of the trap controller.
- The machine-check architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The microcode update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage. After the power-on reset, the continuation of the SMI handler should restore these values, along with the rest of the system's state. Anytime the SMI handler changes these registers in the processor, it must also save and restore them.

### NOTES

A small subset of the MSRs (such as, the time-stamp counter and performance-monitoring counters) are not arbitrarily writable and therefore cannot be saved and restored. SMM-based power-down and restoration should only be performed with operating systems that do not use or rely on the values of these registers.

Operating system developers should be aware of this fact and insure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

#### 34.4.1.1 SMRAM State Save Map and Intel 64 Architecture

When the processor initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area on an Intel 64 processor at [SMBASE + 8000H + 7FFFH] and extends to [SMBASE + 8000H + 7C00H].

Support for Intel 64 architecture is reported by CPUID.80000001:EDX[29] = 1. The layout of the SMRAM state save map is shown in Table 34-3.

Additionally, the SMRAM state save map shown in Table 34-3 also applies to processors with the following CPUID signatures listed in Table 34-2, irrespective of the value in CPUID.80000001:EDX[29].

**Table 34-2. Processor Signatures and 64-bit SMRAM State Save Map Format**

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_17H	Intel Xeon Processor 5200, 5400 series, Intel Core 2 Quad processor Q9xxx, Intel Core 2 Duo processors E8000, T9000,
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad, Intel Core 2 Extreme, Intel Core 2 Duo processors, Intel Pentium dual-core processors
06_1CH	45 nm Intel® Atom™ processors

Table 34-3. SMRAM State Save Map for Intel 64 Architecture

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FF8H	CR0	No
7FF0H	CR3	No
7FE8H	RFLAGS	Yes
7FE0H	IA32_EFER	Yes
7FD8H	RIP	Yes
7FD0H	DR6	No
7FC8H	DR7	No
7FC4H	TR SEL <sup>1</sup>	No
7FC0H	LDTR SEL <sup>1</sup>	No
7FBCH	GS SEL <sup>1</sup>	No
7FB8H	FS SEL <sup>1</sup>	No
7FB4H	DS SEL <sup>1</sup>	No
7FB0H	SS SEL <sup>1</sup>	No
7FACH	CS SEL <sup>1</sup>	No
7FA8H	ES SEL <sup>1</sup>	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
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes

Table 34-3. SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
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.

### 34.4.2 SMRAM Caching

An IA-32 processor does not automatically write back and invalidate its caches before entering SMM or before exiting SMM. Because of this behavior, care must be taken in the placement of the SMRAM in system memory and in the caching of the SMRAM to prevent cache incoherence when switching back and forth between SMM and protected mode operation. Any of the following three methods of locating the SMRAM in system memory will guarantee cache coherency.

- Place the SMRAM in a dedicated section of system memory that the operating system and applications are prevented from accessing. Here, the SMRAM can be designated as cacheable (WB, WT, or WC) for optimum processor performance, without risking cache incoherence when entering or exiting SMM.
- Place the SMRAM in a section of memory that overlaps an area used by the operating system (such as the video memory), but designate the SMRAM as uncacheable (UC). This method prevents cache access when in SMM to maintain cache coherency, but the use of uncacheable memory reduces the performance of SMM code.
- Place the SMRAM in a section of system memory that overlaps an area used by the operating system and/or application code, but explicitly flush (write back and invalidate) the caches upon entering and exiting SMM mode. This method maintains cache coherency, but incurs the overhead of two complete cache flushes.

For Pentium 4, Intel Xeon, and P6 family processors, a combination of the first two methods of locating the SMRAM is recommended. Here the SMRAM is split between an overlapping and a dedicated region of memory. Upon entering SMM, the SMRAM space that is accessed overlaps video memory (typically located in low memory). This SMRAM section is designated as UC memory. The initial SMM code then jumps to a second SMRAM section that is located in a dedicated region of system memory (typically in high memory). This SMRAM section can be cached for optimum processor performance.

For systems that explicitly flush the caches upon entering SMM (the third method described above), the cache flush can be accomplished by asserting the FLUSH# pin at the same time as the request to enter SMM (generally initiated by asserting the SMI# pin). The priorities of the FLUSH# and SMI# pins are such that the FLUSH# is serviced first. To guarantee this behavior, the processor requires that the following constraints on the interaction of FLUSH# and SMI# be met. In a system where the FLUSH# and SMI# pins are synchronous and the set up and hold times are met, then the FLUSH# and SMI# pins may be asserted in the same clock. In asynchronous systems, the FLUSH# pin must be asserted at least one clock before the SMI# pin to guarantee that the FLUSH# pin is serviced first.

Upon leaving SMM (for systems that explicitly flush the caches), the WBINVD instruction should be executed prior to leaving SMM to flush the caches.

## NOTES

In systems based on the Pentium processor that use the FLUSH# pin to write back and invalidate cache contents before entering SMM, the processor will prefetch at least one cache line in between when the Flush Acknowledge cycle is run and the subsequent recognition of SMI# and the assertion of SMIACK#.

It is the obligation of the system to ensure that these lines are not cached by returning KEN# inactive to the Pentium processor.

### 34.4.2.1 System Management Range Registers (SMRR)

SMI handler code and data stored by SMM code resides in SMRAM. The SMRR interface is an enhancement in Intel 64 architecture to limit cacheable reference of addresses in SMRAM to code running in SMM. The SMRR interface can be configured only by code running in SMM. Details of SMRR is described in Section 11.11.2.4.

## 34.5 SMI HANDLER EXECUTION ENVIRONMENT

Section 34.5.1 describes the initial execution environment for an SMI handler. An SMI handler may re-configure its execution environment to other supported operating modes. Section 34.5.2 discusses modifications an SMI handler can make to its execution environment.

### 34.5.1 Initial SMM Execution Environment

After saving the current context of the processor, the processor initializes its core registers to the values shown in Table 34-4. Upon entering SMM, the PE and PG flags in control register CR0 are cleared, which places the processor in an environment similar to real-address mode. The differences between the SMM execution environment and the real-address mode execution environment are as follows:

- The addressable address space ranges from 0 to FFFFFFFFH (4 GBytes).
- The normal 64-KByte segment limit for real-address mode is increased to 4 GBytes.
- The default operand and address sizes are set to 16 bits, which restricts the addressable SMRAM address space to the 1-MByte real-address mode limit for native real-address-mode code. However, operand-size and address-size override prefixes can be used to access the address space beyond the 1-MByte.

**Table 34-4. Processor Register Initialization in SMM**

Register	Contents
General-purpose registers	Undefined
EFLAGS	00000002H
EIP	00008000H
CS selector	SMM Base shifted right 4 bits (default 3000H)
CS base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H

**Table 34-4. Processor Register Initialization in SMM**

DS, ES, FS, GS, SS Bases	000000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFH
CR0	PE, EM, TS, and PG flags set to 0; others unmodified
CR4	Cleared to zero
DR6	Undefined
DR7	00000400H

- Near jumps and calls can be made to anywhere in the 4-GByte address space if a 32-bit operand-size override prefix is used. Due to the real-address-mode style of base-address formation, a far call or jump cannot transfer control to a segment with a base address of more than 20 bits (1 MByte). However, since the segment limit in SMM is 4 GBytes, offsets into a segment that go beyond the 1-MByte limit are allowed when using 32-bit operand-size override prefixes. Any program control transfer that does not have a 32-bit operand-size override prefix truncates the EIP value to the 16 low-order bits.
- Data and the stack can be located anywhere in the 4-GByte address space, but can be accessed only with a 32-bit address-size override if they are located above 1 MByte. As with the code segment, the base address for a data or stack segment cannot be more than 20 bits.

The value in segment register CS is automatically set to the default of 30000H for the SMBASE shifted 4 bits to the right; that is, 3000H. The EIP register is set to 8000H. When the EIP value is added to shifted CS value (the SMBASE), the resulting linear address points to the first instruction of the SMI handler.

The other segment registers (DS, SS, ES, FS, and GS) are cleared to 0 and their segment limits are set to 4 GBytes. In this state, the SMRAM address space may be treated as a single flat 4-GByte linear address space. If a segment register is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base (hidden part of the segment register). The limits and attributes are not modified.

Maskable hardware interrupts, exceptions, NMI interrupts, SMI interrupts, A20M interrupts, single-step traps, breakpoint traps, and INIT operations are inhibited when the processor enters SMM. Maskable hardware interrupts, exceptions, single-step traps, and breakpoint traps can be enabled in SMM if the SMM execution environment provides and initializes an interrupt table and the necessary interrupt and exception handlers (see Section 34.6).

### 34.5.2 SMI Handler Operating Mode Switching

Within SMM, an SMI handler may change the processor's operating mode (e.g., to enable PAE paging, enter 64-bit mode, etc.) after it has made proper preparation and initialization to do so. For example, if switching to 32-bit protected mode, the SMI handler should follow the guidelines provided in Chapter 9, "Processor Management and Initialization". If the SMI handler does wish to change operating mode, it is responsible for executing the appropriate mode-transition code after each SMI.

It is recommended that the SMI handler make use of all means available to protect the integrity of its critical code and data. In particular, it should use the system-management range register (SMRR) interface if it is available (see Section 11.11.2.4). The SMRR interface can protect only the first 4 GBytes of the physical address space. The SMI handler should take that fact into account if it uses operating modes that allow access to physical addresses beyond that 4-GByte limit (e.g. PAE paging or 64-bit mode).

Execution of the RSM instruction restores the pre-SMI processor state from the SMRAM state-state map (see Section 34.4.1) into which it was stored when the processor entered SMM. (The SMBASE field in the SMRAM state-state map does not determine the state following RSM but rather the initial environment following the next entry to SMM.) Any required change to operating mode is performed by the RSM instruction; there is no need for the SMI handler to change modes explicitly prior to executing RSM.

## 34.6 EXCEPTIONS AND INTERRUPTS WITHIN SMM

When the processor enters SMM, all hardware interrupts are disabled in the following manner:



- The IF flag in the EFLAGS register is cleared, which inhibits maskable hardware interrupts from being generated.
- The TF flag in the EFLAGS register is cleared, which disables single-step traps.
- Debug register DR7 is cleared, which disables breakpoint traps. (This action prevents a debugger from accidentally breaking into an SMI handler if a debug breakpoint is set in normal address space that overlays code or data in SMRAM.)
- NMI, SMI, and A20M interrupts are blocked by internal SMM logic. (See Section 34.8 for more information about how NMIs are handled in SMM.)

Software-invoked interrupts and exceptions can still occur, and maskable hardware interrupts can be enabled by setting the IF flag. Intel recommends that SMM code be written in so that it does not invoke software interrupts (with the INT *n*, INTO, INT 3, or BOUND instructions) or generate exceptions.

If the SMI handler requires interrupt and exception handling, an SMM interrupt table and the necessary exception and interrupt handlers must be created and initialized from within SMM. Until the interrupt table is correctly initialized (using the LIDT instruction), exceptions and software interrupts will result in unpredictable processor behavior.

The following restrictions apply when designing SMM interrupt and exception-handling facilities:

- The interrupt table should be located at linear address 0 and must contain real-address mode style interrupt vectors (4 bytes containing CS and IP).
- Due to the real-address mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more than 20 bits.
- An interrupt or exception cannot transfer control to a segment offset of more than 16 bits (64 KBytes).
- When an exception or interrupt occurs, only the 16 least-significant bits of the return address (EIP) are pushed onto the stack. If the offset of the interrupted procedure is greater than 64 KBytes, it is not possible for the interrupt/exception handler to return control to that procedure. (One solution to this problem is for a handler to adjust the return address on the stack.)
- The SMBASE relocation feature affects the way the processor will return from an interrupt or exception generated while the SMI handler is executing. For example, if the SMBASE is relocated to above 1 MByte, but the exception handlers are below 1 MByte, a normal return to the SMI handler is not possible. One solution is to provide the exception handler with a mechanism for calculating a return address above 1 MByte from the 16-bit return address on the stack, then use a 32-bit far call to return to the interrupted procedure.
- If an SMI handler needs access to the debug trap facilities, it must insure that an SMM accessible debug handler is available and save the current contents of debug registers DR0 through DR3 (for later restoration). Debug registers DR0 through DR3 and DR7 must then be initialized with the appropriate values.
- If an SMI handler needs access to the single-step mechanism, it must insure that an SMM accessible single-step handler is available, and then set the TF flag in the EFLAGS register.
- If the SMI design requires the processor to respond to maskable hardware interrupts or software-generated interrupts while in SMM, it must ensure that SMM accessible interrupt handlers are available and then set the IF flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked upon entry to SMM, so they do not need to be enabled.

## 34.7 MANAGING SYNCHRONOUS AND ASYNCHRONOUS SYSTEM MANAGEMENT INTERRUPTS

When coding for a multiprocessor system or a system with Intel HT Technology, it was not always possible for an SMI handler to distinguish between a synchronous SMI (triggered during an I/O instruction) and an asynchronous SMI. To facilitate the discrimination of these two events, incremental state information has been added to the SMM state save map.

Processors that have an SMM revision ID of 30004H or higher have the incremental state information described below.

### 34.7.1 I/O State Implementation

Within the extended SMM state save map, a bit (IO\_SMI) is provided that is set only when an SMI is either taken immediately after a *successful* I/O instruction or is taken after a *successful* iteration of a REP I/O instruction (the *successful* notion pertains to the processor point of view; not necessarily to the corresponding platform function). When set, the IO\_SMI bit provides a strong indication that the corresponding SMI was synchronous. In this case, the SMM State Save Map also supplies the port address of the I/O operation. The IO\_SMI bit and the I/O Port Address may be used in conjunction with the information logged by the platform to confirm that the SMI was indeed synchronous.

The IO\_SMI bit by itself is a strong indication, not a guarantee, that the SMI is synchronous. This is because an asynchronous SMI might coincidentally be taken after an I/O instruction. In such a case, the IO\_SMI bit would still be set in the SMM state save map.

Information characterizing the I/O instruction is saved in two locations in the SMM State Save Map (Table 34-5). The IO\_SMI bit also serves as a valid bit for the rest of the I/O information fields. The contents of these I/O information fields are not defined when the IO\_SMI bit is not set.

**Table 34-5. I/O Instruction Information in the SMM State Save Map**

State (SMM Rev. ID: 30004H or higher)	Format								
	31	16	15	8	7	4	3	1	0
I/O State Field SMRAM offset 7FA4		I/O Port		Reserved		I/O Type		I/O Length	IO_SMI
	31								0
I/O Memory Address Field SMRAM offset 7FA0	I/O Memory Address								

When IO\_SMI is set, the other fields may be interpreted as follows:

- I/O length:
  - 001 – Byte
  - 010 – Word
  - 100 – Dword
- I/O instruction type (Table 34-6)

**Table 34-6. I/O Instruction Type Encodings**

Instruction	Encoding
IN Immediate	1001
IN DX	0001
OUT Immediate	1000
OUT DX	0000
INS	0011
OUTS	0010
REP INS	0111
REP OUTS	0110

## 34.8 NMI HANDLING WHILE IN SMM

NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence. This assumes that NMIs were not blocked before the SMI occurred. If NMIs were blocked before the SMI occurred, they are blocked after execution of RSM.

Although NMI requests are blocked when the processor enters SMM, they may be enabled through software by executing an IRET instruction. If the SMI handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same “real mode” manner in which they are handled outside of SMM.

A special case can occur if an SMI handler nests inside an NMI handler and then another NMI occurs. During NMI interrupt handling, NMI interrupts are disabled, so normally NMI interrupts are serviced and completed with an IRET instruction one at a time. When the processor enters SMM while executing an NMI handler, the processor saves the SMRAM state save map but does not save the attribute to keep NMI interrupts disabled. Potentially, an NMI could be latched (while in SMM or upon exit) and serviced upon exit of SMM even though the previous NMI handler has still not completed. One or more NMIs could thus be nested inside the first NMI handler. The NMI interrupt handler should take this possibility into consideration.

Also, for the Pentium processor, exceptions that invoke a trap or fault handler will enable NMI interrupts from inside of SMM. This behavior is implementation specific for the Pentium processor and is not part of the IA-32 architecture.

## 34.9 SMM REVISION IDENTIFIER

The SMM revision identifier field is used to indicate the version of SMM and the SMM extensions that are supported by the processor (see Figure 34-2). The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at offset 7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture.

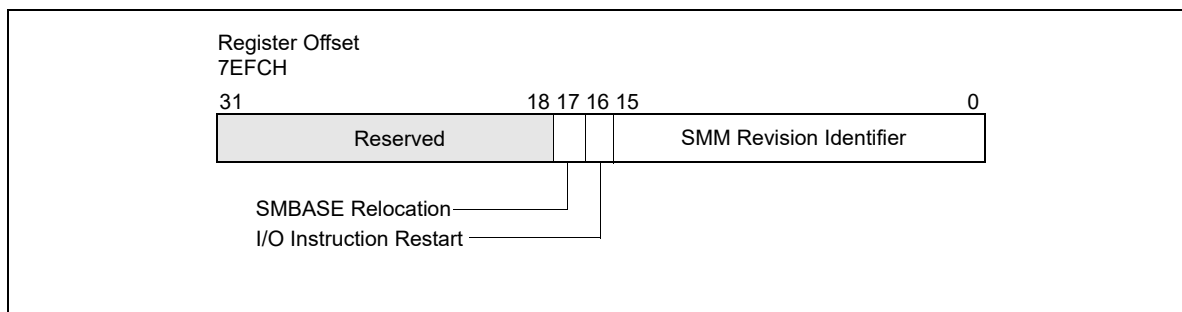


Figure 34-2. SMM Revision Identifier

The upper word of the SMM revision identifier refers to the extensions available. If the I/O instruction restart flag (bit 16) is set, the processor supports the I/O instruction restart (see Section 34.12); if the SMBASE relocation flag (bit 17) is set, SMRAM base address relocation is supported (see Section 34.11).

## 34.10 AUTO HALT RESTART

If the processor is in a HALT state (due to the prior execution of a HLT instruction) when it receives an SMI, the processor records the fact in the auto HALT restart flag in the saved processor state (see Figure 34-3). (This flag is located at offset 7F02H and bit 0 in the state save area of the SMRAM.)

If the processor sets the auto HALT restart flag upon entering SMM (indicating that the SMI occurred when the processor was in the HALT state), the SMI handler has two options:

- It can leave the auto HALT restart flag set, which instructs the RSM instruction to return program control to the HLT instruction. This option in effect causes the processor to re-enter the HALT state after handling the SMI. (This is the default operation.)
- It can clear the auto HALT restart flag, which instructs the RSM instruction to return program control to the instruction following the HLT instruction.

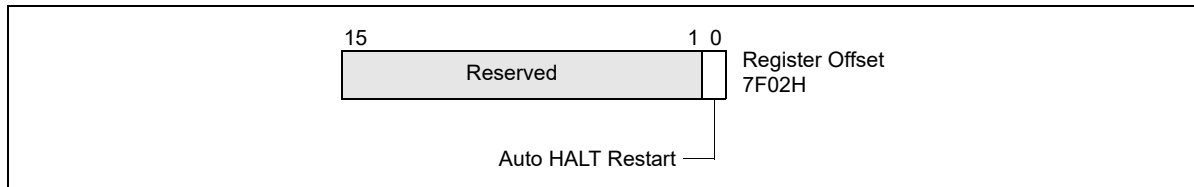


Figure 34-3. Auto HALT Restart Field

These options are summarized in Table 34-7. If the processor was not in a HALT state when the SMI was received (the auto HALT restart flag is cleared), setting the flag to 1 will cause unpredictable behavior when the RSM instruction is executed.

Table 34-7. Auto HALT Restart Flag Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
0	0	Returns to next instruction in interrupted program or task.
0	1	Unpredictable.
1	0	Returns to next instruction after HLT instruction.
1	1	Returns to HALT state.

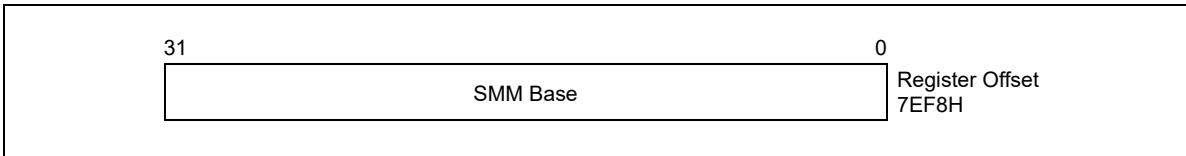
If the HLT instruction is restarted, the processor will generate a memory access to fetch the HLT instruction (if it is not in the internal cache), and execute a HLT bus transaction. This behavior results in multiple HLT bus transactions for the same HLT instruction.

### 34.10.1 Executing the HLT Instruction in SMM

The HLT instruction should not be executed during SMM, unless interrupts have been enabled by setting the IF flag in the EFLAGS register. If the processor is halted in SMM, the only event that can remove the processor from this state is a maskable hardware interrupt or a hardware reset.

## 34.11 SMBASE RELOCATION

The default base address for the SMRAM is 30000H. This value is contained in an internal processor register called the SMBASE register. The operating system or executive can relocate the SMRAM by setting the SMBASE field in the saved state map (at offset 7EF8H) to a new value (see Figure 34-4). The RSM instruction reloads the internal SMBASE register with the value in the SMBASE field each time it exits SMM. All subsequent SMI requests will use the new SMBASE value to find the starting address for the SMI handler (at SMBASE + 8000H) and the SMRAM state save area (from SMBASE + FE00H to SMBASE + FFFFH). (The processor resets the value in its internal SMBASE register to 30000H on a RESET, but does not change it on an INIT.)



**Figure 34-4. SMBASE Relocation Field**

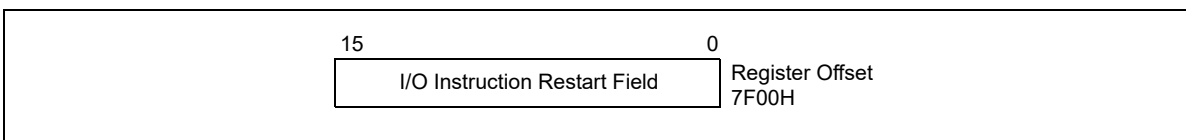
In multiple-processor systems, initialization software must adjust the SMBASE value for each processor so that the SMRAM state save areas for each processor do not overlap. (For Pentium and Intel486 processors, the SMBASE values must be aligned on a 32-KByte boundary or the processor will enter shutdown state during the execution of a RSM instruction.)

If the SMBASE relocation flag in the SMM revision identifier field is set, it indicates the ability to relocate the SMBASE (see Section 34.9).

## 34.12 I/O INSTRUCTION RESTART

If the I/O instruction restart flag in the SMM revision identifier field is set (see Section 34.9), the I/O instruction restart mechanism is present on the processor. This mechanism allows an interrupted I/O instruction to be re-executed upon returning from SMM mode. For example, if an I/O instruction is used to access a powered-down I/O device, a chip set supporting this device can intercept the access and respond by asserting SMI#. This action invokes the SMI handler to power-up the device. Upon returning from the SMI handler, the I/O instruction restart mechanism can be used to re-execute the I/O instruction that caused the SMI.

The I/O instruction restart field (at offset 7F00H in the SMM state-save area, see Figure 34-5) controls I/O instruction restart. When an RSM instruction is executed, if this field contains the value FFH, then the EIP register is modified to point to the I/O instruction that received the SMI request. The processor will then automatically re-execute the I/O instruction that the SMI trapped. (The processor saves the necessary machine state to insure that re-execution of the instruction is handled coherently.)



**Figure 34-5. I/O Instruction Restart Field**

If the I/O instruction restart field contains the value 00H when the RSM instruction is executed, then the processor begins program execution with the instruction following the I/O instruction. (When a repeat prefix is being used, the next instruction may be the next I/O instruction in the repeat loop.) Not re-executing the interrupted I/O instruction is the default behavior; the processor automatically initializes the I/O instruction restart field to 00H upon entering SMM. Table 34-8 summarizes the states of the I/O instruction restart field.

**Table 34-8. I/O Instruction Restart Field Values**

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
00H	00H	Does not re-execute trapped I/O instruction.
00H	FFH	Re-executes trapped I/O instruction.

The I/O instruction restart mechanism does not indicate the cause of the SMI. It is the responsibility of the SMI handler to examine the state of the processor to determine the cause of the SMI and to determine if an I/O instruction was interrupted and should be restarted upon exiting SMM. If an SMI interrupt is signaled on a non-I/O instruction boundary, setting the I/O instruction restart field to FFH prior to executing the RSM instruction will likely result in a program error.

### 34.12.1 Back-to-Back SMI Interrupts When I/O Instruction Restart Is Being Used

If an SMI interrupt is signaled while the processor is servicing an SMI interrupt that occurred on an I/O instruction boundary, the processor will service the new SMI request before restarting the originally interrupted I/O instruction. If the I/O instruction restart field is set to FFH prior to returning from the second SMI handler, the EIP will point to an address different from the originally interrupted I/O instruction, which will likely lead to a program error. To avoid this situation, the SMI handler must be able to recognize the occurrence of back-to-back SMI interrupts when I/O instruction restart is being used and insure that the handler sets the I/O instruction restart field to 00H prior to returning from the second invocation of the SMI handler.

## 34.13 SMM MULTIPLE-PROCESSOR CONSIDERATIONS

The following should be noted when designing multiple-processor systems:

- Any processor in a multiprocessor system can respond to an SMM.
- Each processor needs its own SMRAM space. This space can be in system memory or in a separate RAM.
- The SMRAMs for different processors can be overlapped in the same memory space. The only stipulation is that each processor needs its own state save area and its own dynamic data storage area. (Also, for the Pentium and Intel486 processors, the SMBASE address must be located on a 32-KByte boundary.) Code and static data can be shared among processors. Overlapping SMRAM spaces can be done more efficiently with the P6 family processors because they do not require that the SMBASE address be on a 32-KByte boundary.
- The SMI handler will need to initialize the SMBASE for each processor.
- Processors can respond to local SMIs through their SMI# pins or to SMIs received through the APIC interface. The APIC interface can distribute SMIs to different processors.
- Two or more processors can be executing in SMM at the same time.
- When operating Pentium processors in dual processing (DP) mode, the SMIACT# pin is driven only by the MRM processor and should be sampled with ADS#. For additional details, see Chapter 14 of the *Pentium Processor Family User's Manual, Volume 1*.

SMM is not re-entrant, because the SMRAM State Save Map is fixed relative to the SMBASE. If there is a need to support two or more processors in SMM mode at the same time then each processor should have dedicated SMRAM spaces. This can be done by using the SMBASE Relocation feature (see Section 34.11).

## 34.14 DEFAULT TREATMENT OF SMIS AND SMM WITH VMX OPERATION AND SMX OPERATION

Under the default treatment, the interactions of SMIs and SMM with VMX operation are few. This section details those interactions. It also explains how this treatment affects SMX operation.

### 34.14.1 Default Treatment of SMI Delivery

Ordinary SMI delivery saves processor state into SMRAM and then loads state based on architectural definitions. Under the default treatment, processors that support VMX operation perform SMI delivery as follows:

```

enter SMM;
save the following internal to the processor:
    CR4.VMXE
    an indication of whether the logical processor was in VMX operation (root or non-root)
IF the logical processor is in VMX operation
    THEN
        save current VMCS pointer internal to the processor;
        leave VMX operation;
        save VMX-critical state defined below;

```

```

FI;
IF the logical processor supports SMX operation
  THEN
    save internal to the logical processor an indication of whether the Intel® TXT private space is locked;
    IF the TXT private space is unlocked
      THEN lock the TXT private space;
    FI;
FI;
CR4.VMXE ← 0;
perform ordinary SMI delivery:
  save processor state in SMRAM;
  set processor state to standard SMM values;1
  invalidate linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H
  are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 28.3);

```

The pseudocode above makes reference to the saving of **VMX-critical state**. This state consists of the following: (1) SS.DPL (the current privilege level); (2) RFLAGS.VM<sup>2</sup>; (3) the state of blocking by STI and by MOV SS (see Table 24-3 in Section 24.4.2); (4) the state of virtual-NMI blocking (only if the processor is in VMX non-root operation and the “virtual NMIs” VM-execution control is 1); and (5) an indication of whether an MTF VM exit is pending (see Section 25.5.2). These data may be saved internal to the processor or in the VMCS region of the current VMCS. Processors that do not support SMI recognition while there is blocking by STI or by MOV SS need not save the state of such blocking.

If the logical processor supports the 1-setting of the “enable EPT” VM-execution control and the logical processor was in VMX non-root operation at the time of an SMI, it saves the value of that control into bit 0 of the 32-bit field at offset SMBASE + 8000H + 7EE0H (SMBASE + FEE0H; see Table 34-3).<sup>3</sup> If the logical processor was not in VMX non-root operation at the time of the SMI, it saves 0 into that bit. If the logical processor saves 1 into that bit (it was in VMX non-root operation and the “enable EPT” VM-execution control was 1), it saves the value of the EPT pointer (EPTP) into the 64-bit field at offset SMBASE + 8000H + 7ED8H (SMBASE + FED8H).

Because SMI delivery causes a logical processor to leave VMX operation, all the controls associated with VMX non-root operation are disabled in SMM and thus cannot cause VM exits while the logical processor in SMM.

### 34.14.2 Default Treatment of RSM

Ordinary execution of RSM restores processor state from SMRAM. Under the default treatment, processors that support VMX operation perform RSM as follows:

```

IF VMXE = 1 in CR4 image in SMRAM
  THEN fail and enter shutdown state;
  ELSE
    restore state normally from SMRAM;
    invalidate linear mappings and combined mappings associated with all VPIDs and all PCIDs; combined mappings are invalidated
    for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 28.3);
    IF the logical processor supports SMX operation and the Intel® TXT private space was unlocked at the time of the last SMI (as
    saved)
      THEN unlock the TXT private space;
    FI;
    CR4.VMXE ← value stored internally;

```

1. This causes the logical processor to block INIT signals, NMIs, and SMIs.
2. Section 34.14 and Section 34.15 use the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of these registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to the lower 32 bits of the register.
3. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, SMI functions as the “enable EPT” VM-execution control were 0. See Section 24.6.2.



IF internal storage indicates that the logical processor  
had been in VMX operation (root or non-root)

THEN

enter VMX operation (root or non-root);

restore VMX-critical state as defined in Section 34.14.1;

set to their fixed values any bits in CR0 and CR4 whose values must be fixed in VMX operation (see Section 23.8);<sup>1</sup>

IF RFLAGS.VM = 0 AND (in VMX root operation OR the “unrestricted guest” VM-execution control is 0)<sup>2</sup>

THEN

CS.RPL ← SS.DPL;

SS.RPL ← SS.DPL;

FI;

restore current VMCS pointer;

FI;

leave SMM;

IF logical processor will be in VMX operation or in SMX operation after RSM

THEN block A20M and leave A20M mode;

FI;

FI;

RSM unblocks SMIs. It restores the state of blocking by NMI (see Table 24-3 in Section 24.4.2) as follows:

- If the RSM is not to VMX non-root operation or if the “virtual NMIs” VM-execution control will be 0, the state of NMI blocking is restored normally.
- If the RSM is to VMX non-root operation and the “virtual NMIs” VM-execution control will be 1, NMIs are not blocked after RSM. The state of virtual-NMI blocking is restored as part of VMX-critical state.

INIT signals are blocked after RSM if and only if the logical processor will be in VMX root operation.

If RSM returns a logical processor to VMX non-root operation, it re-establishes the controls associated with the current VMCS. If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs immediately after RSM if the enabling conditions apply. The same is true for the “NMI-window exiting” VM-execution control. Such VM exits occur with their normal priority. See Section 25.2.

If an MTF VM exit was pending at the time of the previous SMI, an MTF VM exit is pending on the instruction boundary following execution of RSM. The following items detail the treatment of MTF VM exits that may be pending following RSM:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these MTF VM exits. These MTF VM exits take priority over debug-trap exceptions and lower priority events.
- These MTF VM exits wake the logical processor if RSM caused the logical processor to enter the HLT state (see Section 34.10). They do not occur if the logical processor just entered the shutdown state.

### 34.14.3 Protection of CR4.VMXE in SMM

Under the default treatment, CR4.VMXE is treated as a reserved bit while a logical processor is in SMM. Any attempt by software running in SMM to set this bit causes a general-protection exception. In addition, software cannot use VMX instructions or enter VMX operation while in SMM.

### 34.14.4 VMXOFF and SMI Unblocking

The VMXOFF instruction can be executed only with the default treatment (see Section 34.15.1) and only outside SMM. If SMIs are blocked when VMXOFF is executed, VMXOFF unblocks them unless

1. If the RSM is to VMX non-root operation and both the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls will be 1, CR0.PE and CR0.PG retain the values that were loaded from SMRAM regardless of what is reported in the capability MSR IA32\_VMX\_CRO\_FIXED0.
2. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 24.6.2.



IA32\_SMM\_MONITOR\_CTL[bit 2] is 1 (see Section 34.15.5 for details regarding this MSR).<sup>1</sup> Section 34.15.7 identifies a case in which SMIs may be blocked when VMXOFF is executed.

Not all processors allow this bit to be set to 1. Software should consult the VMX capability MSR IA32\_VMX\_MISC (see Appendix A.6) to determine whether this is allowed.

## 34.15 DUAL-MONITOR TREATMENT OF SMIs AND SMM

Dual-monitor treatment is activated through the cooperation of the **executive monitor** (the VMM that operates outside of SMM to provide basic virtualization) and the **SMM-transfer monitor (STM)** (the VMM that operates inside SMM—while in VMX operation—to support system-management functions). Control is transferred to the STM through VM exits; VM entries are used to return from SMM.

The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32\_VMX\_BASIC (see Appendix A.1) to determine whether it is supported.

### 34.15.1 Dual-Monitor Treatment Overview

The dual-monitor treatment uses an executive monitor and an SMM-transfer monitor (STM). Transitions from the executive monitor or its guests to the STM are called **SMM VM exits** and are discussed in Section 34.15.2. SMM VM exits are caused by SMIs as well as executions of VMCALL in VMX root operation. The latter allow the executive monitor to call the STM for service.

The STM runs in VMX root operation and uses VMX instructions to establish a VMCS and perform VM entries to its own guests. This is done all inside SMM (see Section 34.15.3). The STM returns from SMM, not by using the RSM instruction, but by using a VM entry that returns from SMM. Such VM entries are described in Section 34.15.4.

Initially, there is no STM and the default treatment (Section 34.14) is used. The dual-monitor treatment is not used until it is enabled and activated. The steps to do this are described in Section 34.15.5 and Section 34.15.6.

It is not possible to leave VMX operation under the dual-monitor treatment; VMXOFF will fail if executed. The dual-monitor treatment must be deactivated first. The STM deactivates dual-monitor treatment using a VM entry that returns from SMM with the “deactivate dual-monitor treatment” VM-entry control set to 1 (see Section 34.15.7).

The executive monitor configures any VMCS that it uses for VM exits to the executive monitor. SMM VM exits, which transfer control to the STM, use a different VMCS. Under the dual-monitor treatment, each logical processor uses a separate VMCS called the **SMM-transfer VMCS**. When the dual-monitor treatment is active, the logical processor maintains another VMCS pointer called the **SMM-transfer VMCS pointer**. The SMM-transfer VMCS pointer is established when the dual-monitor treatment is activated.

### 34.15.2 SMM VM Exits

An SMM VM exit is a VM exit that begins outside SMM and that ends in SMM.

Unlike other VM exits, SMM VM exits can begin in VMX root operation. SMM VM exits result from the arrival of an SMI outside SMM or from execution of VMCALL in VMX root operation outside SMM. Execution of VMCALL in VMX root operation causes an SMM VM exit only if the valid bit is set in the IA32\_SMM\_MONITOR\_CTL MSR (see Section 34.15.5).

Execution of VMCALL in VMX root operation causes an SMM VM exit even under the default treatment. This SMM VM exit activates the dual-monitor treatment (see Section 34.15.6).

Differences between SMM VM exits and other VM exits are detailed in Sections 34.15.2.1 through 34.15.2.5. Differences between SMM VM exits that activate the dual-monitor treatment and other SMM VM exits are described in Section 34.15.6.

---

1. Setting IA32\_SMM\_MONITOR\_CTL[bit 2] to 1 prevents VMXOFF from unblocking SMIs regardless of the value of the register’s valid bit (bit 0).

### 34.15.2.1 Architectural State Before a VM Exit

System-management interrupts (SMIs) that cause SMM VM exits always do so directly. They do not save state to SMRAM as they do under the default treatment.

### 34.15.2.2 Updating the Current-VMCS and Executive-VMCS Pointers

SMM VM exits begin by performing the following steps:

1. The executive-VMCS pointer field in the SMM-transfer VMCS is loaded as follows:
  - If the SMM VM exit commenced in VMX non-root operation, it receives the current-VMCS pointer.
  - If the SMM VM exit commenced in VMX root operation, it receives the VMXON pointer.
2. The current-VMCS pointer is loaded with the value of the SMM-transfer VMCS pointer.

The last step ensures that the current VMCS is the SMM-transfer VMCS. VM-exit information is recorded in that VMCS, and VM-entry control fields in that VMCS are updated. State is saved into the guest-state area of that VMCS. The VM-exit controls and host-state area of that VMCS determine how the VM exit operates.

### 34.15.2.3 Recording VM-Exit Information

SMM VM exits differ from other VM exit with regard to the way they record VM-exit information. The differences follow.

- **Exit reason.**
  - Bits 15:0 of this field contain the basic exit reason. The field is loaded with the reason for the SMM VM exit: I/O SMI (an SMI arrived immediately after retirement of an I/O instruction), other SMI, or VMCALL. See Appendix C, “VMX Basic Exit Reasons”.
  - SMM VM exits are the only VM exits that may occur in VMX root operation. Because the SMM-transfer monitor may need to know whether it was invoked from VMX root or VMX non-root operation, this information is stored in bit 29 of the exit-reason field (see Table 24-14 in Section 24.9.1). The bit is set by SMM VM exits from VMX root operation.
  - If the SMM VM exit occurred in VMX non-root operation and an MTF VM exit was pending, bit 28 of the exit-reason field is set; otherwise, it is cleared.
  - Bits 27:16 and bits 31:30 are cleared.
- **Exit qualification.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, the exit qualification contains information about the I/O instruction that retired immediately before the SMI. It has the format given in Table 34-9.

**Table 34-9. Exit Qualification for SMIs That Arrive Immediately After the Retirement of an I/O Instruction**

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte  Other values not used.
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)

**Table 34-9. Exit Qualification for SMIs That Arrive Immediately After the Retirement of an I/O Instruction (Contd.)**

Bit Position(s)	Contents
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in the I/O instruction)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

- **Guest linear address.** This field is used for VM exits due to SMIs that arrive immediately after the retirement of an INS or OUTS instruction for which the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) is usable. The field receives the value of the linear address generated by ES:(E)DI (for INS) or segment:(E)SI (for OUTS; the default segment is DS but can be overridden by a segment override prefix) at the time the instruction started. If the relevant segment is not usable, the value is undefined. On processors that support Intel 64 architecture, bits 63:32 are clear if the logical processor was not in 64-bit mode before the VM exit.
- **I/O RCX, I/O RSI, I/O RDI, and I/O RIP.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, these fields receive the values that were in RCX, RSI, RDI, and RIP, respectively, before the I/O instruction executed. Thus, the value saved for I/O RIP addresses the I/O instruction.

#### 34.15.2.4 Saving Guest State

SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area.

The value of the VMX-preemption timer is saved into the corresponding field in the guest-state area if the “save VMX-preemption timer value” VM-exit control is 1. That field becomes undefined if, in addition, either the SMM VM exit is from VMX root operation or the SMM VM exit is from VMX non-root operation and the “activate VMX-preemption timer” VM-execution control is 0.

#### 34.15.2.5 Updating Non-Register State

SMM VM exits affect the non-register state of a logical processor as follows:

- SMM VM exits cause non-maskable interrupts (NMIs) to be blocked; they may be unblocked through execution of IRET or through a VM entry (depending on the value loaded for the interruptibility state and the setting of the “virtual NMIs” VM-execution control).
- SMM VM exits cause SMIs to be blocked; they may be unblocked by a VM entry that returns from SMM (see Section 34.15.4).

SMM VM exits invalidate linear mappings and combined mappings associated with VPID 0000H for all PCIDs. Combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 28.3). (Ordinary VM exits are not required to perform such invalidation if the “enable VPID” VM-execution control is 1; see Section 27.5.5.)

### 34.15.3 Operation of the SMM-Transfer Monitor

Once invoked, the SMM-transfer monitor (STM) is in VMX root operation and can use VMX instructions to configure VMCSs and to cause VM entries to virtual machines supported by those structures. As noted in Section 34.15.1, the VMXOFF instruction cannot be used under the dual-monitor treatment and thus cannot be used by the STM.

The RSM instruction also cannot be used under the dual-monitor treatment. As noted in Section 25.1.3, it causes a VM exit if executed in SMM in VMX non-root operation. If executed in VMX root operation, it causes an invalid-opcode exception. The STM uses VM entries to return from SMM (see Section 34.15.4).

### 34.15.4 VM Entries that Return from SMM

The SMM-transfer monitor (STM) returns from SMM using a VM entry with the “entry to SMM” VM-entry control clear. VM entries that return from SMM reverse the effects of an SMM VM exit (see Section 34.15.2).

VM entries that return from SMM may differ from other VM entries in that they do not necessarily enter VMX non-root operation. If the executive-VMCS pointer field in the current VMCS contains the VMXON pointer, the logical processor remains in VMX root operation after VM entry.

For differences between VM entries that return from SMM and other VM entries see Sections 34.15.4.1 through 34.15.4.10.

#### 34.15.4.1 Checks on the Executive-VMCS Pointer Field

VM entries that return from SMM perform the following checks on the executive-VMCS pointer field in the current VMCS:

- Bits 11:0 must be 0.
- The pointer must not set any bits beyond the processor’s physical-address width.<sup>1,2</sup>
- The 32 bits located in memory referenced by the physical address in the pointer must contain the processor’s VMCS revision identifier (see Section 24.2).

The checks above are performed before the checks described in Section 34.15.4.2 and before any of the following checks:

- If the “deactivate dual-monitor treatment” VM-entry control is 0 and the executive-VMCS pointer field does not contain the VMXON pointer, the launch state of the executive VMCS (the VMCS referenced by the executive-VMCS pointer field) must be launched (see Section 24.11.3).
- If the “deactivate dual-monitor treatment” VM-entry control is 1, the executive-VMCS pointer field must contain the VMXON pointer (see Section 34.15.7).<sup>3</sup>

#### 34.15.4.2 Checks on VM-Execution Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-execution control fields specified in Section 26.2.1.1. They do not apply the checks to the current VMCS. Instead, VM-entry behavior depends on whether the executive-VMCS pointer field contains the VMXON pointer:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are not performed at all.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the checks are performed on the VM-execution control fields in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS). These checks are performed after checking the executive-VMCS pointer field itself (for proper alignment).

Other VM entries ensure that, if “activate VMX-preemption timer” VM-execution control is 0, the “save VMX-preemption timer value” VM-exit control is also 0. This check is not performed by VM entries that return from SMM.

#### 34.15.4.3 Checks on VM-Entry Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-entry control fields specified in Section 26.2.1.3.

- 
1. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
  2. If IA32\_VMX\_BASIC[48] is read as 1, this pointer must not set any bits in the range 63:32; see Appendix A.1.
  3. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

Specifically, if the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the VM-entry interruption-information field must not indicate injection of a pending MTF VM exit (see Section 26.5.2). Specifically, the following cannot all be true for that field:

- the valid bit (bit 31) is 1
- the interruption type (bits 10:8) is 7 (other event); and
- the vector (bits 7:0) is 0 (pending MTF VM exit).

#### 34.15.4.4 Checks on the Guest State Area

Section 26.3.1 specifies checks performed on fields in the guest-state area of the VMCS. Some of these checks are conditioned on the settings of certain VM-execution controls (e.g., “virtual NMIs” or “unrestricted guest”). VM entries that return from SMM modify these checks based on whether the executive-VMCS pointer field contains the VMXON pointer:<sup>1</sup>

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are performed as all relevant VM-execution controls were 0. (As a result, some checks may not be performed at all.)
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), this check is performed based on the settings of the VM-execution controls in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS).

For VM entries that return from SMM, the activity-state field must not indicate the wait-for-SIPI state if the executive-VMCS pointer field contains the VMXON pointer (the VM entry is to VMX root operation).

#### 34.15.4.5 Loading Guest State

VM entries that return from SMM load the SMBASE register from the SMBASE field.

VM entries that return from SMM invalidate linear mappings and combined mappings associated with all VPIDs. Combined mappings are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 28.3). (Ordinary VM entries are required to perform such invalidation only for VPID 0000H and are not required to do even that if the “enable VPID” VM-execution control is 1; see Section 26.3.2.5.)

#### 34.15.4.6 VMX-Preemption Timer

A VM entry that returns from SMM activates the VMX-preemption timer only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation) and the “activate VMX-preemption timer” VM-execution control is 1 in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field). In this case, VM entry starts the VMX-preemption timer with the value in the VMX-preemption timer-value field in the current VMCS.

#### 34.15.4.7 Updating the Current-VMCS and SMM-Transfer VMCS Pointers

Successful VM entries (returning from SMM) load the SMM-transfer VMCS pointer with the current-VMCS pointer. Following this, they load the current-VMCS pointer from a field in the current VMCS:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the current-VMCS pointer is loaded from the VMCS-link pointer field.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the current-VMCS pointer is loaded with the value of the executive-VMCS pointer field.

If the VM entry successfully enters VMX non-root operation, the VM-execution controls in effect after the VM entry are those from the new current VMCS. This includes any structures external to the VMCS referenced by VM-execution control fields.

---

1. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

The updating of these VMCS pointers occurs before event injection. Event injection is determined, however, by the VM-entry control fields in the VMCS that was current when the VM entry commenced.

#### 34.15.4.8 VM Exits Induced by VM Entry

Section 26.5.1.2 describes how the event-delivery process invoked by event injection may lead to a VM exit. Section 26.6.3 to Section 26.6.7 describe other situations that may cause a VM exit to occur immediately after a VM entry.

Whether these VM exits occur is determined by the VM-execution control fields in the current VMCS. For VM entries that return from SMM, they can occur only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation).

In this case, determination is based on the VM-execution control fields in the VMCS that is current after the VM entry. This is the VMCS referenced by the value of the executive-VMCS pointer field at the time of the VM entry (see Section 34.15.4.7). This VMCS also controls the delivery of such VM exits. Thus, VM exits induced by a VM entry returning from SMM are to the executive monitor and not to the STM.

#### 34.15.4.9 SMI Blocking

VM entries that return from SMM determine the blocking of system-management interrupts (SMIs) as follows:

- If the “deactivate dual-monitor treatment” VM-entry control is 0, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.
- If the “deactivate dual-monitor treatment” VM-entry control is 1, the blocking of SMIs depends on whether the logical processor is in SMX operation:<sup>1</sup>
  - If the logical processor is in SMX operation, SMIs are blocked after VM entry.
  - If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

VM entries that return from SMM and that do not deactivate the dual-monitor treatment may leave SMIs blocked. This feature exists to allow the STM to invoke functionality outside of SMM without unblocking SMIs.

#### 34.15.4.10 Failures of VM Entries That Return from SMM

Section 26.7 describes the treatment of VM entries that fail during or after loading guest state. Such failures record information in the VM-exit information fields and load processor state as would be done on a VM exit. The VMCS used is the one that was current before the VM entry commenced. Control is thus transferred to the STM and the logical processor remains in SMM.

### 34.15.5 Enabling the Dual-Monitor Treatment

Code and data for the SMM-transfer monitor (STM) reside in a region of SMRAM called the **monitor segment** (MSEG). Code running in SMM determines the location of MSEG and establishes its content. This code is also responsible for enabling the dual-monitor treatment.

SMM code enables the dual-monitor treatment and specifies the location of MSEG by writing to the IA32\_SMM\_MONITOR\_CTL MSR (index 9BH). The MSR has the following format:

- Bit 0 is the register’s valid bit. The STM may be invoked using VMCALL only if this bit is 1. Because VMCALL is used to activate the dual-monitor treatment (see Section 34.15.6), the dual-monitor treatment cannot be activated if the bit is 0. This bit is cleared when the logical processor is reset.
- Bit 1 is reserved.

---

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D*.

- Bit 2 determines whether executions of VMXOFF unblock SMIs under the default treatment of SMIs and SMM. Executions of VMXOFF unblock SMIs unless bit 2 is 1 (the value of bit 0 is irrelevant). See Section 34.14.4. Certain leaf functions of the GETSEC instruction clear this bit (see Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D*).
- Bits 11:3 are reserved.
- Bits 31:12 contain a value that, when shifted left 12 bits, is the physical address of MSEG (the MSEG base address).
- Bits 63:32 are reserved.

The following items detail use of this MSR:

- The IA32\_SMM\_MONITOR\_CTL MSR is supported only on processors that support the dual-monitor treatment.<sup>1</sup> On other processors, accesses to the MSR using RDMSR or WRMSR generate a general-protection fault (#GP(0)).
- A write to the IA32\_SMM\_MONITOR\_CTL MSR using WRMSR generates a general-protection fault (#GP(0)) if executed outside of SMM or if an attempt is made to set any reserved bit. An attempt to write to the IA32\_SMM\_MONITOR\_CTL MSR fails if made as part of a VM exit that does not end in SMM or part of a VM entry that does not begin in SMM.
- Reads from the IA32\_SMM\_MONITOR\_CTL MSR using RDMSR are allowed any time RDMSR is allowed. The MSR may be read as part of any VM exit.
- The dual-monitor treatment can be activated only if the valid bit in the MSR is set to 1.

The 32 bytes located at the MSEG base address are called the **MSEG header**. The format of the MSEG header is given in Table 34-10 (each field is 32 bits).

**Table 34-10. Format of MSEG Header**

Byte Offset	Field
0	MSEG-header revision identifier
4	SMM-transfer monitor features
8	GDTR limit
12	GDTR base offset
16	CS selector
20	EIP offset
24	ESP offset
28	CR3 offset

To ensure proper behavior in VMX operation, software should maintain the MSEG header in writeback cacheable memory. Future implementations may allow or require a different memory type.<sup>2</sup> Software should consult the VMX capability MSR IA32\_VMX\_BASIC (see Appendix A.1).

SMM code should enable the dual-monitor treatment (by setting the valid bit in IA32\_SMM\_MONITOR\_CTL MSR) only after establishing the content of the MSEG header as follows:

1. Software should consult the VMX capability MSR IA32\_VMX\_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.
2. Alternatively, software may map the MSEG header with the UC memory type; this may be necessary, depending on how memory is organized. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32\_VMX\_BASIC with exceptions noted in Appendix A.1.



- Bytes 3:0 contain the **MSEG revision identifier**. Different processors may use different MSEG revision identifiers. These identifiers enable software to avoid using an MSEG header formatted for one processor on a processor that uses a different format. Software can discover the MSEG revision identifier that a processor uses by reading the VMX capability MSR IA32\_VMX\_MISC (see Appendix A.6).
- Bytes 7:4 contain the **SMM-transfer monitor features field**. Bits 31:1 of this field are reserved and must be zero. Bit 0 of the field is the **IA-32e mode SMM feature bit**. It indicates whether the logical processor will be in IA-32e mode after the STM is activated (see Section 34.15.6).
- Bytes 31:8 contain fields that determine how processor state is loaded when the STM is activated (see Section 34.15.6.5). SMM code should establish these fields so that activating of the STM invokes the STM's initialization code.

### 34.15.6 Activating the Dual-Monitor Treatment

The dual-monitor treatment may be enabled by SMM code as described in Section 34.15.5. The dual-monitor treatment is activated only if it is enabled and only by the executive monitor. The executive monitor activates the dual-monitor treatment by executing VMCALL in VMX root operation.

When VMCALL activates the dual-monitor treatment, it causes an SMM VM exit. Differences between this SMM VM exit and other SMM VM exits are discussed in Sections 34.15.6.1 through 34.15.6.6. See also “VMCALL—Call to VM Monitor” in Chapter 30.

#### 34.15.6.1 Initial Checks

An execution of VMCALL attempts to activate the dual-monitor treatment if (1) the processor supports the dual-monitor treatment;<sup>1</sup> (2) the logical processor is in VMX root operation; (3) the logical processor is outside SMM and the valid bit is set in the IA32\_SMM\_MONITOR\_CTL MSR; (4) the logical processor is not in virtual-8086 mode and not in compatibility mode; (5) CPL = 0; and (6) the dual-monitor treatment is not active.

Such an execution of VMCALL begins with some initial checks. These checks are performed before updating the current-VMCS pointer and the executive-VMCS pointer field (see Section 34.15.2.2).

The VMCS that manages SMM VM exit caused by this VMCALL is the current VMCS established by the executive monitor. The VMCALL performs the following checks on the current VMCS in the order indicated:

1. There must be a current VMCS pointer.
2. The launch state of the current VMCS must be clear.
3. Reserved bits in the VM-exit controls in the current VMCS must be set properly. Software may consult the VMX capability MSR IA32\_VMX\_EXIT\_CTLS to determine the proper settings (see Appendix A.4).

If any of these checks fail, subsequent checks are skipped and VMCALL fails. If all these checks succeed, the logical processor uses the IA32\_SMM\_MONITOR\_CTL MSR to determine the base address of MSEG. The following checks are performed in the order indicated:

1. The logical processor reads the 32 bits at the base of MSEG and compares them to the processor's MSEG revision identifier.
2. The logical processor reads the SMM-transfer monitor features field:
  - Bit 0 of the field is the IA-32e mode SMM feature bit, and it indicates whether the logical processor will be in IA-32e mode after the SMM-transfer monitor (STM) is activated.
    - If the VMCALL is executed on a processor that does not support Intel 64 architecture, the IA-32e mode SMM feature bit must be 0.
    - If the VMCALL is executed in 64-bit mode, the IA-32e mode SMM feature bit must be 1.
  - Bits 31:1 of this field are currently reserved and must be zero.

If any of these checks fail, subsequent checks are skipped and the VMCALL fails.

- 
1. Software should consult the VMX capability MSR IA32\_VMX\_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.



### 34.15.6.2 Updating the Current-VMCS and Executive-VMCS Pointers

Before performing the steps in Section 34.15.2.2, SMM VM exits that activate the dual-monitor treatment begin by loading the SMM-transfer VMCS pointer with the value of the current-VMCS pointer.

### 34.15.6.3 Saving Guest State

As noted in Section 34.15.2.4, SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area. While this is true also for SMM VM exits that activate the dual-monitor treatment, the VMCS used for those VM exits exists outside SMRAM.

The SMM-transfer monitor (STM) can also discover the current value of the SMBASE register by using the RDMSR instruction to read the IA32\_SMBASE MSR (MSR address 9EH). The following items detail use of this MSR:

- The MSR is supported only if IA32\_VMX\_MISC[15] = 1 (see Appendix A.6).
- A write to the IA32\_SMBASE MSR using WRMSR generates a general-protection fault (#GP(0)). An attempt to write to the IA32\_SMBASE MSR fails if made as part of a VM exit or part of a VM entry.
- A read from the IA32\_SMBASE MSR using RDMSR generates a general-protection fault (#GP(0)) if executed outside of SMM. An attempt to read from the IA32\_SMBASE MSR fails if made as part of a VM exit that does not end in SMM.

### 34.15.6.4 Saving MSRs

The VM-exit MSR-store area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are saved into that area.

### 34.15.6.5 Loading Host State

The VMCS that is current during an SMM VM exit that activates the dual-monitor treatment was established by the executive monitor. It does not contain the VM-exit controls and host state required to initialize the STM. For this reason, such SMM VM exits do not load processor state as described in Section 27.5. Instead, state is set to fixed values or loaded based on the content of the MSEG header (see Table 34-10):

- CR0 is set to as follows:
  - PG, NE, ET, MP, and PE are all set to 1.
  - CD and NW are left unchanged.
  - All other bits are cleared to 0.
- CR3 is set as follows:
  - Bits 63:32 are cleared on processors that support IA-32e mode.
  - Bits 31:12 are set to bits 31:12 of the sum of the MSEG base address and the CR3-offset field in the MSEG header.
  - Bits 11:5 and bits 2:0 are cleared (the corresponding bits in the CR3-offset field in the MSEG header are ignored).
  - Bits 4:3 are set to bits 4:3 of the CR3-offset field in the MSEG header.
- CR4 is set as follows:
  - MCE, PGE, and PCIDE are cleared.
  - PAE is set to the value of the IA-32e mode SMM feature bit.
  - If the IA-32e mode SMM feature bit is clear, PSE is set to 1 if supported by the processor; if the bit is set, PSE is cleared.
  - All other bits are unchanged.
- DR7 is set to 400H.
- The IA32\_DEBUGCTL MSR is cleared to 00000000\_00000000H.

- The registers CS, SS, DS, ES, FS, and GS are loaded as follows:
  - All registers are usable.
  - CS.selector is loaded from the corresponding field in the MSEG header (the high 16 bits are ignored), with bits 2:0 cleared to 0. If the result is 0000H, CS.selector is set to 0008H.
  - The selectors for SS, DS, ES, FS, and GS are set to CS.selector+0008H. If the result is 0000H (if the CS selector was FFF8H), these selectors are instead set to 0008H.
  - The base addresses of all registers are cleared to zero.
  - The segment limits for all registers are set to FFFFFFFFH.
  - The AR bytes for the registers are set as follows:
    - CS.Type is set to 11 (execute/read, accessed, non-conforming code segment).
    - For SS, DS, ES, FS, and GS, the Type is set to 3 (read/write, accessed, expand-up data segment).
    - The S bits for all registers are set to 1.
    - The DPL for each register is set to 0.
    - The P bits for all registers are set to 1.
    - On processors that support Intel 64 architecture, CS.L is loaded with the value of the IA-32e mode SMM feature bit.
    - CS.D is loaded with the inverse of the value of the IA-32e mode SMM feature bit.
    - For each of SS, DS, ES, FS, and GS, the D/B bit is set to 1.
    - The G bits for all registers are set to 1.
- LDTR is unusable. The LDTR selector is cleared to 0000H, and the register is otherwise undefined (although the base address is always canonical)
- GDTR.base is set to the sum of the MSEG base address and the GDTR base-offset field in the MSEG header (bits 63:32 are always cleared on processors that support IA-32e mode). GDTR.limit is set to the corresponding field in the MSEG header (the high 16 bits are ignored).
- IDTR.base is unchanged. IDTR.limit is cleared to 0000H.
- RIP is set to the sum of the MSEG base address and the value of the RIP-offset field in the MSEG header (bits 63:32 are always cleared on logical processors that support IA-32e mode).
- RSP is set to the sum of the MSEG base address and the value of the RSP-offset field in the MSEG header (bits 63:32 are always cleared on logical processor that supports IA-32e mode).
- RFLAGS is cleared, except bit 1, which is always set.
- The logical processor is left in the active state.
- Event blocking after the SMM VM exit is as follows:
  - There is no blocking by STI or by MOV SS.
  - There is blocking by non-maskable interrupts (NMIs) and by SMIs.
- There are no pending debug exceptions after the SMM VM exit.
- For processors that support IA-32e mode, the IA32\_EFER MSR is modified so that LME and LMA both contain the value of the IA-32e mode SMM feature bit.

If any of CR3[63:5], CR4.PAE, CR4.PSE, or IA32\_EFER.LMA is changing, the TLBs are updated so that, after VM exit, the logical processor does not use translations that were cached before the transition. This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if IA32\_EFER.LMA was 1 before and after the transition).

### 34.15.6.6 Loading MSRs

The VM-exit MSR-load area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are loaded from that area.

### 34.15.7 Deactivating the Dual-Monitor Treatment

The SMM-transfer monitor may deactivate the dual-monitor treatment and return the processor to default treatment of SMIs and SMM (see Section 34.14). It does this by executing a VM entry with the “deactivate dual-monitor treatment” VM-entry control set to 1.

As noted in Section 26.2.1.3 and Section 34.15.4.1, an attempt to deactivate the dual-monitor treatment fails in the following situations: (1) the processor is not in SMM; (2) the “entry to SMM” VM-entry control is 1; or (3) the executive-VMCS pointer does not contain the VMXON pointer (the VM entry is to VMX non-root operation).

As noted in Section 34.15.4.9, VM entries that deactivate the dual-monitor treatment ignore the SMI bit in the interruptibility-state field of the guest-state area. Instead, the blocking of SMIs following such a VM entry depends on whether the logical processor is in SMX operation:<sup>1</sup>

- If the logical processor is in SMX operation, SMIs are blocked after VM entry. SMIs may later be unblocked by the VMXOFF instruction (see Section 34.14.4) or by certain leaf functions of the GETSEC instruction (see Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D*).
- If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

## 34.16 SMI AND PROCESSOR EXTENDED STATE MANAGEMENT

On processors that support processor extended states using XSAVE/XRSTOR (see Chapter 13, “Managing State Using the XSAVE Feature Set” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*), the processor does not save any XSAVE/XRSTOR related state on an SMI. It is the responsibility of the SMI handler code to properly preserve the state information (including CR4.OSXSAVE, XCR0, and possibly processor extended states using XSAVE/XRSTOR). Therefore, the SMI handler must follow the rules described in Chapter 13, “Managing State Using the XSAVE Feature Set” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.

## 34.17 MODEL-SPECIFIC SYSTEM MANAGEMENT ENHANCEMENT

This section describes enhancement of system management features that apply only to the 4th generation Intel Core processors. These features are model-specific. BIOS and SMM handler must use CPUID to enumerate DisplayFamily\_DisplayModel signature when programming with these interfaces.

### 34.17.1 SMM Handler Code Access Control

The BIOS may choose to restrict the address ranges of code that SMM handler executes. When SMM handler code execution check is enabled, an attempt by the SMM handler to execute outside the ranges specified by SMRR (see Section 34.4.2.1) will cause the assertion of an unrecoverable machine check exception (MCE).

The interface to enable SMM handler code access check resides in a per-package scope model-specific register MSR\_SMM\_FEATURE\_CONTROL at address 4E0H. An attempt to access MSR\_SMM\_FEATURE\_CONTROL outside of SMM will cause a #GP. Writes to MSR\_SMM\_FEATURE\_CONTROL is further protected by configuration interface of MSR\_SMM\_MCA\_CAP at address 17DH.

Details of the interface of MSR\_SMM\_FEATURE\_CONTROL and MSR\_SMM\_MCA\_CAP are described in Table 2-28 in Chapter 2, “Model-Specific Registers (MSRs)” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4*.

---

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

### 34.17.2 SMI Delivery Delay Reporting

Entry into the system management mode occurs at instruction boundary. In situations where a logical processor is executing an instruction involving a long flow of internal operations, servicing an SMI by that logical processor will be delayed. Delayed servicing of SMI of each logical processor due to executing long flows of internal operation in a physical processor can be queried via a package-scope register MSR\_SMM\_DELAYED at address 4E2H.

The interface to enable reporting of SMI delivery delay due to long internal flows resides in a per-package scope model-specific register MSR\_SMM\_DELAYED. An attempt to access MSR\_SMM\_DELAYED outside of SMM will cause a #GP. Availability to MSR\_SMM\_DELAYED is protected by configuration interface of MSR\_SMM\_MCA\_CAP at address 17DH.

Details of the interface of MSR\_SMM\_DELAYED and MSR\_SMM\_MCA\_CAP are described in Table 2-28 in Chapter 2, "Model-Specific Registers (MSRs)" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*.

### 34.17.3 Blocked SMI Reporting

A logical processor may have entered into a state and blocked from servicing other interrupts (including SMI). Logical processors in a physical processor that are blocked in serving SMI can be queried in a package-scope register MSR\_SMM\_BLOCKED at address 4E3H. An attempt to access MSR\_SMM\_BLOCKED outside of SMM will cause a #GP.

Details of the interface of MSR\_SMM\_BLOCKED is described in Table 2-28 in Chapter 2, "Model-Specific Registers (MSRs)" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*.

## 17. Updates to Chapter 41, Volume 3D

Change bars show changes to Chapter 41 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

-----

Change to chapter: Update to Section 41.2.2 "Intel SGX Launch Control Configuration".

# CHAPTER 41

## INTEL® SGX INTERACTIONS WITH IA32 AND INTEL® 64 ARCHITECTURE

---

Intel® SGX provides Intel® Architecture with a collection of enclave instructions for creating protected execution environments on processors supporting IA32 and Intel® 64 architectures. These Intel SGX instructions are designed to work with legacy software and the various IA32 and Intel 64 modes of operation.

### 41.1 INTEL® SGX AVAILABILITY IN VARIOUS PROCESSOR MODES

The Intel SGX extensions (see Table 36-1) are available only when the processor is executing in protected mode of operation. Additionally, the extensions are not available in System Management Mode (SMM) of operation or in Virtual 8086 (VM86) mode of operation. Finally, all leaf functions of ENCLU and ENCLS require CR0.PG enabled.

The exact details of exceptions resulting from illegal modes and their priority are listed in the reference pages of ENCLS and ENCLU.

### 41.2 IA32\_FEATURE\_CONTROL

IA32\_FEATURE\_CONTROL MSR provides two new bits related to two aspects of Intel SGX: using the instruction extensions and launch control configuration.

#### 41.2.1 Availability of Intel SGX

IA32\_FEATURE\_CONTROL[bit 18] allows BIOS to control the availability of Intel SGX extensions. For Intel SGX extensions to be available on a logical processor, bit 18 in the IA32\_FEATURE\_CONTROL MSR on that logical processor must be set, and IA32\_FEATURE\_CONTROL MSR on that logical processor must be locked (bit 0 must be set). See Section 36.7.1 for additional details. OS is expected to examine the value of bit 18 prior to enabling Intel SGX on the thread, as the settings of bit 18 is not reflected by CPUID.

#### 41.2.2 Intel SGX Launch Control Configuration

The IA32\_SGXLEPUBKEYHASHn MSRs used to configure authorized launch enclaves' MRSIGNER digest value. They are present on logical processors that support the collection of SGX1 leaf functions (i.e. CPUID.(EAX=12H, ECX=00H):EAX[0] = 1) and that CPUID.(EAX=07H, ECX=00H):ECX[30] = 1. IA32\_FEATURE\_CONTROL[bit 17] allows to BIOS to enable write access to these MSRs. If IA32\_FEATURE\_CONTROL.LE\_WR (bit 17) is set to 1 and IA32\_FEATURE\_CONTROL is locked on that logical processor, IA32\_SGXLEPUBKEYHASH MSRs on that logical processor are writeable. If this bit 17 is not set or IA32\_FEATURE\_CONTROL is not locked, IA32\_SGXLEPUBKEYHASH MSRs are read only. See Section 38.1.4 for additional details.

### 41.3 INTERACTIONS WITH SEGMENTATION

#### 41.3.1 Scope of Interaction

Intel SGX extensions are available only when the processor is executing in a protected mode operation (see Section 41.1 for Intel SGX availability in various processor modes). Enclaves abide by all the segmentation policies set up by the OS, but they can be more restrictive than the OS.

Intel SGX interacts with segmentation at two levels:

- The Intel SGX instruction (see the enclave instruction in Table 36-1).

- While executing inside an enclave (legacy instructions and enclave instructions permitted inside an enclave).

### 41.3.2 Interactions of Intel® SGX Instructions with Segment, Operand, and Addressing Prefixes

All the memory operands used by the Intel SGX instructions are interpreted as offsets within the data segment (DS). The segment-override prefix on Intel SGX instructions is ignored.

Operand size is fixed for each enclave instruction. The operand-size prefix is reserved, and results in a #UD exception if used.

All address sizes are determined by the operating mode of the processor. The address-size prefix is ignored. This implies that while operating in 64-bit mode of operation, the address size is always 64 bits, and while operating in 32-bit mode of operation, the address size is always 32 bits. Additionally, when operating in 16-bit addressing, memory operands used by enclave instructions use 32 bit addressing; the value of CS.D is ignored.

### 41.3.3 Interaction of Intel® SGX Instructions with Segmentation

All leaf functions of ENCLU and ENCLS instructions require that the DS segment be usable, and be an expand-up segment. Failing this check results in generation of a #GP(0) exception.

The Intel SGX leaf functions used for entering the enclave (ENCLU[EENTER] and ENCLU[ERESUME]) operate as follows:

- All usable segment registers except for FS and GS have a zero base.
- The contents of the FS/GS segment registers (including the hidden portion) is saved in the processor.
- New FS and GS values compatible with enclave security are loaded from the TCS
- The linear ranges and access rights available under the newly-loaded FS and GS must abide to OS policies by ensuring they are subsets of the linear-address range and access rights available for the DS segment.
- The CS segment mode (64-bit, compatible, or 32 bit modes) must be consistent with the segment mode for which the enclave was created, as indicated by the SECS.ATTRIBUTES.MODE64 bit, and that the CPL of the logical processor is 3

An exit from the enclave either via ENCLU[EEXIT] or via an AEX restores the saved values of FS/GS segment registers.

### 41.3.4 Interactions of Enclave Execution with Segmentation

During the course of execution, enclave code abides by all segmentation policies as dictated by IA32 and Intel 64 Architectures, and generates appropriate exceptions on violations.

Additionally, any attempt by software executing inside an enclave to modify the processor's segmentation state (e.g. via MOV seg register, POP seg register, LDS, far jump, etc; excluding WRFSBASE/WRGSBASE) results in the generation of a #UD. See Section 38.6.1 for more information.

Upon enclave entry via the EENTER leaf function, FS is loaded from the (TCS.OFSBASE + SECS.BASEADDR) and TCS.FSLIMIT fields and GS is loaded from the (TCS.OGSBASE + SECS.BASEADDR) and TCS.GSLIMIT fields.

Execution of WRFSBASE and WRGSBASE from inside a 64-bit enclave is allowed. The processor will save the new values into the current SSA frame on an asynchronous exit (AEX) and restore them back on enclave entry via ENCLU[ERESUME] instruction.

## 41.4 INTERACTIONS WITH PAGING

Intel SGX instructions are available only when the processor is executing in a protected mode of operation. Additionally, all Intel SGX leaf functions except for EDBG RD and EDBG WR are available only if paging is enabled. Any attempt to execute these leaf functions with paging disabled results in an invalid-opcode exception (#UD). As with

segmentation, enclaves abide by all the paging policies set up by the OS, but they can be more restrictive than the OS.

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the DS segment, and the linear addresses generated by combining these offsets with DS segment register are subject to paging-based access control if paging is enabled at the time of the execution of the leaf function.

Since the ENCLU[EENTER] and ENCLU[ERESUME] can only be executed when paging is enabled, and since paging cannot be disabled by software running inside an enclave (recall that enclaves always run with CPL = 3), enclave execution is always subject to paging-based access control. The Intel SGX access control itself is implemented as an extension to the existing paging modes. See Section 37.5 for details.

Execution of Intel SGX instructions may set accessed and dirty flags on accesses to EPC pages that do not fault even if the instruction later causes a fault for some other reason.

## 41.5 INTERACTIONS WITH VMX

Intel SGX functionality (including SGX1 and SGX2) can be made available to software running in either VMX root operation or VMX non-root operation, as long as the processor is using a legal mode of operation (see Section 41.1).

A VMM has the flexibility to configure a VMCS to permit a guest to use any subset of the ENCLS leaf functions. Availability of the ENCLU leaf functions in VMX non-root operation has the same requirement as ENCLU leaf functions outside of a virtualized environment.

Details of the VMCS control to allow VMM to configure support of Intel SGX in VMX non-root operation is described in Section 41.5.1

### 41.5.1 VMM Controls to Configure Guest Support of Intel® SGX

Intel SGX capabilities are primarily exposed to the software via the CPUID instruction. VMMs can virtualize CPUID instruction to expose/hide this capability to/from guests.

Some of Intel SGX resources are exposed/controlled via model-specific registers (see Section 36.7). VMMs can virtualize these MSRs for the guests using the MSR bitmaps referenced by pointers in the VMCS.

The VMM can partition the Enclave Page Cache, and assign various partitions to (a subset of) its guests via the usual memory-virtualization techniques such as paging or the extended page table mechanism (EPT).

The VMM can set the “enable ENCLS exiting” VM-execution controls to cause a VM exit when the ENCLS instruction is executed in VMX non-root operation. If the “enable ENCLS exiting” control is 0, all of the ENCLS leaf functions are permitted in VMX non-root operation. If the “enable ENCLS exiting” control is 1, execution of ENCLS leaf functions in VMX non-root operation is governed by consulting the bits in a new 64-bit VM-execution control field called the ENCLS-exiting bitmap (Each bit in the bitmap corresponds to an ENCLS leaf function with an EAX value that is identical to the bit’s position). When bits in the “ENCLS-exiting bitmap” are set, attempts to execute the corresponding ENCLS leaf functions in VMX non-root operation causes VM exits. The checking for these VM exits occurs immediately after checking that CPL = 0.

### 41.5.2 Interactions with the Extended Page Table Mechanism (EPT)

Intel SGX instructions are fully compatible with the extended page-table mechanism (EPT; see Section 28.2).

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the DS segment, and the linear addresses generated by combining these offsets with DS segment register are subject to paging and EPT. As with paging, enclaves abide by all the policies set up by the VMM.

The Intel SGX access control itself is implemented as an extension to paging and EPT, and may be more restrictive. See Section 41.4 for details of this extension.

An execution of an Intel SGX instruction may set accessed and dirty flags for EPT (when enabled; see Section 28.2.4) on accesses to EPC pages that do not fault or cause VM exits even if the instruction later causes a fault or VM exit for some other reason.



### 41.5.3 Interactions with APIC Virtualization

This section applies to Intel SGX in VMX non-root operation when the “virtualize APIC accesses” VM-execution control is 1.

A memory access by an enclave instruction that implicitly uses a cached physical address is never checked for overlap with the APIC-access page. Such accesses never cause APIC-access VM exits and are never redirected to the virtual-APIC page. Implicit memory accesses can only be made to the SECS, the TCS, or the SSA of an enclave (see Section 37.5.3.2).

An explicit Enclave Access (a linear memory access which is either from within an enclave into its ELRANGE, or an access by an Intel SGX instruction that is expected to be in the EPC) that overlaps with the APIC-access page causes a #PF exception (APIC page is expected to be outside of EPC).

Non-Enclave accesses made either by an Intel SGX instruction or by a logical processor inside an enclave to an address that without SGX would have caused redirection to the virtual-APIC page instead cause an APIC-access VM exit.

Other than implicit accesses made by Intel SGX instructions, guest-physical and physical accesses are not considered “enclave accesses”; consequently, such accesses result in undefined behavior if these accesses eventually reach EPC. This applies to any non-enclave physical accesses.

While a logical processor is executing inside an enclave, an attempt to execute an instruction outside of ELRANGE results in a #GP(0), even if the linear address would translate to a physical address that overlaps the APIC-access page.

## 41.6 INTEL® SGX INTERACTIONS WITH ARCHITECTURALLY-VISIBLE EVENTS

All architecturally visible vectored events (IA32 exceptions, interrupts, SMI, NMI, INIT, VM exit) can be detected while inside an enclave and will cause an asynchronous enclave exit if they are not blocked. Additionally, INT3, and the SignalTXTMsg[SENDER] (i.e. GETSEC[SENDER]’s rendezvous event message) events also cause asynchronous enclave exits. Note that SignalTXTMsg[SEXIT] (i.e. GETSEC[SEXIT]’s teardown message) does not cause an AEX.

On an AEX, information about the event causing the AEX is stored in the SSA (see Section 39.4 for details of AEX). The information stored in the SSA only describes the first event that triggered the AEX. If parsing/delivery of the first event results in detection of further events (e.g. VM exit, double fault, etc.), then the event information in the SSA is not updated to reflect these subsequently detected events.

## 41.7 INTERACTIONS WITH THE PROCESSOR EXTENDED STATE AND MISCELLANEOUS STATE

### 41.7.1 Requirements and Architecture Overview

Processor extended states are the ISA features that are enabled by the settings of CR4.OSXSAVE and the XCR0 register. Processor extended states are normally saved/restored by software via XSAVE/XRSTOR instructions. Details of discovery of processor extended states and management of these states are described in CHAPTER 13 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

Additionally, the following requirements apply to Intel SGX:

- On an AEX, the Intel SGX architecture must protect the processor extended state and miscellaneous state by saving them in the enclave’s state-save area (SSA), and clear the secrets from the processor extended state that is used by an enclave.
- Intel SGX architecture must verify that the SSA frame size is large enough to contain all the processor extended states and miscellaneous state used by the enclave.
- Intel SGX architecture must ensure that enclaves can only use processor extended state that is enabled by system software in XCR0.

- Enclave software should be able to discover only those processor extended state and miscellaneous state for which such protection is enabled.
- The processor extended states that are enabled inside the enclave must be approved by the enclave developer:
  - Certain processor extended state (e.g., Memory Protection Extensions, see Chapter 17, “Intel® MPX” of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*) modify the behavior of the legacy ISA software. If such features are enabled for enclaves that do not understand those features, then such a configuration could lead to a compromise of the enclave's security.
- The processor extended states that are enabled inside the enclave must form an integral part of the enclave's identity. This requirement has two implications:
  - Service providers may decide to assign different trust level to the same enclave depending on the ISA features the enclave is using.

To meet these requirements, the Intel SGX architecture defines a sub-field called X-Feature Request Mask (XFRM) in the ATTRIBUTES field of the SECS. On enclave creation (ENCLS[ECREATE] leaf function), the required SSA frame size is calculated by the processor from the list of enabled extended and miscellaneous states and verified against the actual SSA frame size defined by SECS.SSAFRAMESIZE.

On enclave entry, after verifying that XFRM is only enabling features that are already enabled in XCR0, the value in the XCR0 is saved internally by the processor, and is replaced by the XFRM. On enclave exit, the original value of XCR0 is restored. Consequently, while inside the enclave, the processor extended states enabled in XFRM are in enabled state, and those that are disabled in XFRM are in disabled state.

The entire ATTRIBUTES field, including the XFRM subfield is integral part of enclave's identity (i.e., its value is included in reports generated by ENCLU[EREPORT], and select bits from this field can be included in key-derivation for keys obtained via the ENCLU[GETKEY] leaf function).

Enclave developers can create their enclave to work with certain features and fallback to another code path in case those features aren't available (e.g. optimize for AVX and fallback to SSE). For this purpose Intel SGX provides the following fields in SIGSTRUCT: ATTRIBUTES, ATTRIBUTESMASK, MISCSELECT, and MISCMASK. EINIT ensures that the final SECS.ATTRIBUTES and SECS.MISCSELECT comply with the enclave developer's requirements as follows:  
 SIGSTRUCT.ATTRIBUTES & SIGSTRUCT.ATTRIBUTESMASK = SECS.ATTRIBUTES & SIGSTRUCT.ATTRIBUTESMASK  
 SIGSTRUCT.MISCSELECT & SIGSTRUCT.MISCMASK = SECS.MISCSELECT & SIGSTRUCT.MISCMASK.

On an asynchronous enclave exit, the processor extended states enabled by XFRM are saved in the current SSA frame, and overwritten by synthetic state (see Section 39.3 for the definition of the synthetic state). When the interrupted enclave is resumed via the ENCLU[ERESUME] leaf function, the saved state for processor extended states enabled by XFRM is restored.

## 41.7.2 Relevant Fields in Various Data Structures

### 41.7.2.1 SECS.ATTRIBUTES.XFRM

The ATTRIBUTES field of the SECS data structure (see Section 37.7) contains a sub-field called XSAVE-Feature Request Mask (XFRM). Software populates this field at the time of enclave creation according to the features that are enabled by the operating system and approved by the enclave developer.

Intel SGX architecture guarantees that during enclave execution, the processor extended state configuration of the processor is identical to what is required by the XFRM sub-field. All the processor extended states enabled in XFRM are saved on AEX from the enclave and restored on ERESUME.

The XFRM sub-field has the same layout as XCR0, and has consistency requirements that are similar to those for XCR0. Specifically, the consistency requirements on XFRM values depend on the processor implementation and the set of features enabled in CR4.

Legal values for SECS.ATTRIBUTES.XFRM conform to these requirements:

- XFRM[1:0] must be set to 0x3.
- If the processor does not support XSAVE, or if the system software has not enabled XSAVE, then XFRM[63:2] must be zero.
- If the processor does support XSAVE, XFRM must contain a value that would be legal if loaded into XCR0.

The various consistency requirements are enforced at different times in the enclave's life cycle, and the exact enforcement mechanisms are elaborated in Section 41.7.3 through Section 41.7.6.

On processors not supporting XSAVE, software should initialize XFRM to 0x3. On processors supporting XSAVE, software should initialize XFRM to be a subset of XCR0 that would be present at the time of enclave execution. Because bits 0 and 1 of XFRM must always be set, the use of Intel SGX requires that SSE be enabled (CR4.OSFXSR = 1).

### 41.7.2.2 SECS.SSAFRAMESIZE

The SSAFRAMESIZE field in the SECS data structure specifies the number of pages which software allocated<sup>1</sup> for each SSA frame, including both the GPRSGX area, MISC area, the XSAVE area (x87 and XMM states are stored in the latter area), and optionally padding between the MISC and XSAVE area. The GPRSGX area must hold all the general-purpose registers and additional Intel SGX specific information. The MISC area must hold the Miscellaneous state as specified by SECS.MISCSELECT, the XSAVE area holds the set of processor extended states specified by SECS.ATTRIBUTES.XFRM (see Section 37.9 for the layout of SSA and Section 41.7.3 for ECREATE's consistency checks). The SSA is always in non-compacted format.

If the processor does not support XSAVE, the XSAVE area will always be 576 bytes; a copy of XFRM (which will be set to 0x3) is saved at offset 512 on an AEX.

If the processor does support XSAVE, the length of the XSAVE area depends on SECS.ATTRIBUTES.XFRM. The length would be equal to what CPUID.(EAX=0DH, ECX= 0):EBX would return if XCR0 were set to XFRM. The following pseudo code illustrates how software can calculate this length using XFRM as the input parameter without modifying XCR0:

```
offset = 576;
size_last_x = 0;
For x=2 to 63
  IF (XFRM[x] != 0) Then
    tmp_offset = CPUID.(EAX=0DH, ECX= x):EBX[31:0];
    IF (tmp_offset >= offset + size_last_x) Then
      offset = tmp_offset;
      size_last_x = CPUID.(EAX=0DH, ECX= x):EAX[31:0];
    FI;
  FI;
EndFor
return (offset + size_last_x); (* compute_xsave_size(XFRM), see "ECREATE—Create an SECS page in the Enclave Page Cache"*)
```

Where the non-zero bits in XFRM are a subset of non-zero bit fields in XCR0.

The size of the MISC region depends on the setting of SECS.MISCSELECT and can be calculated using the layout information described in Section 37.9.2

### 41.7.2.3 XSAVE Area in SSA

The XSAVE area of an SSA frame begins at offset 0 of the frame.

### 41.7.2.4 MISC Area in SSA

The MISC area of an SSA frame is positioned immediately before the GPRSGX region.

### 41.7.2.5 SIGSTRUCT Fields

Intel SGX provides the flexibility for an enclave developer to choose the enclave's code path according to the features that are enabled on the platform (e.g. optimize for AVX and fallback to SSE). See Section 41.7.1 for details.

---

1. It is the responsibility of the enclave to actually allocate this memory.

SIGSTRUCT includes the following fields:

SIGSTRUCT.ATTRIBUTES, SIGSTRUCT.ATTRIBUTEMASK, SIGSTRUCT.MISCSELECT, SIGSTRUCT.MISCMASK.

#### 41.7.2.6 REPORT.ATTRIBUTES.XFRM and REPORT.MISCSELECT

The processor extended states and miscellaneous states that are enabled inside the enclave form an integral part of the enclave's identity and are therefore included in the enclave's report, as provided by the ENCLU[EREPORT] leaf function. The REPORT structure includes the enclave's XFRM and MISCSELECT configurations.

#### 41.7.2.7 KEYREQUEST

An enclave developer can specify which bits out of XFRM and MISCSELECT ENCLU[EGETKEY] should include in the derivation of the sealing key by specifying ATTRIBUTEMASK and MISCMASK in the KEYREQUEST structure.

### 41.7.3 Processor Extended States and ENCLS[ECREATE]

The ECREATE leaf function of the ENCLS instruction enforces a number of consistency checks described earlier. The execution of ENCLS[ECREATE] leaf function results in a #GP(0) in any of the following cases:

- SECS.ATTRIBUTES.XFRM[1:0] is not 3.
- The processor does not support XSAVE and any of the following is true:
  - SECS.ATTRIBUTES.XFRM[63:2] is not 0.
  - SECS.SSAFRAMESIZE is 0.
- The processor supports XSAVE and any of the following is true:
  - XSETBV would fault on an attempt to load XFRM into XCR0.
  - XFRM[63]=1.
  - The SSAFRAME is too small to hold required, enabled states (see Section 41.7.2.2).

### 41.7.4 Processor Extended States and ENCLU[EENTER]

#### 41.7.4.1 Fault Checking

The EENTER leaf function of the ENCLU instruction enforces a number of consistency requirements described earlier. The execution of the ENCLU[EENTER] leaf function results in a #GP(0) in any of the following cases:

- If CR4.OSFXSR=0.
- If The processor supports XSAVE and either of the following is true:
  - CR4.OSXSAVE=0 and SECS.ATTRIBUTES.XFRM is not 3.
  - (SECS.ATTRIBUTES.XFRM & XCR0) != SECS.ATTRIBUTES.XFRM

#### 41.7.4.2 State Loading

If ENCLU[EENTER] is successful, the current value of XCR0 is saved internally by the processor and replaced by SECS.ATTRIBUTES.XFRM.

## 41.7.5 Processor Extended States and AEX

### 41.7.5.1 State Saving

On an AEX, processor extended states are saved into the XSAVE area of the SSA frame in a compatible format with XSAVE that was executed with  $EDX:EAX = SECS.ATTRIBUTES.XFRM$ , with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if  $REX.W=1$ . The  $XSTATE\_BV$  part of the XSAVE header is saved with 0 for every bit that is 0 in XFRM. Other bits may be saved as 0 if the state saved is initialized.

Note that enclave entry ensures that if  $CR4.OSXSAVE$  is set to 0, then  $SECS.ATTRIBUTES.XFRM$  is set to 3. It should also be noted that it is not possible to enter an enclave with FXSAVE disabled.

### 41.7.5.2 State Synthesis

After saving the extended state, the processor restores XCR0 to the value it held at the time of the most recent enclave entry.

The state of features corresponding to bits set in XFRM is synthesized. In general, these states are initialized. Details of state synthesis on AEX are documented in Section 39.3.1.

## 41.7.6 Processor Extended States and ENCLU[ERESUME]

### 41.7.6.1 Fault Checking

The ERESUME leaf function of the ENCLU instruction enforces a number of consistency requirements described earlier. Specifically, the ENCLU[ERESUME] leaf function results in a #GP(0) in any of the following cases:

- $CR4.OSFXSR=0$ .
- The processor supports XSAVE and either of the following is true:
  - $CR4.OSXSAVE=0$  and  $SECS.ATTRIBUTES.XFRM$  is not 3.
  - $(SECS.ATTRIBUTES.XFRM \& XCR0) \neq SECS.ATTRIBUTES.XFRM$ .

A successful execution of ENCLU[ERESUME] loads state from the XSAVE area of the SSA frame in a fashion similar to that used by the XRSTOR instruction. Data in the XSAVE area that would cause the XRSTOR instruction to fault will cause the ENCLU[ERESUME] leaf function to fault. Examples include, but are not restricted to the following:

- A bit is set in the  $XSTATE\_BV$  field and clear in XFRM.
- The required bytes in the header are not clear.
- Loading data would set a reserved bit in MXCSR.

Any of these conditions will cause ERESUME to fault, even if  $CR4.OSXSAVE=0$ .

### 41.7.6.2 State Loading

If ENCLU[ERESUME] is successful, the current value of XCR0 is saved internally by the processor and replaced by  $SECS.ATTRIBUTES.XFRM$ .

State is loaded from the XSAVE area of the SSA frame as if the XRSTOR instruction were executed with  $XCR0=XFRM$ ,  $EDX:EAX = XFRM$ , with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if  $REX.W=1$ .

ENCLU[ERESUME] ensures that a subsequent execution of XSAVEOPT inside the enclave will operate properly (e.g., by marking all state as modified).

## 41.7.7 Processor Extended States and ENCLU[EEXIT]

The ENCLU[EEXIT] leaf function does not perform any X-feature specific consistency checks, nor performs any state synthesis. It is the responsibility of enclave software to clear any sensitive data from the registers before

executing EEXIT. However, successful execution of the ENCLU[EEXIT] leaf function restores XCR0 to the value it held at the time of the most recent enclave entry.

### 41.7.8 Processor Extended States and ENCLU[EREPORT]

The ENCLU[EREPORT] leaf function creates the MAC-protected REPORT structure that reports on the enclave's identity. ENCLU[EREPORT] includes in the report the values of SECS.ATTRIBUTES.XFRM and SECS.MISCSELECT.

### 41.7.9 Processor Extended States and ENCLU[EGETKEY]

The ENCLU[EGETKEY] leaf function returns a cryptographic key based on the information provided by the KEYREQUEST structure. Intel SGX provides the means for isolation between different operating conditions by allowing an enclave developer to select which bits out of XFRM and MISCSELECT need to be included in the derivation of the keys.

## 41.8 INTERACTIONS WITH SMM

### 41.8.1 Availability of Intel® SGX instructions in SMM

Enclave instructions are not available in SMM, and any attempt to execute ENCLS or ENCLU instructions inside SMM results in an invalid-opcode exception (#UD).

### 41.8.2 SMI while Inside an Enclave

If the logical processor executing inside an enclave receives an SMI, the logical processor exits the enclave asynchronously. The response to an SMI received while executing inside an enclave depends on whether the dual-monitor treatment is enabled. For detailed discussion of transfer to SMM, see Chapter 34, "System Management Mode" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

If the logical processor executing inside an enclave receives an SMI when dual-monitor treatment is not enabled, the logical processor exits the enclave asynchronously, and transfers the control to the SMM handler. In addition to saving the synthetic architectural state to the SMRAM State Save Map (SSM), the logical processor also sets the "Enclave Interruption" bit in the SMRAM SSM (bit position 1 in SMRAM field at offset 7EE0H).

If the logical processor executing inside an enclave receives an SMI when dual-monitor treatment is enabled, the logical processor exits the enclave asynchronously, and transfers the control to the SMM monitor via SMM VM exit. The SMM VM exit sets the "Enclave Interruption" bit in the Exit Reason (see Table 41-1) and in the Guest Interruptibility State field (see Table 41-2) of the SMM VMCS.

### 41.8.3 SMRAM Synthetic State of AEX Triggered by SMI

All processor registers saved in the SMRAM have the same synthetic values listed in Section 39.3. Additional SMRAM fields that are treated specially on SMI are:

**Table 41-1. SMRAM Synthetic States on Asynchronous Enclave Exit**

Position	Field	Value	Writable
SMRAM Offset 07EE0H.Bit 1	ENCLAVE_INTERRUPTION	Set to 1 if exit occurred in enclave mode	No

## 41.9 INTERACTIONS OF INIT, SIPI, AND WAIT-FOR-SIPI WITH INTEL® SGX

INIT received inside an enclave, while the logical processor is not in VMX operation, causes the logical processor to exit the enclave asynchronously. After the AEX, the processor's architectural state is initialized to "Power-on" state (Table 9.1 in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*). If the logical processor is BSP, then it proceeds to execute the BIOS initialization code. If the logical processor is an AP, it enters wait-for-SIPI state.

INIT received inside an enclave, while the logical processor (LP) is in VMX root operation, follows regular Intel Architecture behavior and is blocked.

INIT received inside an enclave, while the logical processor is in VMX non-root operation, causes an AEX. Subsequent to the AEX, the INIT causes a VM exit with the Enclave Interruption bit in the exit-reason field in the VMCS.

A processor cannot be inside an enclave in the wait-for-SIPI state. Consequently, a SIPI received while inside an enclave is lost.

Intel SGX does not change the behavior of the processor in the wait-for-SIPI state.

The SGX-related processor states after INIT-SIPI-SIPI is as follows:

- EPC Settings: Unchanged
- EPCM: Unchanged
- CPUID.LEAF\_12H.\*: Unchanged
- ENCLAVE\_MODE: 0 (LP exits enclave asynchronously)
- MEE state: Unchanged

Software should be aware that following INIT-SIPI-SIPI, the EPC might contain valid pages and should take appropriate measures such as initialize the EPC with the EREMOVE leaf function.

## 41.10 INTERACTIONS WITH DMA

DMA is not allowed to access any Processor Reserved Memory.

## 41.11 INTERACTIONS WITH TXT

### 41.11.1 Enclaves Created Prior to Execution of GETSEC

Enclaves which have been created before the GETSEC[SENDER] leaf function are available for execution after the successful completion of GETSEC[SENDER] and the corresponding SINIT ACM. Actions that a TXT Launched Environment performs in preparation to execute code in the Launched Environment, also applies to enclave code that would run after GETSEC[SENDER].

### 41.11.2 Interaction of GETSEC with Intel® SGX

All leaf functions of the GETSEC instruction are illegal inside an enclave, and results in an invalid-opcode exception (#UD).

Responding Logical Processors (RLP) which are executing inside an enclave at the time a GETSEC[SENDER] event occurs perform an AEX from the enclave and then enter the Wait-for-SIPI state.

RLP executing inside an enclave at the time of GETSEC[SEXIT], behave as defined for GETSEC[SEXIT]-that is, the RLPs pause during execution of SEXIT and resume after the completion of SEXIT.

The execution of a TXT launch does not affect Intel SGX configuration or security parameters.



### 41.11.3 Interactions with Authenticated Code Modules (ACMs)

Intel SGX only allows launching ACMs with an Intel SGX SVN that is at the same level or higher than the expected Intel SGX SVN. The expected Intel SGX SVN is specified by BIOS and locked down by the processor on the first successful execution of an Intel SGX instruction that doesn't return an error code. Intel SGX provides interfaces for system software to discover whether a non-faulting Intel SGX instruction has been executed, and evaluate the suitability of the Intel SGX SVN value of any ACM that is expected to be launched by the OS or the VMM.

These interfaces are provided through a read-only MSR called the IA32\_SGX\_SVN\_STATUS MSR (MSR address 500h). The IA32\_SGX\_SVN\_STATUS MSR has the format shown in Table 41-2.

**Table 41-2. Layout of the IA32\_SGX\_SVN\_STATUS MSR**

Bit Position	Name	ACM Module ID	Value
0	Lock	N.A.	<ul style="list-style-type: none"> <li>▪ If 1, indicates that a non-faulting Intel SGX instruction has been executed, consequently, launching a properly signed ACM but with Intel SGX SVN value less than the BIOS specified Intel SGX SVN threshold would lead to an TXT shutdown.</li> <li>▪ If 0, indicates that the processor will allow a properly signed ACM to launch irrespective of the Intel SGX SVN value of the ACM.</li> </ul>
15:1	RSVD	N.A.	0
23:16	SGX_SVN_SINIT	SINIT ACM	<ul style="list-style-type: none"> <li>▪ If CPUID.01H:ECX.SMX = 1, this field reflects the expected threshold of Intel SGX SVN for the SINIT ACM.</li> <li>▪ If CPUID.01H:ECX.SMX = 0, this field is reserved (0).</li> </ul>
63:24	RSVD	N.A.	0

OS/VMM that wishes to launch an architectural ACM such as SINIT is expected to read the IA32\_SGX\_SVN\_STATUS MSR to determine whether the ACM can be launched or a new ACM is needed:

- If either the Intel SGX SVN of the ACM is greater than the value reported by IA32\_SGX\_SVN\_STATUS, or the lock bit in the IA32\_SGX\_SVN\_STATUS is not set, then the OS/VMM can safely launch the ACM.
- If the Intel SGX SVN value reported in the corresponding component of the IA32\_SGX\_SVN\_STATUS is greater than the Intel SGX SVN value in the ACM's header, and if bit 0 of IA32\_SGX\_SVN\_STATUS is 1, then the OS/VMM should not launch that version of the ACM. It should obtain an updated version of the ACM either from the BIOS or from an external resource.

However, OSVs/VMMs are strongly advised to update their version of the ACM any time they detect that the Intel SGX SVN of the ACM carried by the OS/VMM is lower than that reported by IA32\_SGX\_SVN\_STATUS MSR, irrespective of the setting of the lock bit.

## 41.12 INTERACTIONS WITH CACHING OF LINEAR-ADDRESS TRANSLATIONS

Entering and exiting an enclave causes the logical processor to flush all the global linear-address context as well as the linear-address context associated with the current VPID and PCID. The MONITOR FSM is also cleared.

## 41.13 INTERACTIONS WITH INTEL® TRANSACTIONAL SYNCHRONIZATION EXTENSIONS (INTEL® TSX)

1. ENCLU or ENCLS instructions inside an HLE region will cause the flow to be aborted and restarted non-speculatively. ENCLU or ENCLS instructions inside an RTM region will cause the flow to be aborted and transfer control to the fallback handler.
2. If XBEGIN is executed inside an enclave, the processor does NOT check whether the address of the fallback handler is within the enclave.
3. If an RTM transaction is executing inside an enclave and there is an attempt to fetch an instruction outside the enclave, the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.



4. If an RTM transaction is executing inside an enclave and there is a data access to an address within the enclave that denied due to EPCM content (e.g., to a page belonging to a different enclave), the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.

5. If an RTM transaction executing inside an enclave aborts and the address of the fallback handler is outside the enclave, a #GP is delivered after the abort (EIP reported is that of the fallback handler).

### 41.13.1 HLE and RTM Debug

RTM debug will be suppressed on opt-out enclave entry. After opt-out entry, the logical processor will behave as if IA32\_DEBUG\_CTL[15]=0. Any #DB detected inside an RTM transaction region will just cause an abort with no exception delivered.

After opt-in entry, if either DR7[11] = 0 OR IA32\_DEBUGCTL[15] = 0, any #DB or #BP detected inside an RTM transaction region will just cause an abort with no exception delivered.

After opt-in entry, if DR7[11] = 1 AND IA32\_DEBUGCTL[15] = 1, any #DB or #BP detected inside an RTM transaction will

- terminate speculative execution,
- set RIP to the address of the XBEGIN instruction, and
- be delivered as #DB (implying an Intel SGX AEX; any #BP is converted to #DB).
- DR6[16] will be cleared, indicating RTM debug (if the #DB causes a VM exit, DR6 is not modified but bit 16 of the pending debug exceptions field in the VMCS will be set).

## 41.14 INTEL® SGX INTERACTIONS WITH S STATES

Whenever an Intel SGX enabled processor enters S3-S5 state, enclaves are destroyed. This is due to the EPC being destroyed when power down occurs. It is the application runtime's responsibility to re-instantiate an enclave after a power transition for which the enclaves were destroyed.

## 41.15 INTEL® SGX INTERACTIONS WITH MACHINE CHECK ARCHITECTURE (MCA)

### 41.15.1 Interactions with MCA Events

All architecturally visible machine check events (#MC and CMCI) that are detected while inside an enclave cause an asynchronous enclave exit.

Any machine check exception (#MC) that occurs after Intel SGX is first enables causes Intel SGX to be disabled, (CPUID.SGX\_Leaf.0:EAX[SGX1] == 0). It cannot be enabled until after the next reset.

### 41.15.2 Machine Check Enables (IA32\_MCi\_CTL)

All supported IA32\_MCi\_CTL bits for all the machine check banks must be set for Intel SGX to be available (CPUID.SGX\_Leaf.0:EAX[SGX1] == 1). Any act of clearing bits from '1' to '0' in any of the IA32\_MCi\_CTL register may disable Intel SGX (set CPUID.SGX\_Leaf.0:EAX[SGX1] to 0) until the next reset.

### 41.15.3 CR4.MCE

CR4.MCE can be set or cleared with no interactions with Intel SGX.

## 41.16 INTEL® SGX INTERACTIONS WITH PROTECTED MODE VIRTUAL INTERRUPTS

ENCLS[EENTER] modifies neither EFLAGS.VIP nor EFLAGS.VIF.

ENCLS[ERESUME] loads EFLAGS in a manner similar to that of an execution of IRET with CPL = 3. This means that ERESUME modifies neither EFLAGS.VIP nor EFLAGS.VIF regardless of the value of the EFLAGS image in the SSA frame.

AEX saves EFLAGS.VIP and EFLAGS.VIF unmodified into the EFLAGS image in the SSA frame. AEX modifies neither EFLAGS.VIP nor EFLAGS.VIF after saving EFLAGS.

If CR4.PVI = 1, CPL = 3, EFLAGS.VM = 0, IOPL < 3, EFLAGS.VIP = 1, and EFLAGS.VIF = 0, execution of STI causes a #GP fault. In this case, STI modifies neither EFLAGS.IF nor EFLAGS.VIF. This behavior applies without change within an enclave (where CPL is always 3). Note that, if IOPL = 3, STI always sets EFLAGS.IF without fault; CR4.PVI, EFLAGS.VIP, and EFLAGS.VIF are neither consulted nor modified in this case.

## 41.17 INTEL SGX INTERACTION WITH PROTECTION KEYS

SGX interactions with PKRU are as follows:

- CPUID.(EAX=12H, ECX=1):ECX.PKRU indicates whether SECS.ATTRIBUTES.XFRM.PKRU can be set. If SECS.ATTRIBUTES.XFRM.PKRU is set, then PKRU is saved and cleared as part of AEX and is restored as part of ERESUME. If CR4.PKE is set, an enclave can execute RDPKRU and WRKRU independent of whether SECS.ATTRIBUTES.XFRM.PKRU is set.

SGX interactions with domain permission checks are as follows:

- 1) If CR4.PKE is not set, then legacy and SGX permission checks are not effected.
- 2) If CR4.PKE is set, then domain permission checks are applied to all non-enclave access and enclave accesses to user pages in addition to legacy and SGX permission checks at a higher priority than SGX permission checks.
- 3) Implicit accesses aren't subject to domain permission checks.



## 18. Updates to Chapter 2, Volume 4

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers*.

-----  
Changes to this chapter: Various updates/additions throughout chapter. Added Section 2.16.1 "MSRs Specific to 7th Generation Intel® Core™ Processors based on Kaby Lake Microarchitecture".

## CHAPTER 2 MODEL-SPECIFIC REGISTERS (MSRS)

This chapter lists MSRs across Intel processor families. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions.

Register addresses are given in both hexadecimal and decimal. The register name is the mnemonic register name and the bit description describes individual bits in registers.

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H leaf function to query the combination of DisplayFamily and DisplayModel to determine model-specific availability of MSRs (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Table 2-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

**Table 2-1. CPUID Signature Values of DisplayFamily\_DisplayModel**

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_85H	Future Intel® Xeon Phi™ Processor based on Knights Mill microarchitecture
06_57H	Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series based on Knights Landing microarchitecture
06_66H	Future Intel® Core™ processors based on Cannon Lake microarchitecture
06_8EH, 06_9EH	7th generation Intel® Core™ processors based on Kaby Lake microarchitecture
06_55H	Intel® Xeon® Processor Scalable Family based on Skylake microarchitecture
06_4EH, 06_5EH	6th generation Intel Core processors and Intel Xeon processor E3-1500m v5 product family and E3-1200 v5 product family based on Skylake microarchitecture
06_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture
06_4FH	Intel Xeon processor E5 v4 Family based on Broadwell microarchitecture, Intel Xeon processor E7 v4 Family, Intel Core i7-69xx Processor Extreme Edition
06_47H	5th generation Intel Core processors, Intel Xeon processor E3-1200 v4 product family based on Broadwell microarchitecture
06_3DH	Intel Core M-5xxx Processor, 5th generation Intel Core processors based on Broadwell microarchitecture
06_3FH	Intel Xeon processor E5-4600/2600/1600 v3 product families, Intel Xeon processor E7 v3 product families based on Haswell-E microarchitecture, Intel Core i7-59xx Processor Extreme Edition
06_3CH, 06_45H, 06_46H	4th Generation Intel Core processor and Intel Xeon processor E3-1200 v3 product family based on Haswell microarchitecture
06_3EH	Intel Xeon processor E7-8800/4800/2800 v2 product families based on Ivy Bridge-E microarchitecture
06_3EH	Intel Xeon processor E5-2600/1600 v2 product families and Intel Xeon processor E5-2400 v2 product family based on Ivy Bridge-E microarchitecture, Intel Core i7-49xx Processor Extreme Edition
06_3AH	3rd Generation Intel Core Processor and Intel Xeon processor E3-1200 v2 product family based on Ivy Bridge microarchitecture
06_2DH	Intel Xeon processor E5 Family based on Intel microarchitecture code name Sandy Bridge, Intel Core i7-39xx Processor Extreme Edition
06_2FH	Intel Xeon Processor E7 Family
06_2AH	Intel Xeon processor E3-1200 product family; 2nd Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series

**Table 2-1. CPUID Signature (Contd.)Values of DisplayFamily\_DisplayModel (Contd.)**

DisplayFamily_DisplayModel	Processor Families/Processor Number 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_7AH	Intel® Atom™ processors based on Goldmont Plus Microarchitecture
06_5FH	Intel Atom processors based on Goldmont Microarchitecture (code name Denverton)
06_5CH	Intel Atom processors based on Goldmont Microarchitecture
06_4CH	Intel Atom processor X7-Z8000 and X5-Z8000 series based on Airmont Microarchitecture
06_5DH	Intel Atom processor X3-C3000 based on Silvermont Microarchitecture
06_5AH	Intel Atom processor Z3500 series
06_4AH	Intel Atom processor Z3400 series
06_37H	Intel Atom processor E3000 series, Z3600 series, Z3700 series
06_4DH	Intel Atom processor C2000 series
06_36H	Intel Atom processor S1000 Series
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	Intel Atom processor family, Intel Atom processor D2000, N2000, E2000, Z2000, C1000 series
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon processor, Intel Xeon processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon processor, Intel Pentium III processor
06_03H, 06_05H	Intel Pentium II Xeon processor, Intel Pentium II processor
06_01H	Intel Pentium Pro processor
05_01H, 05_02H, 05_04H	Intel Pentium processor, Intel Pentium processor with MMX Technology

The Intel® Quark™ SoC X1000 processor can be identified by the signature of DisplayFamily\_DisplayModel = 05\_09H and SteppingID = 0

## 2.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32\_”. Table 2-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table 2-2 and certain bit fields in an MSR address that may overlap with architectural MSR addresses are model-specific.

Code that accesses a machine specified MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table 2-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of "DF\_DM" (see Table 2-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as "MAXPHYADDR" in Table 2-2. "MAXPHYADDR" is reported by CPUID.8000\_0008H leaf.

MSR address range between 40000000H - 400000FFH is marked as a specially reserved range. All existing and future processors will not implement any features using any MSR in this range.

**Table 2-2. IA-32 Architectural MSRs**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
0H	0	IA32_P5_MC_ADDR (P5_MC_ADDR)	See Section 2.22, "MSRs in Pentium Processors."	<b>Pentium Processor (05_01H)</b>
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Section 2.22, "MSRs in Pentium Processors."	DF_DM = 05_01H
6H	6	IA32_MONITOR_FILTER_SIZE	See Section 8.10.5, "Monitor/Mwait Address Range Determination."	0F_03H
10H	16	IA32_TIME_STAMP_COUNTER (TSC)	See Section 17.17, "Time-Stamp Counter."	05_01H
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	<b>Platform ID (RO)</b> The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	06_01H
		49:0	Reserved.	
		52:50	<b>Platform Id (RO)</b> Contains information concerning the intended platform for the processor.  52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7	
		63:53	Reserved.	
1BH	27	IA32_APIC_BASE (APIC_BASE)	This register holds the APIC base address, permitting the relocation of the APIC memory map. See Section 10.4.4, "Local APIC Status and Location" and Section 10.4.5, "Relocating the Local APIC Registers".	06_01H
		7:0	Reserved	
		8	BSP flag (R/W)	

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		9	Reserved	
		10	Enable x2APIC mode	06_1AH
		11	APIC Global Enable (R/W)	
		(MAXPHYADDR - 1):12	APIC Base (R/W)	
		63: MAXPHYADDR	Reserved	
3AH	58	IA32_FEATURE_CONTROL	<b>Control Features in Intel 64 Processor (R/W)</b>	If any one enumeration condition for defined bit field holds
		0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written, writes to this bit will result in GP(0).  Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support for Intel Virtualization Technology and prior to transferring control to an option ROM or the OS. Hence, once the Lock bit is set, the entire IA32_FEATURE_CONTROL contents are preserved across RESET when PWRGOOD is not deasserted.	If any one enumeration condition for defined bit field position greater than bit 0 holds
		1	Enable VMX inside SMX operation (R/WL): This bit enables a system executive to use VMX in conjunction with SMX to support Intel® Trusted Execution Technology.  BIOS must set this bit only when the CPUID function 1 returns VMX feature flag and SMX feature flag set (ECX bits 5 and 6 respectively).	If CPUID.01H:ECX[5] = 1 && CPUID.01H:ECX[6] = 1
		2	Enable VMX outside SMX operation (R/WL): This bit enables VMX for system executive that do not require SMX.  BIOS must set this bit only when the CPUID function 1 returns VMX feature flag set (ECX bit 5).	If CPUID.01H:ECX[5] = 1
		7:3	Reserved	
		14:8	SENTER Local Function Enables (R/WL): When set, each bit in the field represents an enable control for a corresponding SENTER function. This bit is supported only if CPUID.1:ECX.[bit 6] is set	If CPUID.01H:ECX[6] = 1
		15	SENTER Global Enable (R/WL): This bit must be set to enable SENTER leaf functions. This bit is supported only if CPUID.1:ECX.[bit 6] is set	If CPUID.01H:ECX[6] = 1
		16	Reserved	



Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		17	SGX Launch Control Enable (R/WL): This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR.	If CPUID.(EAX=07H, ECX=0H): ECX[30] = 1
		18	SGX Global Enable (R/WL): This bit must be set to enable SGX leaf functions.	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		19	Reserved	
		20	LMCE On (R/WL): When set, system software can program the MSRs associated with LMCE to configure delivery of some machine check exceptions to a single logical processor.	If IA32_MCG_CAP[27] = 1
		63:21	Reserved	
3BH	59	IA32_TSC_ADJUST	Per Logical Processor TSC Adjust (R/Write to clear)	If CPUID.(EAX=07H, ECX=0H): EBX[1] = 1
		63:0	<b>THREAD_ADJUST:</b> 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.	
79H	121	IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)	BIOS Update Trigger (W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 9.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
8BH	139	IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)	BIOS Update Signature (RO) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
		31:0	Reserved	
		63:32	It is recommended that this field be pre-loaded with 0 prior to executing CPUID. If the field remains 0 following the execution of CPUID; this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature.	

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
8CH	140	IA32_SGXLEPUBKEYHASH0	IA32_SGXLEPUBKEYHASH[63:0] (R/W) Bits 63:0 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1, Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1
8DH	141	IA32_SGXLEPUBKEYHASH1	IA32_SGXLEPUBKEYHASH[127:64] (R/W) Bits 127:64 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1, Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1
8EH	142	IA32_SGXLEPUBKEYHASH2	IA32_SGXLEPUBKEYHASH[191:128] (R/W) Bits 191:128 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1, Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1
8FH	143	IA32_SGXLEPUBKEYHASH3	IA32_SGXLEPUBKEYHASH[255:192] (R/W) Bits 255:192 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1
9BH	155	IA32_SMM_MONITOR_CTL	SMM Monitor Configuration (R/W)	If CPUID.01H: ECX[5]=1    CPUID.01H: ECX[6] = 1
		0	Valid (R/W)	
		1	Reserved	
		2	Controls SMI unblocking by VMXOFF (see Section 34.14.4)	If IA32_VMX_MISC[28]
		11:3	Reserved	
		31:12	MSEG Base (R/W)	
		63:32	Reserved	
9EH	158	IA32_SMBASE	Base address of the logical processor's SMRAM image (RO, SMM only)	If IA32_VMX_MISC[15]
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
C2H	194	IA32_PMC1 (PERFCTR1)	General Performance Counter 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
C3H	195	IA32_PMC2	General Performance Counter 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
C4H	196	IA32_PMC3	General Performance Counter 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
C5H	197	IA32_PMC4	General Performance Counter 4 (R/W)	If CPUID.0AH: EAX[15:8] > 4
C6H	198	IA32_PMC5	General Performance Counter 5 (R/W)	If CPUID.0AH: EAX[15:8] > 5
C7H	199	IA32_PMC6	General Performance Counter 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
C8H	200	IA32_PMC7	General Performance Counter 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7
E7H	231	IA32_MPERF	TSC Frequency Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	<b>CO_MCNT: CO TSC Frequency Clock Count</b> Increments at fixed interval (relative to TSC freq.) when the logical processor is in CO. Cleared upon overflow / wrap-around of IA32_APERF.	
E8H	232	IA32_APERF	Actual Performance Clock Counter (R/Write to clear).	If CPUID.06H: ECX[0] = 1
		63:0	<b>CO_ACNT: CO Actual Frequency Clock Count</b> Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in CO. Cleared upon overflow / wrap-around of IA32_MPERF.	
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (RO) Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	06_01H
		7:0	VCNT: The number of variable memory type ranges in the processor.	
		8	Fixed range MTRRs are supported when set.	
		9	Reserved.	
		10	WC Supported when set.	
		11	SMRR Supported when set.	
		63:12	Reserved.	
174H	372	IA32_SYSENTER_CS	SYSENTER_CS_MSR (R/W)	06_01H
		15:0	CS Selector.	
		31:16	Not used.	Can be read and written.
		63:32	Not used.	Writes ignored; reads return zero.

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
175H	373	IA32_SYSENTER_ESP	SYSENTER_ESP_MSR (R/W)	06_01H
176H	374	IA32_SYSENTER_EIP	SYSENTER_EIP_MSR (R/W)	06_01H
179H	377	IA32_MCG_CAP (MCG_CAP)	Global Machine Check Capability (RO)	06_01H
		7:0	Count: Number of reporting banks.	
		8	MCG_CTL_P: IA32_MCG_CTL is present if this bit is set	
		9	MCG_EXT_P: Extended machine check state registers are present if this bit is set	
		10	MCP_CMCI_P: Support for corrected MC error event is present.	06_01H
		11	MCG_TES_P: Threshold-based error status register are present if this bit is set.	
		15:12	Reserved	
		23:16	MCG_EXT_CNT: Number of extended machine check state registers present.	
		24	MCG_SER_P: The processor supports software error recovery if this bit is set.	
		25	Reserved.	
		26	MCG_ELOG_P: Indicates that the processor allows platform firmware to be invoked when an error is detected so that it may provide additional platform specific information in an ACPI format "Generic Error Data Entry" that augments the data included in machine check bank registers.	06_3EH
		27	MCG_LMCE_P: Indicates that the processor support extended state in IA32_MCG_STATUS and associated MSR necessary to configure Local Machine Check Exception (LMCE).	06_3EH
63:28	Reserved.			
17AH	378	IA32_MCG_STATUS (MCG_STATUS)	Global Machine Check Status (R/W0)	06_01H
		0	RIPV. Restart IP valid	06_01H
		1	EIPV. Error IP valid	06_01H
		2	MCIP. Machine check in progress	06_01H
		3	LMCE_S.	If IA32_MCG_CAP.LMCE_P[2:7] = 1
		63:4	Reserved.	
17BH	379	IA32_MCG_CTL (MCG_CTL)	Global Machine Check Control (R/W)	If IA32_MCG_CAP.CTL_P[8] = 1
180H-185H	384-389	Reserved		06_0EH <sup>1</sup>

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
186H	390	IA32_PERFEVTSELO (PERFEVTSELO)	Performance Event Select Register 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
		7:0	Event Select: Selects a performance event logic unit.	
		15:8	UMask: Qualifies the microarchitectural condition to detect on the selected event logic.	
		16	USR: Counts while in privilege level is not ring 0.	
		17	OS: Counts while in privilege level is ring 0.	
		18	Edge: Enables edge detection if set.	
		19	PC: enables pin control.	
		20	INT: enables interrupt on counter overflow.	
		21	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
		22	EN: enables the corresponding performance counter to commence counting when this bit is set.	
		23	INV: invert the CMASK.	
	31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.		
	63:32	Reserved.		
187H	391	IA32_PERFEVTSEL1 (PERFEVTSEL1)	Performance Event Select Register 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
188H	392	IA32_PERFEVTSEL2	Performance Event Select Register 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
189H	393	IA32_PERFEVTSEL3	Performance Event Select Register 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
18AH-197H	394-407	Reserved		06_0EH <sup>2</sup>
198H	408	IA32_PERF_STATUS	Current performance status. (RO) See Section 14.1.1, "Software Interface For Initiating Performance State Transitions".	0F_03H
		15:0	Current performance State Value	
		63:16	Reserved.	

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
199H	409	IA32_PERF_CTL	Performance Control MSR. (R/W) Software makes a request for a new Performance state (P-State) by writing this MSR. See Section 14.1.1, "Software Interface For Initiating Performance State Transitions".	0F_03H
		15:0	Target performance State Value	
		31:16	Reserved.	
		32	IDA Engage. (R/W) When set to 1: disengages IDA	06_0FH (Mobile only)
		63:33	Reserved.	
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation Control (R/W) See Section 14.7.3, "Software Controlled Clock Modulation."	If CPUID.01H:EDX[22] = 1
		0	Extended On-Demand Clock Modulation Duty Cycle:	If CPUID.06H:EAX[5] = 1
		3:1	On-Demand Clock Modulation Duty Cycle: Specific encoded values for target duty cycle modulation.	If CPUID.01H:EDX[22] = 1
		4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	If CPUID.01H:EDX[22] = 1
		63:5	Reserved.	
19BH	411	IA32_THERM_INTERRUPT	Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the processor's thermal sensors and thermal monitor. See Section 14.7.2, "Thermal Monitor."	If CPUID.01H:EDX[22] = 1
		0	High-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
		1	Low-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
		2	PROCHOT# Interrupt Enable	If CPUID.01H:EDX[22] = 1
		3	FORCEPR# Interrupt Enable	If CPUID.01H:EDX[22] = 1
		4	Critical Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
		7:5	Reserved.	
		14:8	Threshold #1 Value	If CPUID.01H:EDX[22] = 1
		15	Threshold #1 Interrupt Enable	If CPUID.01H:EDX[22] = 1
		22:16	Threshold #2 Value	If CPUID.01H:EDX[22] = 1
		23	Threshold #2 Interrupt Enable	If CPUID.01H:EDX[22] = 1
		24	Power Limit Notification Enable	If CPUID.06H:EAX[4] = 1
63:25	Reserved.			

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
19CH	412	IA32_THERM_STATUS	Thermal Status Information (RO) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 14.7.2, "Thermal Monitor"	If CPUID.01H:EDX[22] = 1
		0	Thermal Status (RO):	If CPUID.01H:EDX[22] = 1
		1	Thermal Status Log (R/W):	If CPUID.01H:EDX[22] = 1
		2	PROCHOT # or FORCEPR# event (RO)	If CPUID.01H:EDX[22] = 1
		3	PROCHOT # or FORCEPR# log (R/WCO)	If CPUID.01H:EDX[22] = 1
		4	Critical Temperature Status (RO)	If CPUID.01H:EDX[22] = 1
		5	Critical Temperature Status log (R/WCO)	If CPUID.01H:EDX[22] = 1
		6	Thermal Threshold #1 Status (RO)	If CPUID.01H:ECX[8] = 1
		7	Thermal Threshold #1 log (R/WCO)	If CPUID.01H:ECX[8] = 1
		8	Thermal Threshold #2 Status (RO)	If CPUID.01H:ECX[8] = 1
		9	Thermal Threshold #2 log (R/WCO)	If CPUID.01H:ECX[8] = 1
		10	Power Limitation Status (RO)	If CPUID.06H:EAX[4] = 1
		11	Power Limitation log (R/WCO)	If CPUID.06H:EAX[4] = 1
		12	Current Limit Status (RO)	If CPUID.06H:EAX[7] = 1
		13	Current Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
		14	Cross Domain Limit Status (RO)	If CPUID.06H:EAX[7] = 1
		15	Cross Domain Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
		22:16	Digital Readout (RO)	If CPUID.06H:EAX[0] = 1
		26:23	Reserved.	
		30:27	Resolution in Degrees Celsius (RO)	If CPUID.06H:EAX[0] = 1
31	Reading Valid (RO)	If CPUID.06H:EAX[0] = 1		
63:32	Reserved.			
1A0H	416	IA32_MISC_ENABLE	<b>Enable Misc. Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.	
		0	<b>Fast-Strings Enable</b> When set, the fast-strings feature (for REP MOVS and REP STORS) is enabled (default); when clear, fast-strings are disabled.	OF_OH
		2:1	Reserved.	

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		3	<b>Automatic Thermal Control Circuit Enable (R/W)</b> 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled. Note: In some products clearing this bit might be ignored in critical thermal conditions, and TM1, TM2 and adaptive thermal throttling will still be activated. The default value of this field varies with product . See respective tables where default value is listed.	0F_0H
		6:4	Reserved	
		7	<b>Performance Monitoring Available (R)</b> 1 = Performance monitoring enabled 0 = Performance monitoring disabled	0F_0H
		10:8	Reserved.	
		11	<b>Branch Trace Storage Unavailable (RO)</b> 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported	0F_0H
		12	<b>Processor Event Based Sampling (PEBS) Unavailable (RO)</b> 1 = PEBS is not supported; 0 = PEBS is supported.	06_0FH
		15:13	Reserved.	
		16	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> 0= Enhanced Intel SpeedStep Technology disabled 1 = Enhanced Intel SpeedStep Technology enabled	If CPUID.01H: ECX[7] =1
		17	Reserved.	



Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		18	<p><b>ENABLE MONITOR FSM (R/W)</b></p> <p>When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0.</p> <p>When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1).</p> <p>If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.</p>	0F_03H
		21:19	Reserved.	
		22	<p><b>Limit CPUID Maxval (R/W)</b></p> <p>When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 2. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 2.</p> <p>Before setting this bit, BIOS must execute the CPUID.0H and examine the maximum value returned in EAX[7:0]. If the maximum value is greater than 2, this bit is supported.</p> <p>Otherwise, this bit is not supported. Setting this bit when the maximum value is not greater than 2 may generate a #GP exception.</p> <p>Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 2.</p>	0F_03H
		23	<p><b>xTPR Message Disable (R/W)</b></p> <p>When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.</p>	if CPUID.01H:ECX[14] = 1
		33:24	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		34	<p><b>XD Bit Disable (R/W)</b></p> <p>When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H: EDX[20]=0).</p> <p>When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages.</p> <p>BIOS must not alter the contents of this bit location, if XD bit is not supported. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.</p>	if CPUID.80000001H:EDX[20] = 1
		63:35	Reserved.	
1B0H	432	IA32_ENERGY_PERF_BIAS	Performance Energy Bias Hint (R/W)	if CPUID.6H:ECX[3] = 1
		3:0	<p>Power Policy Preference:</p> <p>0 indicates preference to highest performance.</p> <p>15 indicates preference to maximize energy saving.</p>	
		63:4	Reserved.	
1B1H	433	IA32_PACKAGE_THERM_STATUS	<p>Package Thermal Status Information (RO)</p> <p>Contains status information about the package's thermal sensor.</p> <p>See Section 14.8, "Package Level Thermal Management."</p>	If CPUID.06H: EAX[6] = 1
		0	Pkg Thermal Status (RO):	
		1	Pkg Thermal Status Log (R/W):	
		2	Pkg PROCHOT # event (RO)	
		3	Pkg PROCHOT # log (R/WCO)	
		4	Pkg Critical Temperature Status (RO)	
		5	Pkg Critical Temperature Status log (R/WCO)	
		6	Pkg Thermal Threshold #1 Status (RO)	
		7	Pkg Thermal Threshold #1 log (R/WCO)	
		8	Pkg Thermal Threshold #2 Status (RO)	
		9	Pkg Thermal Threshold #1 log (R/WCO)	
		10	Pkg Power Limitation Status (RO)	
		11	Pkg Power Limitation log (R/WCO)	
		15:12	Reserved.	
22:16	Pkg Digital Readout (RO)			

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:23	Reserved.	
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Pkg Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the package's thermal sensor. See Section 14.8, "Package Level Thermal Management."	If CPUID.06H: EAX[6] = 1
		0	Pkg High-Temperature Interrupt Enable	
		1	Pkg Low-Temperature Interrupt Enable	
		2	Pkg PROCHOT# Interrupt Enable	
		3	Reserved.	
		4	Pkg Overheat Interrupt Enable	
		7:5	Reserved.	
		14:8	Pkg Threshold #1 Value	
		15	Pkg Threshold #1 Interrupt Enable	
		22:16	Pkg Threshold #2 Value	
		23	Pkg Threshold #2 Interrupt Enable	
		24	Pkg Power Limit Notification Enable	
		63:25	Reserved.	
		1D9H	473	IA32_DEBUGCTL (MSR_DEBUGCTLA, MSR_DEBUGCTLB)
0	LBR: Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.			06_01H
1	BTF: Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.			06_01H
5:2	Reserved.			
6	TR: Setting this bit to 1 enables branch trace messages to be sent.			06_0EH
7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.			06_0EH
8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.			06_0EH
	9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH
		11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H: ECX[15] = 1 && CPUID.0AH: EAX[7:0] > 1
		12	FREEZE_PERFMON_ON_PMI: When set, each ENABLE bit of the global counter control MSR are frozen (address 38FH) on a PMI request	If CPUID.01H: ECX[15] = 1 && CPUID.0AH: EAX[7:0] > 1
		13	ENABLE_UNCORE_PMI: When set, enables the logical processor to receive and generate PMI on behalf of the uncore.	06_1AH
		14	FREEZE_WHILE_SMM: When set, freezes perfmon and trace messages while in SMM.	If IA32_PERF_CAPABILITIES[12] = 1
		15	RTM_DEBUG: When set, enables DR7 debug bit on XBEGIN	If (CPUID.(EAX=07H, ECX=0):EBX[11] = 1)
		63:16	Reserved.	
1F2H	498	IA32_SMRR_PHYSBASE	<b>SMRR Base Address (Writeable only in SMM)</b> Base address of SMM memory range.	If IA32_MTRRCAP.SMRR[11] = 1
		7:0	Type. Specifies memory type of the range.	
		11:8	Reserved.	
		31:12	<b>PhysBase.</b> SMRR physical Base Address.	
		63:32	Reserved.	
1F3H	499	IA32_SMRR_PHYSMASK	<b>SMRR Range Mask. (Writeable only in SMM)</b> Range Mask of SMM memory range.	If IA32_MTRRCAP[SMRR] = 1
		10:0	Reserved.	
		11	<b>Valid</b> Enable range mask.	
		31:12	<b>PhysMask</b> SMRR address range mask.	
		63:32	Reserved.	
1F8H	504	IA32_PLATFORM_DCA_CAP	DCA Capability (R)	If CPUID.01H: ECX[18] = 1
1F9H	505	IA32_CPU_DCA_CAP	If set, CPU supports Prefetch-Hint type.	If CPUID.01H: ECX[18] = 1
1FAH	506	IA32_DCA_0_CAP	DCA type 0 Status and Control register.	If CPUID.01H: ECX[18] = 1
		0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	
		2:1	TRANSACTION	
		6:3	DCA_TYPE	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		10:7	DCA_QUEUE_SIZE	
		12:11	Reserved.	
		16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	
		23:17	Reserved.	
		24	SW_BLOCK: SW can request DCA block by setting this bit.	
		25	Reserved.	
		26	HW_BLOCK: Set when DCA is blocked by HW (e.g. CRO.CD = 1).	
		31:27	Reserved.	
200H	512	IA32_MTRR_PHYSBASE0 (MTRRphysBase0)	See Section 11.11.2.3, "Variable Range MTRRs."	If CPUID.01H: EDX.MTRR[12] = 1
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	If CPUID.01H: EDX.MTRR[12] = 1
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	If CPUID.01H: EDX.MTRR[12] = 1
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	If CPUID.01H: EDX.MTRR[12] = 1
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	If CPUID.01H: EDX.MTRR[12] = 1
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	If CPUID.01H: EDX.MTRR[12] = 1
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	If CPUID.01H: EDX.MTRR[12] = 1
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	If CPUID.01H: EDX.MTRR[12] = 1
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	If CPUID.01H: EDX.MTRR[12] = 1
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	If CPUID.01H: EDX.MTRR[12] = 1
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	If CPUID.01H: EDX.MTRR[12] = 1
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	If CPUID.01H: EDX.MTRR[12] = 1
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	If CPUID.01H: EDX.MTRR[12] = 1
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	If CPUID.01H: EDX.MTRR[12] = 1
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	If CPUID.01H: EDX.MTRR[12] = 1
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	If CPUID.01H: EDX.MTRR[12] = 1

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
210H	528	IA32_MTRR_PHYSBASE8	MTRRphysBase8	if IA32_MTRRCAP[7:0] > 8
211H	529	IA32_MTRR_PHYSMASK8	MTRRphysMask8	if IA32_MTRRCAP[7:0] > 8
212H	530	IA32_MTRR_PHYSBASE9	MTRRphysBase9	if IA32_MTRRCAP[7:0] > 9
213H	531	IA32_MTRR_PHYSMASK9	MTRRphysMask9	if IA32_MTRRCAP[7:0] > 9
250H	592	IA32_MTRR_FIX64K_00000	MTRRfix64K_00000	If CPUID.01H: EDX.MTRR[12] = 1
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	If CPUID.01H: EDX.MTRR[12] = 1
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	If CPUID.01H: EDX.MTRR[12] = 1
268H	616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000 )	See Section 11.11.2.2, "Fixed Range MTRRs."	If CPUID.01H: EDX.MTRR[12] = 1
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	If CPUID.01H: EDX.MTRR[12] = 1
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	If CPUID.01H: EDX.MTRR[12] = 1
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	If CPUID.01H: EDX.MTRR[12] = 1
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	If CPUID.01H: EDX.MTRR[12] = 1
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	If CPUID.01H: EDX.MTRR[12] = 1
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	If CPUID.01H: EDX.MTRR[12] = 1
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	If CPUID.01H: EDX.MTRR[12] = 1
277H	631	IA32_PAT	IA32_PAT (R/W)	If CPUID.01H: EDX.MTRR[16] = 1
		2:0	PA0	
		7:3	Reserved.	
		10:8	PA1	
		15:11	Reserved.	
		18:16	PA2	
		23:19	Reserved.	
		26:24	PA3	
		31:27	Reserved.	
		34:32	PA4	
		39:35	Reserved.	
		42:40	PA5	
		47:43	Reserved.	
50:48	PA6			

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		55:51	Reserved.	
		58:56	PA7	
		63:59	Reserved.	
280H	640	IA32_MCO_CTL2	MSR to enable/disable CMCI capability for bank 0. (R/W) See Section 15.3.2.5, "IA32_MCi_CTL2 MSRs".	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 0
		14:0	Corrected error count threshold.	
		29:15	Reserved.	
		30	CMCI_EN	
		63:31	Reserved.	
281H	641	IA32_MC1_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1
282H	642	IA32_MC2_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
283H	643	IA32_MC3_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3
284H	644	IA32_MC4_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4
285H	645	IA32_MC5_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
286H	646	IA32_MC6_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
287H	647	IA32_MC7_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
288H	648	IA32_MC8_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
289H	649	IA32_MC9_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9
28AH	650	IA32_MC10_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
28BH	651	IA32_MC11_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
28CH	652	IA32_MC12_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
28DH	653	IA32_MC13_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
28EH	654	IA32_MC14_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
28FH	655	IA32_MC15_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15
290H	656	IA32_MC16_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16
291H	657	IA32_MC17_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17
292H	658	IA32_MC18_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18
293H	659	IA32_MC19_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
294H	660	IA32_MC20_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
295H	661	IA32_MC21_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
296H	662	IA32_MC22_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
297H	663	IA32_MC23_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
298H	664	IA32_MC24_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24
299H	665	IA32_MC25_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
29AH	666	IA32_MC26_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26



Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
29BH	667	IA32_MC27_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
29CH	668	IA32_MC28_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
29DH	669	IA32_MC29_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
29EH	670	IA32_MC30_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30
29FH	671	IA32_MC31_CTL2	(R/W) same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 31
2FFH	767	IA32_MTRR_DEF_TYPE	MTRRdefType (R/W)	If CPUID.01H: EDX.MTRR[12] = 1
		2:0	Default Memory Type	
		9:3	Reserved.	
		10	Fixed Range MTRR Enable	
		11	MTRR Enable	
		63:12	Reserved.	
309H	777	IA32_FIXED_CTR0 (MSR_PERF_FIXED_CTR0)	Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any.	If CPUID.0AH: EDX[4:0] > 0
30AH	778	IA32_FIXED_CTR1 (MSR_PERF_FIXED_CTR1)	Fixed-Function Performance Counter 1 (R/W): Counts CPU_CLK_Unhalted.Core	If CPUID.0AH: EDX[4:0] > 1
30BH	779	IA32_FIXED_CTR2 (MSR_PERF_FIXED_CTR2)	Fixed-Function Performance Counter 2 (R/W): Counts CPU_CLK_Unhalted.Ref	If CPUID.0AH: EDX[4:0] > 2
345H	837	IA32_PERF_CAPABILITIES	Read Only MSR that enumerates the existence of performance monitoring features. (RO)	If CPUID.01H: ECX[15] = 1
		5:0	LBR format	
		6	PEBS Trap	
		7	PEBSSaveArchRegs	
		11:8	PEBS Record Format	
		12	1: Freeze while SMM is supported.	
		13	1: Full width of counter writable via IA32_A_PMCx.	
		63:14	Reserved.	

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
38DH	909	IA32_FIXED_CTR_CTRL	Fixed-Function Performance Counter Control (R/W)  Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.	If CPUID.0AH: EAX[7:0] > 1
		0	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0.	
		1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0.	
		2	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH: EAX[7:0] > 2
		3	EN0_PMI: Enable PMI when fixed counter 0 overflows.	
		4	EN1_OS: Enable Fixed Counter 1 to count while CPL = 0.	
		5	EN1_Usr: Enable Fixed Counter 1 to count while CPL > 0.	
		6	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH: EAX[7:0] > 2
		7	EN1_PMI: Enable PMI when fixed counter 1 overflows.	
		8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0.	
		9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0.	
		10	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH: EAX[7:0] > 2
		11	EN2_PMI: Enable PMI when fixed counter 2 overflows.	
	63:12	Reserved.		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
38EH	910	IA32_PERF_GLOBAL_STATUS	Global Performance Counter Status (RO)	If CPUID.0AH: EAX[7:0] > 0
		0	Ovf_PMC0: Overflow status of IA32_PMC0.	If CPUID.0AH: EAX[15:8] > 0
		1	Ovf_PMC1: Overflow status of IA32_PMC1.	If CPUID.0AH: EAX[15:8] > 1
		2	Ovf_PMC2: Overflow status of IA32_PMC2.	If CPUID.0AH: EAX[15:8] > 2
		3	Ovf_PMC3: Overflow status of IA32_PMC3.	If CPUID.0AH: EAX[15:8] > 3
		31:4	Reserved.	
		32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0.	If CPUID.0AH: EAX[7:0] > 1
		33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1.	If CPUID.0AH: EAX[7:0] > 1
		34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2.	If CPUID.0AH: EAX[7:0] > 1
		54:35	Reserved.	
		55	Trace_ToPA_PMI: A PMI occurred due to a ToPA entry memory buffer was completely filled.	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && IA32_RTIT_CTL.ToPA = 1
		57:56	Reserved.	
		58	LBR_Frz: LBRs are frozen due to <ul style="list-style-type: none"> <li>▪ IA32_DEBUGCTL.FREEZE_LBR_ON_PMI=1,</li> <li>▪ The LBR stack overflowed</li> </ul>	If CPUID.0AH: EAX[7:0] > 3
		59	CTR_Frz: Performance counters in the core PMU are frozen due to <ul style="list-style-type: none"> <li>▪ IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1,</li> <li>▪ one or more core PMU counters overflowed.</li> </ul>	If CPUID.0AH: EAX[7:0] > 3
		60	ASCI: Data in the performance counters in the core PMU may include contributions from the direct or indirect operation intel SGX to protect an enclave.	If CPUID.(EAX=07H, ECX=0):EBX[2] = 1
		61	Ovf_Uncore: Uncore counter overflow status.	If CPUID.0AH: EAX[7:0] > 2
62	OvfBuf: DS SAVE area Buffer overflow status.	If CPUID.0AH: EAX[7:0] > 0		
63	CondChgd: status bits of this register has changed.	If CPUID.0AH: EAX[7:0] > 0		

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
38FH	911	IA32_PERF_GLOBAL_CTRL	Global Performance Counter Control (R/W) Counter increments while the result of ANDing respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.	If CPUID.0AH: EAX[7:0] > 0
		0	EN_PMC0	If CPUID.0AH: EAX[15:8] > 0
		1	EN_PMC1	If CPUID.0AH: EAX[15:8] > 1
		2	EN_PMC2	If CPUID.0AH: EAX[15:8] > 2
		n	EN_PMCn	If CPUID.0AH: EAX[15:8] > n
		31:n+1	Reserved.	
		32	EN_FIXED_CTR0	If CPUID.0AH: EDX[4:0] > 0
		33	EN_FIXED_CTR1	If CPUID.0AH: EDX[4:0] > 1
		34	EN_FIXED_CTR2	If CPUID.0AH: EDX[4:0] > 2
		63:35	Reserved.	
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0 && CPUID.0AH: EAX[7:0] <= 3
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[15:8] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved.	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH: EDX[4:0] > 0
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EDX[4:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EDX[4:0] > 2
		54:35	Reserved.	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && IA32_RTIT_CTL.ToPA = 1
		60:56	Reserved.	
		61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
		62	Set 1 to Clear OvfBuf: bit.	If CPUID.0AH: EAX[7:0] > 0
63	Set to 1 to clear CondChgd: bit.	If CPUID.0AH: EAX[7:0] > 0		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
390H	912	IA32_PERF_GLOBAL_STATUS_RESET	Global Performance Counter Overflow Reset Control (R/W)	If CPUID.OAH: EAX[7:0] > 3
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.OAH: EAX[15:8] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.OAH: EAX[15:8] > 1
		2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.OAH: EAX[15:8] > 2
		n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.OAH: EAX[15:8] > n
		31:n	Reserved.	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.OAH: EDX[4:0] > 0
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.OAH: EDX[4:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.OAH: EDX[4:0] > 2
		54:35	Reserved.	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && IA32_RTIT_CTL.ToPA[8] = 1
		57:56	Reserved.	
		58	Set 1 to Clear LBR_Frz bit.	If CPUID.OAH: EAX[7:0] > 3
		59	Set 1 to Clear CTR_Frz bit.	If CPUID.OAH: EAX[7:0] > 3
		58	Set 1 to Clear ASCII bit.	If CPUID.OAH: EAX[7:0] > 3
		61	Set 1 to Clear Ovf_Uncore bit.	O6_2EH
		62	Set 1 to Clear OvfBuf: bit.	If CPUID.OAH: EAX[7:0] > 0
		63	Set to 1 to clear CondChgd: bit.	If CPUID.OAH: EAX[7:0] > 0
391H	913	IA32_PERF_GLOBAL_STATUS_SET	Global Performance Counter Overflow Set Control (R/W)	If CPUID.OAH: EAX[7:0] > 3
		0	Set 1 to cause Ovf_PMC0 = 1.	If CPUID.OAH: EAX[7:0] > 3
		1	Set 1 to cause Ovf_PMC1 = 1	If CPUID.OAH: EAX[15:8] > 1
		2	Set 1 to cause Ovf_PMC2 = 1	If CPUID.OAH: EAX[15:8] > 2
		n	Set 1 to cause Ovf_PMCn = 1	If CPUID.OAH: EAX[15:8] > n
		31:n	Reserved.	
		32	Set 1 to cause Ovf_FIXED_CTR0 = 1.	If CPUID.OAH: EAX[7:0] > 3
		33	Set 1 to cause Ovf_FIXED_CTR1 = 1.	If CPUID.OAH: EAX[7:0] > 3
		34	Set 1 to cause Ovf_FIXED_CTR2 = 1.	If CPUID.OAH: EAX[7:0] > 3
		54:35	Reserved.	
		55	Set 1 to cause Trace_ToPA_PMI = 1.	If CPUID.OAH: EAX[7:0] > 3

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		57:56	Reserved.	
		58	Set 1 to cause LBR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		59	Set 1 to cause CTR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		58	Set 1 to cause ASCII = 1.	If CPUID.0AH: EAX[7:0] > 3
		61	Set 1 to cause Ovf_Uncore = 1.	If CPUID.0AH: EAX[7:0] > 3
		62	Set 1 to cause OvfBuf = 1.	If CPUID.0AH: EAX[7:0] > 3
		63	Reserved	
392H	914	IA32_PERF_GLOBAL_INUSE	Indicator of core perfmon interface is in use (RO)	If CPUID.0AH: EAX[7:0] > 3
		0	IA32_PERFEVTSEL0 in use	
		1	IA32_PERFEVTSEL1 in use	If CPUID.0AH: EAX[15:8] > 1
		2	IA32_PERFEVTSEL2 in use	If CPUID.0AH: EAX[15:8] > 2
		n	IA32_PERFEVTSELn in use	If CPUID.0AH: EAX[15:8] > n
		31:n+1	Reserved.	
		32	IA32_FIXED_CTR0 in use	
		33	IA32_FIXED_CTR1 in use	
		34	IA32_FIXED_CTR2 in use	
		62:35	Reserved or Model specific.	
		63	PMI in use.	
3F1H	1009	IA32_PEBS_ENABLE	PEBS Control (R/W)	
		0	Enable PEBS on IA32_PMC0.	06_OFH
		3:1	Reserved or Model specific.	
		31:4	Reserved.	
		35:32	Reserved or Model specific.	
		63:36	Reserved.	
400H	1024	IA32_MCO_CTL	MCO_CTL	If IA32_MCG_CAP.CNT > 0
401H	1025	IA32_MCO_STATUS	MCO_STATUS	If IA32_MCG_CAP.CNT > 0
402H	1026	IA32_MCO_ADDR <sup>1</sup>	MCO_ADDR	If IA32_MCG_CAP.CNT > 0
403H	1027	IA32_MCO_MISC	MCO_MISC	If IA32_MCG_CAP.CNT > 0
404H	1028	IA32_MC1_CTL	MC1_CTL	If IA32_MCG_CAP.CNT > 1
405H	1029	IA32_MC1_STATUS	MC1_STATUS	If IA32_MCG_CAP.CNT > 1
406H	1030	IA32_MC1_ADDR <sup>2</sup>	MC1_ADDR	If IA32_MCG_CAP.CNT > 1
407H	1031	IA32_MC1_MISC	MC1_MISC	If IA32_MCG_CAP.CNT > 1
408H	1032	IA32_MC2_CTL	MC2_CTL	If IA32_MCG_CAP.CNT > 2
409H	1033	IA32_MC2_STATUS	MC2_STATUS	If IA32_MCG_CAP.CNT > 2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
40AH	1034	IA32_MC2_ADDR <sup>1</sup>	MC2_ADDR	If IA32_MCG_CAP.CNT >2
40BH	1035	IA32_MC2_MISC	MC2_MISC	If IA32_MCG_CAP.CNT >2
40CH	1036	IA32_MC3_CTL	MC3_CTL	If IA32_MCG_CAP.CNT >3
40DH	1037	IA32_MC3_STATUS	MC3_STATUS	If IA32_MCG_CAP.CNT >3
40EH	1038	IA32_MC3_ADDR <sup>1</sup>	MC3_ADDR	If IA32_MCG_CAP.CNT >3
40FH	1039	IA32_MC3_MISC	MC3_MISC	If IA32_MCG_CAP.CNT >3
410H	1040	IA32_MC4_CTL	MC4_CTL	If IA32_MCG_CAP.CNT >4
411H	1041	IA32_MC4_STATUS	MC4_STATUS	If IA32_MCG_CAP.CNT >4
412H	1042	IA32_MC4_ADDR <sup>1</sup>	MC4_ADDR	If IA32_MCG_CAP.CNT >4
413H	1043	IA32_MC4_MISC	MC4_MISC	If IA32_MCG_CAP.CNT >4
414H	1044	IA32_MC5_CTL	MC5_CTL	If IA32_MCG_CAP.CNT >5
415H	1045	IA32_MC5_STATUS	MC5_STATUS	If IA32_MCG_CAP.CNT >5
416H	1046	IA32_MC5_ADDR <sup>1</sup>	MC5_ADDR	If IA32_MCG_CAP.CNT >5
417H	1047	IA32_MC5_MISC	MC5_MISC	If IA32_MCG_CAP.CNT >5
418H	1048	IA32_MC6_CTL	MC6_CTL	If IA32_MCG_CAP.CNT >6
419H	1049	IA32_MC6_STATUS	MC6_STATUS	If IA32_MCG_CAP.CNT >6
41AH	1050	IA32_MC6_ADDR <sup>1</sup>	MC6_ADDR	If IA32_MCG_CAP.CNT >6
41BH	1051	IA32_MC6_MISC	MC6_MISC	If IA32_MCG_CAP.CNT >6
41CH	1052	IA32_MC7_CTL	MC7_CTL	If IA32_MCG_CAP.CNT >7
41DH	1053	IA32_MC7_STATUS	MC7_STATUS	If IA32_MCG_CAP.CNT >7
41EH	1054	IA32_MC7_ADDR <sup>1</sup>	MC7_ADDR	If IA32_MCG_CAP.CNT >7
41FH	1055	IA32_MC7_MISC	MC7_MISC	If IA32_MCG_CAP.CNT >7
420H	1056	IA32_MC8_CTL	MC8_CTL	If IA32_MCG_CAP.CNT >8
421H	1057	IA32_MC8_STATUS	MC8_STATUS	If IA32_MCG_CAP.CNT >8
422H	1058	IA32_MC8_ADDR <sup>1</sup>	MC8_ADDR	If IA32_MCG_CAP.CNT >8
423H	1059	IA32_MC8_MISC	MC8_MISC	If IA32_MCG_CAP.CNT >8
424H	1060	IA32_MC9_CTL	MC9_CTL	If IA32_MCG_CAP.CNT >9
425H	1061	IA32_MC9_STATUS	MC9_STATUS	If IA32_MCG_CAP.CNT >9
426H	1062	IA32_MC9_ADDR <sup>1</sup>	MC9_ADDR	If IA32_MCG_CAP.CNT >9
427H	1063	IA32_MC9_MISC	MC9_MISC	If IA32_MCG_CAP.CNT >9
428H	1064	IA32_MC10_CTL	MC10_CTL	If IA32_MCG_CAP.CNT >10
429H	1065	IA32_MC10_STATUS	MC10_STATUS	If IA32_MCG_CAP.CNT >10
42AH	1066	IA32_MC10_ADDR <sup>1</sup>	MC10_ADDR	If IA32_MCG_CAP.CNT >10
42BH	1067	IA32_MC10_MISC	MC10_MISC	If IA32_MCG_CAP.CNT >10
42CH	1068	IA32_MC11_CTL	MC11_CTL	If IA32_MCG_CAP.CNT >11
42DH	1069	IA32_MC11_STATUS	MC11_STATUS	If IA32_MCG_CAP.CNT >11
42EH	1070	IA32_MC11_ADDR <sup>1</sup>	MC11_ADDR	If IA32_MCG_CAP.CNT >11

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
42FH	1071	IA32_MC11_MISC	MC11_MISC	If IA32_MCG_CAP.CNT >11
430H	1072	IA32_MC12_CTL	MC12_CTL	If IA32_MCG_CAP.CNT >12
431H	1073	IA32_MC12_STATUS	MC12_STATUS	If IA32_MCG_CAP.CNT >12
432H	1074	IA32_MC12_ADDR <sup>7</sup>	MC12_ADDR	If IA32_MCG_CAP.CNT >12
433H	1075	IA32_MC12_MISC	MC12_MISC	If IA32_MCG_CAP.CNT >12
434H	1076	IA32_MC13_CTL	MC13_CTL	If IA32_MCG_CAP.CNT >13
435H	1077	IA32_MC13_STATUS	MC13_STATUS	If IA32_MCG_CAP.CNT >13
436H	1078	IA32_MC13_ADDR <sup>7</sup>	MC13_ADDR	If IA32_MCG_CAP.CNT >13
437H	1079	IA32_MC13_MISC	MC13_MISC	If IA32_MCG_CAP.CNT >13
438H	1080	IA32_MC14_CTL	MC14_CTL	If IA32_MCG_CAP.CNT >14
439H	1081	IA32_MC14_STATUS	MC14_STATUS	If IA32_MCG_CAP.CNT >14
43AH	1082	IA32_MC14_ADDR <sup>7</sup>	MC14_ADDR	If IA32_MCG_CAP.CNT >14
43BH	1083	IA32_MC14_MISC	MC14_MISC	If IA32_MCG_CAP.CNT >14
43CH	1084	IA32_MC15_CTL	MC15_CTL	If IA32_MCG_CAP.CNT >15
43DH	1085	IA32_MC15_STATUS	MC15_STATUS	If IA32_MCG_CAP.CNT >15
43EH	1086	IA32_MC15_ADDR <sup>7</sup>	MC15_ADDR	If IA32_MCG_CAP.CNT >15
43FH	1087	IA32_MC15_MISC	MC15_MISC	If IA32_MCG_CAP.CNT >15
440H	1088	IA32_MC16_CTL	MC16_CTL	If IA32_MCG_CAP.CNT >16
441H	1089	IA32_MC16_STATUS	MC16_STATUS	If IA32_MCG_CAP.CNT >16
442H	1090	IA32_MC16_ADDR <sup>7</sup>	MC16_ADDR	If IA32_MCG_CAP.CNT >16
443H	1091	IA32_MC16_MISC	MC16_MISC	If IA32_MCG_CAP.CNT >16
444H	1092	IA32_MC17_CTL	MC17_CTL	If IA32_MCG_CAP.CNT >17
445H	1093	IA32_MC17_STATUS	MC17_STATUS	If IA32_MCG_CAP.CNT >17
446H	1094	IA32_MC17_ADDR <sup>7</sup>	MC17_ADDR	If IA32_MCG_CAP.CNT >17
447H	1095	IA32_MC17_MISC	MC17_MISC	If IA32_MCG_CAP.CNT >17
448H	1096	IA32_MC18_CTL	MC18_CTL	If IA32_MCG_CAP.CNT >18
449H	1097	IA32_MC18_STATUS	MC18_STATUS	If IA32_MCG_CAP.CNT >18
44AH	1098	IA32_MC18_ADDR <sup>7</sup>	MC18_ADDR	If IA32_MCG_CAP.CNT >18
44BH	1099	IA32_MC18_MISC	MC18_MISC	If IA32_MCG_CAP.CNT >18
44CH	1100	IA32_MC19_CTL	MC19_CTL	If IA32_MCG_CAP.CNT >19
44DH	1101	IA32_MC19_STATUS	MC19_STATUS	If IA32_MCG_CAP.CNT >19
44EH	1102	IA32_MC19_ADDR <sup>7</sup>	MC19_ADDR	If IA32_MCG_CAP.CNT >19
44FH	1103	IA32_MC19_MISC	MC19_MISC	If IA32_MCG_CAP.CNT >19
450H	1104	IA32_MC20_CTL	MC20_CTL	If IA32_MCG_CAP.CNT >20
451H	1105	IA32_MC20_STATUS	MC20_STATUS	If IA32_MCG_CAP.CNT >20
452H	1106	IA32_MC20_ADDR <sup>7</sup>	MC20_ADDR	If IA32_MCG_CAP.CNT >20
453H	1107	IA32_MC20_MISC	MC20_MISC	If IA32_MCG_CAP.CNT >20



Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
454H	1108	IA32_MC21_CTL	MC21_CTL	If IA32_MCG_CAP.CNT >21
455H	1109	IA32_MC21_STATUS	MC21_STATUS	If IA32_MCG_CAP.CNT >21
456H	1110	IA32_MC21_ADDR <sup>1</sup>	MC21_ADDR	If IA32_MCG_CAP.CNT >21
457H	1111	IA32_MC21_MISC	MC21_MISC	If IA32_MCG_CAP.CNT >21
458H		IA32_MC22_CTL	MC22_CTL	If IA32_MCG_CAP.CNT >22
459H		IA32_MC22_STATUS	MC22_STATUS	If IA32_MCG_CAP.CNT >22
45AH		IA32_MC22_ADDR <sup>1</sup>	MC22_ADDR	If IA32_MCG_CAP.CNT >22
45BH		IA32_MC22_MISC	MC22_MISC	If IA32_MCG_CAP.CNT >22
45CH		IA32_MC23_CTL	MC23_CTL	If IA32_MCG_CAP.CNT >23
45DH		IA32_MC23_STATUS	MC23_STATUS	If IA32_MCG_CAP.CNT >23
45EH		IA32_MC23_ADDR <sup>1</sup>	MC23_ADDR	If IA32_MCG_CAP.CNT >23
45FH		IA32_MC23_MISC	MC23_MISC	If IA32_MCG_CAP.CNT >23
460H		IA32_MC24_CTL	MC24_CTL	If IA32_MCG_CAP.CNT >24
461H		IA32_MC24_STATUS	MC24_STATUS	If IA32_MCG_CAP.CNT >24
462H		IA32_MC24_ADDR <sup>1</sup>	MC24_ADDR	If IA32_MCG_CAP.CNT >24
463H		IA32_MC24_MISC	MC24_MISC	If IA32_MCG_CAP.CNT >24
464H		IA32_MC25_CTL	MC25_CTL	If IA32_MCG_CAP.CNT >25
465H		IA32_MC25_STATUS	MC25_STATUS	If IA32_MCG_CAP.CNT >25
466H		IA32_MC25_ADDR <sup>1</sup>	MC25_ADDR	If IA32_MCG_CAP.CNT >25
467H		IA32_MC25_MISC	MC25_MISC	If IA32_MCG_CAP.CNT >25
468H		IA32_MC26_CTL	MC26_CTL	If IA32_MCG_CAP.CNT >26
469H		IA32_MC26_STATUS	MC26_STATUS	If IA32_MCG_CAP.CNT >26
46AH		IA32_MC26_ADDR <sup>1</sup>	MC26_ADDR	If IA32_MCG_CAP.CNT >26
46BH		IA32_MC26_MISC	MC26_MISC	If IA32_MCG_CAP.CNT >26
46CH		IA32_MC27_CTL	MC27_CTL	If IA32_MCG_CAP.CNT >27
46DH		IA32_MC27_STATUS	MC27_STATUS	If IA32_MCG_CAP.CNT >27
46EH		IA32_MC27_ADDR <sup>1</sup>	MC27_ADDR	If IA32_MCG_CAP.CNT >27
46FH		IA32_MC27_MISC	MC27_MISC	If IA32_MCG_CAP.CNT >27
470H		IA32_MC28_CTL	MC28_CTL	If IA32_MCG_CAP.CNT >28
471H		IA32_MC28_STATUS	MC28_STATUS	If IA32_MCG_CAP.CNT >28
472H		IA32_MC28_ADDR <sup>1</sup>	MC28_ADDR	If IA32_MCG_CAP.CNT >28
473H		IA32_MC28_MISC	MC28_MISC	If IA32_MCG_CAP.CNT >28
480H	1152	IA32_VMX_BASIC	<b>Reporting Register of Basic VMX Capabilities (R/O)</b> See Appendix A.1, "Basic VMX Information."	If CPUID.01H:ECX.[5] = 1

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
481H	1153	IA32_VMX_PINBASED_CTL5	<b>Capability Reporting Register of Pin-based VM-execution Controls (R/O)</b> See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If CPUID.01H:ECX.[5] = 1
482H	1154	IA32_VMX_PROCBASED_CTL5	<b>Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If CPUID.01H:ECX.[5] = 1
483H	1155	IA32_VMX_EXIT_CTL5	<b>Capability Reporting Register of VM-exit Controls (R/O)</b> See Appendix A.4, "VM-Exit Controls."	If CPUID.01H:ECX.[5] = 1
484H	1156	IA32_VMX_ENTRY_CTL5	<b>Capability Reporting Register of VM-entry Controls (R/O)</b> See Appendix A.5, "VM-Entry Controls."	If CPUID.01H:ECX.[5] = 1
485H	1157	IA32_VMX_MISC	<b>Reporting Register of Miscellaneous VMX Capabilities (R/O)</b> See Appendix A.6, "Miscellaneous Data."	If CPUID.01H:ECX.[5] = 1
486H	1158	IA32_VMX_CRO_FIXED0	<b>Capability Reporting Register of CRO Bits Fixed to 0 (R/O)</b> See Appendix A.7, "VMX-Fixed Bits in CRO."	If CPUID.01H:ECX.[5] = 1
487H	1159	IA32_VMX_CRO_FIXED1	<b>Capability Reporting Register of CRO Bits Fixed to 1 (R/O)</b> See Appendix A.7, "VMX-Fixed Bits in CRO."	If CPUID.01H:ECX.[5] = 1
488H	1160	IA32_VMX_CR4_FIXED0	<b>Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)</b> See Appendix A.8, "VMX-Fixed Bits in CR4."	If CPUID.01H:ECX.[5] = 1
489H	1161	IA32_VMX_CR4_FIXED1	<b>Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)</b> See Appendix A.8, "VMX-Fixed Bits in CR4."	If CPUID.01H:ECX.[5] = 1
48AH	1162	IA32_VMX_VMCS_ENUM	<b>Capability Reporting Register of VMCS Field Enumeration (R/O)</b> See Appendix A.9, "VMCS Enumeration."	If CPUID.01H:ECX.[5] = 1
48BH	1163	IA32_VMX_PROCBASED_CTL52	<b>Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3.3, "Secondary Processor-Based VM-Execution Controls."	If ( CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTL5[63])
48CH	1164	IA32_VMX_EPT_VPID_CAP	<b>Capability Reporting Register of EPT and VPID (R/O)</b> See Appendix A.10, "VPID and EPT Capabilities."	If ( CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTL5[63] && ( IA32_VMX_PROCBASED_CTL52[33]    IA32_VMX_PROCBASED_CTL52[37]) )

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
48DH	1165	IA32_VMX_TRUE_PINBASED_CTL	<b>Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)</b> See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If ( CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55] )
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTL	<b>Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)</b> See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If ( CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55] )
48FH	1167	IA32_VMX_TRUE_EXIT_CTL	<b>Capability Reporting Register of VM-exit Flex Controls (R/O)</b> See Appendix A.4, "VM-Exit Controls."	If ( CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55] )
490H	1168	IA32_VMX_TRUE_ENTRY_CTL	<b>Capability Reporting Register of VM-entry Flex Controls (R/O)</b> See Appendix A.5, "VM-Entry Controls."	If ( CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55] )
491H	1169	IA32_VMX_VMFUNC	<b>Capability Reporting Register of VM-function Controls (R/O)</b>	If ( CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55] )
4C1H	1217	IA32_A_PMC0	Full Width Writable IA32_PMC0 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 0) && IA32_PERF_CAPABILITIES[13] = 1
4C2H	1218	IA32_A_PMC1	Full Width Writable IA32_PMC1 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 1) && IA32_PERF_CAPABILITIES[13] = 1
4C3H	1219	IA32_A_PMC2	Full Width Writable IA32_PMC2 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 2) && IA32_PERF_CAPABILITIES[13] = 1
4C4H	1220	IA32_A_PMC3	Full Width Writable IA32_PMC3 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 3) && IA32_PERF_CAPABILITIES[13] = 1
4C5H	1221	IA32_A_PMC4	Full Width Writable IA32_PMC4 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 4) && IA32_PERF_CAPABILITIES[13] = 1
4C6H	1222	IA32_A_PMC5	Full Width Writable IA32_PMC5 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 5) && IA32_PERF_CAPABILITIES[13] = 1

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
4C7H	1223	IA32_A_PMC6	Full Width Writable IA32_PMC6 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 6) && IA32_PERF_CAPABILITIES[13] = 1
4C8H	1224	IA32_A_PMC7	Full Width Writable IA32_PMC7 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 7) && IA32_PERF_CAPABILITIES[13] = 1
4D0H	1232	IA32_MCG_EXT_CTL	Allows software to signal some MCEs to only single logical processor in the system. (R/W) See Section 15.3.1.4, "IA32_MCG_EXT_CTL MSR".	If IA32_MCG_CAP.LMCE_P = 1
		0	LMCE_EN.	
		63:1	Reserved.	
500H	1280	IA32_SGX_SVN_STATUS	Status and SVN Threshold of SGX Support for ACM (RO).	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		0	Lock.	See Section 41.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1	Reserved.	
		23:16	SGX_SVN_SINIT.	See Section 41.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		63:24	Reserved.	
560H	1376	IA32_RTIT_OUTPUT_BASE	<b>Trace Output Base Register (R/W)</b>	If ((CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && (CPUID.(EAX=14H,ECX=0):ECX[0] = 1)    (CPUID.(EAX=14H,ECX=0):ECX[2] = 1) )
		6:0	Reserved	
		MAXPHYADDR <sup>3</sup> -1:7	Base physical address	
		63:MAXPHYADDR	<b>Reserved.</b>	
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	<b>Trace Output Mask Pointers Register (R/W)</b>	If ((CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && (CPUID.(EAX=14H,ECX=0):ECX[0] = 1)    (CPUID.(EAX=14H,ECX=0):ECX[2] = 1) )
		6:0	Reserved	
		31:7	MaskOrTableOffset	
		63:32	<b>Output Offset.</b>	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
570H	1392	IA32_RTIT_CTL	<b>Trace Control Register (R/W)</b>	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		0	<b>TraceEn</b>	
		1	<b>CYCEn</b>	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		2	<b>OS</b>	
		3	<b>User</b>	
		4	<b>PwrEvtEn</b>	
		5	<b>FUPonPTW</b>	
		6	<b>FabricEn</b>	If (CPUID.(EAX=07H, ECX=0):ECX[3] = 1)
		7	<b>CR3 filter</b>	
		8	<b>ToPA</b>	
		9	<b>MTCEn</b>	If (CPUID.(EAX=07H, ECX=0):EBX[3] = 1)
		10	<b>TSCEn</b>	
		11	<b>DisRETC</b>	
		12	<b>PTWEn</b>	
		13	<b>BranchEn</b>	
		17:14	<b>MTCFreq</b>	If (CPUID.(EAX=07H, ECX=0):EBX[3] = 1)
		18	Reserved, MBZ	
		22:19	<b>CYCThresh</b>	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		23	Reserved, MBZ	
		27:24	<b>PSBFreq</b>	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		31:28	Reserved, MBZ	
35:32	<b>ADDR0_CFG</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)		
39:36	<b>ADDR1_CFG</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)		
43:40	<b>ADDR2_CFG</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)		
47:44	<b>ADDR3_CFG</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)		
63:48	Reserved, MBZ.			
571H	1393	IA32_RTIT_STATUS	<b>Tracing Status Register (R/W)</b>	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		0	<b>FilterEn</b> (writes ignored)	If (CPUID.(EAX=07H, ECX=0):EBX[2] = 1)

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		1	<b>ContexEn</b> (writes ignored)	
		2	<b>TriggerEn</b> (writes ignored)	
		3	Reserved	
		4	<b>Error</b>	
		5	<b>Stopped</b>	
		31:6	Reserved, MBZ	
		48:32	<b>PacketByteCnt</b>	If (CPUID.(EAX=07H, ECX=0):EBX[1] > 3)
		63:49	<b>Reserved</b>	
572H	1394	IA32_RTIT_CR3_MATCH	<b>Trace Filter CR3 Match Register (R/W)</b>	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		4:0	Reserved	
		63:5	CR3[63:5] value to match	
580H	1408	IA32_RTIT_ADDR0_A	<b>Region 0 Start Address (R/W)</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
581H	1409	IA32_RTIT_ADDR0_B	<b>Region 0 End Address (R/W)</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
582H	1410	IA32_RTIT_ADDR1_A	<b>Region 1 Start Address (R/W)</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)
		47:0	Virtual Address	
		63:48	SignExt_VA	
583H	1411	IA32_RTIT_ADDR1_B	<b>Region 1 End Address (R/W)</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)
		47:0	Virtual Address	
		63:48	SignExt_VA	
584H	1412	IA32_RTIT_ADDR2_A	<b>Region 2 Start Address (R/W)</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:0	Virtual Address	
		63:48	SignExt_VA	
585H	1413	IA32_RTIT_ADDR2_B	<b>Region 2 End Address (R/W)</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:0	Virtual Address	
		63:48	SignExt_VA	
586H	1414	IA32_RTIT_ADDR3_A	<b>Region 3 Start Address (R/W)</b>	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		47:0	Virtual Address	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:48	SignExt_VA	
587H	1415	IA32_RTIT_ADDR3_B	<b>Region 3 End Address (R/W)</b>	If (CPUID.(EAX=07H, ECX=1);EAX[2:0] > 3)
		47:0	Virtual Address	
		63:48	SignExt_VA	
600H	1536	IA32_DS_AREA	<b>DS Save Area (R/W)</b> Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 18.6.3.4, "Debug Store (DS) Mechanism."	If (CPUID.01H:EDX.DS[21] = 1)
		63:0	The linear address of the first byte of the DS buffer management area, if IA-32e mode is active.	
		31:0	The linear address of the first byte of the DS buffer management area, if not in IA-32e mode.	
		63:32	Reserved if not in IA-32e mode.	
6E0H	1760	IA32_TSC_DEADLINE	<b>TSC Target of Local APIC's TSC Deadline Mode (R/W)</b>	If CPUID.01H:ECX.[24] = 1
770H	1904	IA32_PM_ENABLE	<b>Enable/disable HWP (R/W)</b>	If CPUID.06H:EAX.[7] = 1
		0	<b>HWP_ENABLE (R/W1-Once)</b> See Section 14.4.2, "Enabling HWP"	If CPUID.06H:EAX.[7] = 1
		63:1	Reserved.	
771H	1905	IA32_HWP_CAPABILITIES	<b>HWP Performance Range Enumeration (RO)</b>	If CPUID.06H:EAX.[7] = 1
		7:0	<b>Highest_Performance</b> See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"	If CPUID.06H:EAX.[7] = 1
		15:8	<b>Guaranteed_Performance</b> See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"	If CPUID.06H:EAX.[7] = 1
		23:16	<b>Most_Efficient_Performance</b> See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"	If CPUID.06H:EAX.[7] = 1
		31:24	<b>Lowest_Performance</b> See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"	If CPUID.06H:EAX.[7] = 1
		63:32	Reserved.	
772H	1906	IA32_HWP_REQUEST_PKG	<b>Power Management Control Hints for All Logical Processors in a Package (R/W)</b>	If CPUID.06H:EAX.[11] = 1

**Table 2-2. IA-32 Architectural MSRs (Contd.)**

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		7:0	<b>Minimum_Performance</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[11] = 1
		15:8	<b>Maximum_Performance</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[11] = 1
		23:16	<b>Desired_Performance</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[11] = 1
		31:24	<b>Energy_Performance_Preference</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[11] = 1 && CPUID.06H:EAX.[10] = 1
		41:32	<b>Activity_Window</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[11] = 1 && CPUID.06H:EAX.[9] = 1
		63:42	Reserved.	
773H	1907	IA32_HWP_INTERRUPT	<b>Control HWP Native Interrupts (R/W)</b>	If CPUID.06H:EAX.[8] = 1
		0	<b>EN_Guaranteed_Performance_Change</b> See Section 14.4.6, "HWP Notifications"	If CPUID.06H:EAX.[8] = 1
		1	<b>EN_Excursion_Minimum</b> See Section 14.4.6, "HWP Notifications"	If CPUID.06H:EAX.[8] = 1
		63:2	Reserved.	
774H	1908	IA32_HWP_REQUEST	<b>Power Management Control Hints to a Logical Processor (R/W)</b>	If CPUID.06H:EAX.[7] = 1
		7:0	<b>Minimum_Performance</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[7] = 1
		15:8	<b>Maximum_Performance</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[7] = 1
		23:16	<b>Desired_Performance</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[7] = 1
		31:24	<b>Energy_Performance_Preference</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[10] = 1
		41:32	<b>Activity_Window</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[9] = 1
		42	<b>Package_Control</b> See Section 14.4.4, "Managing HWP"	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[11] = 1
		63:43	Reserved.	
777H	1911	IA32_HWP_STATUS	<b>Log bits indicating changes to Guaranteed &amp; excursions to Minimum (R/W)</b>	If CPUID.06H:EAX.[7] = 1



Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	<b>Guaranteed_Performance_Change (R/WCO)</b> See Section 14.4.5, "HWP Feedback"	If CPUID.06H:EAX.[7] = 1
		1	Reserved.	
		2	<b>Excursion_To_Minimum (R/WCO)</b> See Section 14.4.5, "HWP Feedback"	If CPUID.06H:EAX.[7] = 1
		63:3	Reserved.	
802H	2050	IA32_X2APIC_APICID	<b>x2APIC ID Register (R/O)</b> See x2APIC Specification	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
803H	2051	IA32_X2APIC_VERSION	<b>x2APIC Version Register (R/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
808H	2056	IA32_X2APIC_TPR	<b>x2APIC Task Priority Register (R/W)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80AH	2058	IA32_X2APIC_PPR	<b>x2APIC Processor Priority Register (R/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80BH	2059	IA32_X2APIC_EOI	<b>x2APIC EOI Register (W/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80DH	2061	IA32_X2APIC_LDR	<b>x2APIC Logical Destination Register (R/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80FH	2063	IA32_X2APIC_SIVR	<b>x2APIC Spurious Interrupt Vector Register (R/W)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
810H	2064	IA32_X2APIC_ISR0	<b>x2APIC In-Service Register Bits 31:0 (R/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
811H	2065	IA32_X2APIC_ISR1	<b>x2APIC In-Service Register Bits 63:32 (R/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
812H	2066	IA32_X2APIC_ISR2	<b>x2APIC In-Service Register Bits 95:64 (R/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
813H	2067	IA32_X2APIC_ISR3	<b>x2APIC In-Service Register Bits 127:96 (R/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
814H	2068	IA32_X2APIC_ISR4	<b>x2APIC In-Service Register Bits 159:128 (R/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
815H	2069	IA32_X2APIC_ISR5	x2APIC In-Service Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
816H	2070	IA32_X2APIC_ISR6	x2APIC In-Service Register Bits 223:192 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
817H	2071	IA32_X2APIC_ISR7	x2APIC In-Service Register Bits 255:224 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
818H	2072	IA32_X2APIC_TMR0	x2APIC Trigger Mode Register Bits 31:0 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
819H	2073	IA32_X2APIC_TMR1	x2APIC Trigger Mode Register Bits 63:32 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81AH	2074	IA32_X2APIC_TMR2	x2APIC Trigger Mode Register Bits 95:64 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81BH	2075	IA32_X2APIC_TMR3	x2APIC Trigger Mode Register Bits 127:96 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81CH	2076	IA32_X2APIC_TMR4	x2APIC Trigger Mode Register Bits 159:128 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81DH	2077	IA32_X2APIC_TMR5	x2APIC Trigger Mode Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81EH	2078	IA32_X2APIC_TMR6	x2APIC Trigger Mode Register Bits 223:192 (R/O)	If (CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1)
81FH	2079	IA32_X2APIC_TMR7	x2APIC Trigger Mode Register Bits 255:224 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
820H	2080	IA32_X2APIC_IRR0	x2APIC Interrupt Request Register Bits 31:0 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
821H	2081	IA32_X2APIC_IRR1	x2APIC Interrupt Request Register Bits 63:32 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
822H	2082	IA32_X2APIC_IRR2	x2APIC Interrupt Request Register Bits 95:64 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
823H	2083	IA32_X2APIC_IRR3	x2APIC Interrupt Request Register Bits 127:96 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
824H	2084	IA32_X2APIC_IJR4	x2APIC Interrupt Request Register Bits 159:128 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
825H	2085	IA32_X2APIC_IJR5	x2APIC Interrupt Request Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
826H	2086	IA32_X2APIC_IJR6	x2APIC Interrupt Request Register Bits 223:192 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
827H	2087	IA32_X2APIC_IJR7	x2APIC Interrupt Request Register Bits 255:224 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
828H	2088	IA32_X2APIC_ESR	x2APIC Error Status Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
82FH	2095	IA32_X2APIC_LVT_CMCI	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
830H	2096	IA32_X2APIC_ICR	x2APIC Interrupt Command Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
832H	2098	IA32_X2APIC_LVT_TIMER	x2APIC LVT Timer Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
833H	2099	IA32_X2APIC_LVT_THERMAL	x2APIC LVT Thermal Sensor Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
834H	2100	IA32_X2APIC_LVT_PMI	x2APIC LVT Performance Monitor Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
835H	2101	IA32_X2APIC_LVT_LINT0	x2APIC LVT LINT0 Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
836H	2102	IA32_X2APIC_LVT_LINT1	x2APIC LVT LINT1 Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
837H	2103	IA32_X2APIC_LVT_ERROR	x2APIC LVT Error Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
838H	2104	IA32_X2APIC_INIT_COUNT	x2APIC Initial Count Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
839H	2105	IA32_X2APIC_CUR_COUNT	x2APIC Current Count Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
83EH	2110	IA32_X2APIC_DIV_CONF	<b>x2APIC Divide Configuration Register (R/W)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
83FH	2111	IA32_X2APIC_SELF_IPI	<b>x2APIC Self IPI Register (W/O)</b>	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
C80H	3200	IA32_DEBUG_INTERFACE	<b>Silicon Debug Feature Control (R/W)</b>	If CPUID.01H:ECX.[11] = 1
		0	<b>Enable (R/W)</b> BIOS set 1 to enable Silicon debug features. Default is 0	If CPUID.01H:ECX.[11] = 1
		29:1	Reserved.	
		30	<b>Lock (R/W):</b> If 1, locks any further change to the MSR. The lock bit is set automatically on the first SMI assertion even if not explicitly set by BIOS. Default is 0.	If CPUID.01H:ECX.[11] = 1
		31	<b>Debug Occurred (R/O):</b> This “sticky bit” is set by hardware to indicate the status of bit 0. Default is 0.	If CPUID.01H:ECX.[11] = 1
		63:32	Reserved.	
C81H	3201	IA32_L3_QOS_CFG	<b>L3 QOS Configuration (R/W)</b>	If ( CPUID.(EAX=10H, ECX=1):ECX.[2] = 1 )
		0	<b>Enable (R/W)</b> Set 1 to enable L3 CAT masks and COS to operate in Code and Data Prioritization (CDP) mode	
		63:1	Reserved.	
C8DH	3213	IA32_QM_EVTSEL	<b>Monitoring Event Select Register (R/W)</b>	If ( CPUID.(EAX=07H, ECX=0):EBX.[12] = 1 )
		7:0	<b>Event ID:</b> ID of a supported monitoring event to report via IA32_QM_CTR.	
		31: 8	<b>Reserved.</b>	
		N+31:32	<b>Resource Monitoring ID:</b> ID for monitoring hardware to report monitored data via IA32_QM_CTR.	N = Ceil (Log <sub>2</sub> ( CPUID.(EAX= 0FH, ECX=0H).EBX[31:0] +1))
		63:N+32	<b>Reserved.</b>	
C8EH	3214	IA32_QM_CTR	<b>Monitoring Counter Register (R/O)</b>	If ( CPUID.(EAX=07H, ECX=0):EBX.[12] = 1 )
		61:0	<b>Resource Monitored Data</b>	
		62	<b>Unavailable:</b> If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
		63	<b>Error:</b> If 1, indicates and unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
C8FH	3215	IA32_PQR_ASSOC	<b>Resource Association Register (R/W)</b>	If ( (CPUID.(EAX=07H, ECX=0);EBX[12] = 1) or (CPUID.(EAX=07H, ECX=0);EBX[15] = 1) )
		N-1:0	<b>Resource Monitoring ID (R/W):</b> ID for monitoring hardware to track internal operation, e.g. memory access.	N = Ceil (Log <sub>2</sub> ( CPUID.(EAX= 0FH, ECX=0H).EBX[31:0] +1))
		31:N	<b>Reserved</b>	
		63:32	<b>COS (R/W).</b> The class of service (COS) to enforce (on writes); returns the current COS when read.	If ( CPUID.(EAX=07H, ECX=0);EBX.[15] = 1 )
C90H - D8FH		Reserved MSR Address Space for CAT Mask Registers	See Section 17.19.4.1, "Enumeration and Detection Support of Cache Allocation Technology".	
C90H	3216	IA32_L3_MASK_0	<b>L3 CAT Mask for COS0 (R/W)</b>	If (CPUID.(EAX=10H, ECX=0H);EBX[1] != 0)
		31:0	<b>Capacity Bit Mask (R/W)</b>	
		63:32	Reserved.	
C90H+n	3216+n	IA32_L3_MASK_n	<b>L3 CAT Mask for COSn (R/W)</b>	n = CPUID.(EAX=10H, ECX=1H);EDX[15:0]
		31:0	<b>Capacity Bit Mask (R/W)</b>	
		63:32	Reserved.	
D10H - D4FH		Reserved MSR Address Space for L2 CAT Mask Registers	See Section 17.19.4.1, "Enumeration and Detection Support of Cache Allocation Technology".	
D10H	3344	IA32_L2_MASK_0	<b>L2 CAT Mask for COS0 (R/W)</b>	If (CPUID.(EAX=10H, ECX=0H);EBX[2] != 0)
		31:0	<b>Capacity Bit Mask (R/W)</b>	
		63:32	Reserved.	
D10H+n	3344+n	IA32_L2_MASK_n	<b>L2 CAT Mask for COSn (R/W)</b>	n = CPUID.(EAX=10H, ECX=2H);EDX[15:0]
		31:0	<b>Capacity Bit Mask (R/W)</b>	
		63:32	Reserved.	
D90H	3472	IA32_BNDCFGS	<b>Supervisor State of MPX Configuration. (R/W)</b>	If (CPUID.(EAX=07H, ECX=0H);EBX[14] = 1)
		0	<b>EN:</b> Enable Intel MPX in supervisor mode	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		1	<b>BNDPRESERVE:</b> Preserve the bounds registers for near branch instructions in the absence of the BND prefix	
		11:2	Reserved, must be 0	
		63:12	<b>Base Address of Bound Directory.</b>	
DA0H	3488	IA32_XSS	<b>Extended Supervisor State Mask (R/W)</b>	If (CPUID.(0DH, 1):EAX.[3] = 1
		7:0	<b>Reserved</b>	
		8	<b>Trace Packet Configuration State (R/W)</b>	
		63:9	Reserved.	
DB0H	3504	IA32_PKG_HDC_CTL	<b>Package Level Enable/disable HDC (R/W)</b>	If CPUID.06H:EAX.[13] = 1
		0	<b>HDC_Pkg_Enable (R/W)</b> Force HDC idling or wake up HDC-idled logical processors in the package. See Section 14.5.2, "Package level Enabling HDC"	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved.	
DB1H	3505	IA32_PM_CTL1	<b>Enable/disable HWP (R/W)</b>	If CPUID.06H:EAX.[13] = 1
		0	<b>HDC_Allow_Block (R/W)</b> Allow/Block this logical processor for package level HDC control. See Section 14.5.3	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved.	
DB2H	3506	IA32_THREAD_STALL	<b>Per-Logical_Processor HDC Idle Residency (R/O)</b>	If CPUID.06H:EAX.[13] = 1
		63:0	<b>Stall_Cycle_Cnt (R/W)</b> Stalled cycles due to HDC forced idle on this logical processor. See Section 14.5.4.1	If CPUID.06H:EAX.[13] = 1
4000_0000H - 4000_00FFH		Reserved MSR Address Space	All existing and future processors will not implement MSR in this range.	
C000_0080H		IA32_EFER	<b>Extended Feature Enables</b>	If (CPUID.80000001H:EDX.[20]    CPUID.80000001H:EDX.[29])
		0	<b>SYSCALL Enable: IA32_EFER.SCE (R/W)</b> Enables SYSCALL/SYSRET instructions in 64-bit mode.	
		7:1	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		8	<b>IA-32e Mode Enable: IA32_EFER.LME (R/W)</b> Enables IA-32e mode operation.	
		9	Reserved.	
		10	<b>IA-32e Mode Active: IA32_EFER.LMA (R)</b> Indicates IA-32e mode is active when set.	
		11	<b>Execute Disable Bit Enable: IA32_EFER.NXE (R/W)</b>	
		63:12	Reserved.	
C000_0081H		IA32_STAR	<b>System Call Target Address (R/W)</b>	If CPUID.80000001:EDX.[29] = 1
C000_0082H		IA32_LSTAR	<b>IA-32e Mode System Call Target Address (R/W)</b> Target RIP for the called procedure when SYSCALL is executed in 64-bit mode.	If CPUID.80000001:EDX.[29] = 1
C000_0083H		IA32_CSTAR	<b>IA-32e Mode System Call Target Address (R/W)</b> Not used, as the SYSCALL instruction is not recognized in compatibility mode.	If CPUID.80000001:EDX.[29] = 1
C000_0084H		IA32_FMASK	<b>System Call Flag Mask (R/W)</b>	If CPUID.80000001:EDX.[29] = 1
C000_0100H		IA32_FS_BASE	<b>Map of BASE Address of FS (R/W)</b>	If CPUID.80000001:EDX.[29] = 1
C000_0101H		IA32_GS_BASE	<b>Map of BASE Address of GS (R/W)</b>	If CPUID.80000001:EDX.[29] = 1
C000_0102H		IA32_KERNEL_GS_BASE	<b>Swap Target of BASE Address of GS (R/W)</b>	If CPUID.80000001:EDX.[29] = 1
C000_0103H		IA32_TSC_AUX	Auxiliary TSC (RW)	If CPUID.80000001H:EDX[27] = 1
		31:0	AUX: Auxiliary signature of TSC	
		63:32	Reserved.	

**NOTES:**

1. In processors based on Intel NetBurst® microarchitecture, MSR addresses 180H-197H are supported, software must treat them as model-specific. Starting with Intel Core Duo processors, MSR addresses 180H-185H, 188H-197H are reserved.
2. The \*\_ADDR MSRs may or may not be present; this depends on flag settings in IA32\_MCI\_STATUS. See Section 15.3.2.3 and Section 15.3.2.4 for more information.
3. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].

## 2.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table 2-3 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 2-3. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_0FH, see Table 2-1.

MSRs listed in Table 2-2 and Table 2-3 are also supported by processors based on the Enhanced Intel Core 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 2-3. 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 2.22, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Unique	See Section 2.22, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.17, “Time-Stamp Counter,” and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	<b>Platform ID (R)</b> See Table 2-2.
17H	23	MSR_PLATFORM_ID	Shared	<b>Model Specific Platform ID (R)</b>
		7:0		Reserved.
		12:8		<b>Maximum Qualified Ratio (R)</b> The maximum allowed bus ratio.
		49:13		Reserved.
		52:50		See Table 2-2.
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, “Local APIC Status and Location,” and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	<b>Processor Hard Power-On Configuration (R/W)</b> Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved.
		1		<b>Data Error Checking Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		2		<b>Response Error Checking Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		<b>MCERR# Drive Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.



Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		4		<b>Address Parity Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		5		Reserved.
		6		Reserved.
		7		<b>BINIT# Driver Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		<b>Output Tri-state Enabled (R/O)</b> 1 = Enabled; 0 = Disabled
		9		<b>Execute BIST (R/O)</b> 1 = Enabled; 0 = Disabled
		10		<b>MCERR# Observation Enabled (R/O)</b> 1 = Enabled; 0 = Disabled
		11		Intel TXT Capable Chipset. (R/O) 1 = Present; 0 = Not Present
		12		<b>BINIT# Observation Enabled (R/O)</b> 1 = Enabled; 0 = Disabled
		13		Reserved.
		14		<b>1 MByte Power on Reset Vector (R/O)</b> 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved.
		17:16		<b>APIC Cluster ID (R/O)</b>
		18		<b>N/2 Non-Integer Bus Ratio (R/O)</b> 0 = Integer ratio; 1 = Non-integer ratio
		19		Reserved.
		21:20		<b>Symmetric Arbitration ID (R/O)</b>
		26:22		<b>Integer Bus Frequency Ratio (R/O)</b>
3AH	58	MSR_FEATURE_CONTROL	Unique	<b>Control Features in Intel 64Processor (R/W)</b> See Table 2-2.
		3	Unique	<b>SMRR Enable (R/WL)</b> 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 2-3. 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	<b>Last Branch Record 0 From IP (R/W)</b> One of four pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the <b>source instruction</b> . See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.5</li> </ul>
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	<b>Last Branch Record 1 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	<b>Last Branch Record 2 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	<b>Last Branch Record 3 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	<b>Last Branch Record 0 To IP (R/W)</b> One of four pairs of last branch record registers on the last branch record stack. This To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	<b>Last Branch Record 1 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	<b>Last Branch Record 2 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	<b>Last Branch Record 3 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	<b>BIOS Update Trigger Register (W)</b> See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	<b>BIOS Update Signature ID (RO)</b> See Table 2-2.
A0H	160	MSR_SMRR_PHYSBASE	Unique	<b>System Management Mode Base Address register (WO in SMM)</b> Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		11:0		Reserved.
		31:12		PhysBase. SMRR physical Base Address.
		63:32		Reserved.
A1H	161	MSR_SMRR_PHYSMASK	Unique	<b>System Management Mode Physical Address Mask register (WO in SMM)</b> 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.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		63:32		Reserved.
C1H	193	IA32_PMC0	Unique	<b>Performance Counter Register</b> See Table 2-2.
C2H	194	IA32_PMC1	Unique	<b>Performance Counter Register</b> See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	<b>Scaleable Bus Speed(R0)</b> This field indicates the intended scaleable bus clock speed for processors based on Intel Core microarchitecture:
		2:0		<ul style="list-style-type: none"> <li>▪ 101B: 100 MHz (FSB 400)</li> <li>▪ 001B: 133 MHz (FSB 533)</li> <li>▪ 011B: 167 MHz (FSB 667)</li> <li>▪ 010B: 200 MHz (FSB 800)</li> <li>▪ 000B: 267 MHz (FSB 1067)</li> <li>▪ 100B: 333 MHz (FSB 1333)</li> </ul>
				133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
				266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.
		63:3		Reserved.
CDH	205	MSR_FSB_FREQ	Shared	<b>Scaleable Bus Speed(R0)</b> This field indicates the intended scaleable bus clock speed for processors based on Enhanced Intel Core microarchitecture:
		2:0		<ul style="list-style-type: none"> <li>▪ 101B: 100 MHz (FSB 400)</li> <li>▪ 001B: 133 MHz (FSB 533)</li> <li>▪ 011B: 167 MHz (FSB 667)</li> <li>▪ 010B: 200 MHz (FSB 800)</li> <li>▪ 000B: 267 MHz (FSB 1067)</li> <li>▪ 100B: 333 MHz (FSB 1333)</li> <li>▪ 110B: 400 MHz (FSB 1600)</li> </ul>
				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	<b>Maximum Performance Frequency Clock Count (RW)</b> See Table 2-2.

**Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
E8H	232	IA32_APERF	Unique	<b>Actual Performance Frequency Clock Count (RW)</b> See Table 2-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 2-2.
		11	Unique	<b>SMRR Capability Using MSR 0A0H and 0A1H (R)</b>
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		<b>RIPV</b> 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		<b>EIPV</b> 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		<b>MCIP</b> 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 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
198H	408	MSR_PERF_STATUS	Shared	Current performance status. See Section 14.1.1, “Software Interface For Initiating Performance State Transitions”.
		15:0		Current Performance State Value.
		30:16		Reserved.
		31		XE Operation (R/O). If set, XE operation is enabled. Default is cleared.
		39:32		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		45		Reserved.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		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 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	<b>Clock Modulation (R/W)</b> See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	<b>Thermal Interrupt Control (R/W)</b> See Table 2-2.
19CH	412	IA32_THERM_STATUS	Unique	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.
19DH	413	MSR_THERM2_CTL	Unique	Thermal Monitor 2 Control
		15:0		Reserved.
		16		<b>TM_SELECT (R/W)</b> Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:16		Reserved.
1A0H	416	IA32_MISC_ENABLE		<b>Enable Misc. Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.
		0		<b>Fast-Strings Enable</b> See Table 2-2.
		2:1		Reserved.
		3	Unique	<b>Automatic Thermal Control Circuit Enable (R/W)</b> See Table 2-2.
		6:4		Reserved.
		7	Shared	<b>Performance Monitoring Available (R)</b> See Table 2-2.
		8		Reserved.
		9		<b>Hardware Prefetcher Disable (R/W)</b> When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.

**Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		10	Shared	<b>FERR# Multiplexing Enable (R/W)</b> 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	<b>Branch Trace Storage Unavailable (RO)</b> See Table 2-2.
		12	Shared	<b>Processor Event Based Sampling Unavailable (RO)</b> See Table 2-2.
		13	Shared	<b>TM2 Enable (R/W)</b> 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	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> See Table 2-2.
		18	Shared	<b>ENABLE MONITOR FSM (R/W)</b> See Table 2-2.
		19	Shared	<b>Adjacent Cache Line Prefetch Disable (R/W)</b> When set to 1, the processor fetches the cache line that contains data currently required by the processor. When set to 0, the processor fetches cache lines that comprise a cache line pair (128 bytes). Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		20	Shared	<p><b>Enhanced Intel SpeedStep Technology Select Lock (R/WO)</b> When set, this bit causes the following bits to become read-only:</p> <ul style="list-style-type: none"> <li>Enhanced Intel SpeedStep Technology Select Lock (this bit),</li> <li>Enhanced Intel SpeedStep Technology Enable bit.</li> </ul> <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><b>Limit CPUID Maxval (R/W)</b> See Table 2-2.</p>
		23	Shared	<p><b>xTPR Message Disable (R/W)</b> See Table 2-2.</p>
		33:24		Reserved.
		34	Unique	<p><b>XD Bit Disable (R/W)</b> See Table 2-2.</p>
		36:35		Reserved.
		37	Unique	<p><b>DCU Prefetcher Disable (R/W)</b> 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><b>IDA Disable (R/W)</b> 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><b>Note:</b> 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>
		39	Unique	<p><b>IP Prefetcher Disable (R/W)</b> When set to 1, The IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature.</p> <p>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.</p>
		63:40		Reserved.

**Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1C9H	457	MSR_LASTBRANCH_TOS	Unique	<b>Last Branch Record Stack TOS (R/W)</b> Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_O_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	<b>Debug Control (R/W)</b> See Table 2-2
1DDH	477	MSR_LER_FROM_LIP	Unique	<b>Last Exception Record From Linear IP (R)</b> 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	<b>Last Exception Record To Linear IP (R)</b> This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Unique	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Unique	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Unique	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Unique	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Unique	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Unique	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Unique	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Unique	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Unique	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Unique	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Unique	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Unique	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Unique	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Unique	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Unique	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Unique	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Unique	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Unique	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Unique	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Unique	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Unique	See Table 2-2.



Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
26AH	618	IA32_MTRR_FIX4K_D0000	Unique	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Unique	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Unique	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Unique	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Unique	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Unique	See Table 2-2.
277H	631	IA32_PAT	Unique	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	<b>Default Memory Types (R/W)</b> See Table 2-2.
309H	777	IA32_FIXED_CTR0	Unique	<b>Fixed-Function Performance Counter Register 0 (R/W)</b> See Table 2-2.
309H	777	MSR_PERF_FIXED_CTR0	Unique	<b>Fixed-Function Performance Counter Register 0 (R/W)</b>
30AH	778	IA32_FIXED_CTR1	Unique	<b>Fixed-Function Performance Counter Register 1 (R/W)</b> See Table 2-2.
30AH	778	MSR_PERF_FIXED_CTR1	Unique	<b>Fixed-Function Performance Counter Register 1 (R/W)</b>
30BH	779	IA32_FIXED_CTR2	Unique	<b>Fixed-Function Performance Counter Register 2 (R/W)</b> See Table 2-2.
30BH	779	MSR_PERF_FIXED_CTR2	Unique	<b>Fixed-Function Performance Counter Register 2 (R/W)</b>
345H	837	IA32_PERF_CAPABILITIES	Unique	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
345H	837	MSR_PERF_CAPABILITIES	Unique	RO. This applies to processors that do not support architectural perfmon version 2.
		5:0		LBR Format. See Table 2-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 2-2.
		63:8		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Unique	<b>Fixed-Function-Counter Control Register (R/W)</b> See Table 2-2.
38DH	909	MSR_PERF_FIXED_CTR_CTRL	Unique	<b>Fixed-Function-Counter Control Register (R/W)</b>
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Unique	See Section 18.6.2.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
38FH	911	MSR_PERF_GLOBAL_CTRL	Unique	See Section 18.6.2.2, "Global Counter Control Facilities."

**Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Unique	See Section 18.6.2.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Unique	See Table 2-2. See Section 18.6.2.4, "Processor 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.
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	IA32_MC4_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC4_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC4_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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	IA32_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC3_STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
412H	1042	IA32_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the 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	IA32_MC3_MISC	Unique	Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.
414H	1044	IA32_MC5_CTL	Unique	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
415H	1045	IA32_MC5_STATUS	Unique	Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
416H	1046	IA32_MC5_ADDR	Unique	Machine Check Error Reporting Register - contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCi_STATUS register is set.
417H	1047	IA32_MC5_MISC	Unique	Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.
419H	1045	IA32_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	<b>Reporting Register of Basic VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Unique	<b>Capability Reporting Register of Pin-based VM-execution Controls (R/O)</b> See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Unique	<b>Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Unique	<b>Capability Reporting Register of VM-exit Controls (R/O)</b> See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Unique	<b>Capability Reporting Register of VM-entry Controls (R/O)</b> See Table 2-2. See Appendix A.5, "VM-Entry Controls."

**Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
485H	1157	IA32_VMX_MISC	Unique	<b>Reporting Register of Miscellaneous VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Unique	<b>Capability Reporting Register of CRO Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Unique	<b>Capability Reporting Register of CRO Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	<b>Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	<b>Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	<b>Capability Reporting Register of VMCS Field Enumeration (R/O)</b> See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Unique	<b>Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	<b>DS Save Area (R/W)</b> See Table 2-2. See Section 18.6.3.4, "Debug Store (DS) Mechanism."
107C H		MSR_EMON_L3_CTR_CTL0	Unique	<b>GBUSQ Event Control/Counter Register (R/W)</b> Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107C D H		MSR_EMON_L3_CTR_CTL1	Unique	<b>GBUSQ Event Control/Counter Register (R/W)</b> Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107C E H		MSR_EMON_L3_CTR_CTL2	Unique	<b>GSPNPQ Event Control/Counter Register (R/W)</b> Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107C F H		MSR_EMON_L3_CTR_CTL3	Unique	<b>GSPNPQ Event Control/Counter Register (R/W)</b> Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D 0 H		MSR_EMON_L3_CTR_CTL4	Unique	<b>FSB Event Control/Counter Register (R/W)</b> Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2

**Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
107D1 H		MSR_EMON_L3_CTR_CTL5	Unique	<b>FSB Event Control/Counter Register (R/W)</b> 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	<b>FSB Event Control/Counter Register (R/W)</b> 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	<b>FSB Event Control/Counter Register (R/W)</b> 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	<b>L3/FSB Common Control Register (R/W)</b> Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
C000_0080H		IA32_EFER	Unique	<b>Extended Feature Enables</b> See Table 2-2.
C000_0081H		IA32_STAR	Unique	<b>System Call Target Address (R/W)</b> See Table 2-2.
C000_0082H		IA32_LSTAR	Unique	<b>IA-32e Mode System Call Target Address (R/W)</b> See Table 2-2.
C000_0084H		IA32_FMASK	Unique	<b>System Call Flag Mask (R/W)</b> See Table 2-2.
C000_0100H		IA32_FS_BASE	Unique	<b>Map of BASE Address of FS (R/W)</b> See Table 2-2.
C000_0101H		IA32_GS_BASE	Unique	<b>Map of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Unique	<b>Swap Target of BASE Address of GS (R/W)</b> See Table 2-2.

## 2.3 MSRS IN THE 45 NM AND 32 NM INTEL® ATOM™ PROCESSOR FAMILY

Table 2-4 lists model-specific registers (MSRs) for 45 nm and 32 nm Intel Atom processors, architectural MSR addresses are also included in Table 2-4. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_1CH, 06\_26H, 06\_27H, 06\_35H and 06\_36H; see Table 2-1.

The column “Shared/Unique” applies to logical processors sharing the same core in processors based on the Intel Atom microarchitecture. “Unique” means each logical processor has a separate MSR, or a bit field in an MSR governs only a logical processor. “Shared” means the MSR or the bit field in an MSR address governs the operation of both logical processors in the same core.

**Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 2.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Shared	See Section 2.22, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	<b>Platform ID (R)</b> See Table 2-2.
17H	23	MSR_PLATFORM_ID	Shared	<b>Model Specific Platform ID (R)</b>
		7:0		Reserved.
		12:8		<b>Maximum Qualified Ratio (R)</b> The maximum allowed bus ratio.
		63:13		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	<b>Processor Hard Power-On Configuration (R/W)</b> Enables and disables processor features; <b>(R)</b> indicates current processor configuration.
		0		Reserved.
		1		<b>Data Error Checking Enable (R/W)</b> 1 = Enabled; 0 = Disabled Always 0.
		2		<b>Response Error Checking Enable (R/W)</b> 1 = Enabled; 0 = Disabled Always 0.
		3		<b>AERR# Drive Enable (R/W)</b> 1 = Enabled; 0 = Disabled Always 0.
		4		<b>BERR# Enable for initiator bus requests (R/W)</b> 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved.
		6		Reserved.
		7		<b>BINIT# Driver Enable (R/W)</b> 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved.
9		<b>Execute BIST (R/O)</b> 1 = Enabled; 0 = Disabled		

Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		10		<b>AERR# Observation Enabled (R/O)</b> 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved.
		12		<b>BINIT# Observation Enabled (R/O)</b> 1 = Enabled; 0 = Disabled Always 0.
		13		<b>Reserved.</b>
		14		<b>1 MByte Power on Reset Vector (R/O)</b> 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		<b>APIC Cluster ID (R/O)</b> Always 00B.
		19:18		Reserved.
		21:20		<b>Symmetric Arbitration ID (R/O)</b> Always 00B.
		26:22		<b>Integer Bus Frequency Ratio (R/O)</b>
3AH	58	IA32_FEATURE_CONTROL	Unique	<b>Control Features in Intel 64 Processor (R/W)</b> See Table 2-2.
40H	64	MSR_LASTBRANCH_0_FROM_IP	Unique	<b>Last Branch Record 0 From IP (R/W)</b> One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the <b>source instruction</b> . See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.5</li> </ul>
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	<b>Last Branch Record 1 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	<b>Last Branch Record 2 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	<b>Last Branch Record 3 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_LASTBRANCH_4_FROM_IP	Unique	<b>Last Branch Record 4 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_LASTBRANCH_5_FROM_IP	Unique	<b>Last Branch Record 5 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_LASTBRANCH_6_FROM_IP	Unique	<b>Last Branch Record 6 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_LASTBRANCH_7_FROM_IP	Unique	<b>Last Branch Record 7 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.

**Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	<b>Last Branch Record 0 To IP (R/W)</b> One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	<b>Last Branch Record 1 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	<b>Last Branch Record 2 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	<b>Last Branch Record 3 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Unique	<b>Last Branch Record 4 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Unique	<b>Last Branch Record 5 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Unique	<b>Last Branch Record 6 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
67H	103	MSR_LASTBRANCH_7_TO_IP	Unique	<b>Last Branch Record 7 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Shared	<b>BIOS Update Trigger Register (W)</b> See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	<b>BIOS Update Signature ID (RO)</b> See Table 2-2.
C1H	193	IA32_PMC0	Unique	<b>Performance counter register</b> See Table 2-2.
C2H	194	IA32_PMC1	Unique	<b>Performance Counter Register</b> See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	<b>Scaleable Bus Speed(RO)</b> This field indicates the intended scaleable bus clock speed for processors based on Intel Atom microarchitecture:
		2:0		<ul style="list-style-type: none"> <li>▪ 111B: 083 MHz (FSB 333)</li> <li>▪ 101B: 100 MHz (FSB 400)</li> <li>▪ 001B: 133 MHz (FSB 533)</li> <li>▪ 011B: 167 MHz (FSB 667)</li> </ul> 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	<b>Maximum Performance Frequency Clock Count (RW)</b> See Table 2-2.



Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
E8H	232	IA32_APERF	Unique	<b>Actual Performance Frequency Clock Count (RW)</b> See Table 2-2.
FEH	254	IA32_MTRRCAP	Shared	<b>Memory Type Range Register (R)</b> See Table 2-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	Control register 3. Used to configure the L2 Cache.
		0		<b>L2 Hardware Enabled (RO)</b> 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		<b>L2 Enabled. (R/W)</b> 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		<b>L2 Not Present (RO)</b> 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		<b>RIPV</b> 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		<b>EIPV</b> 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		<b>MCIP</b> 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_PERFVTSELO	Unique	See Table 2-2.

**Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
198H	408	MSR_PERF_STATUS	Shared	Performance Status
		15:0		Current Performance State Value.
		39:16		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		63:45		Reserved.
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	<b>Clock Modulation (R/W)</b> See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	<b>Thermal Interrupt Control (R/W)</b> See Table 2-2.
19CH	412	IA32_THERM_STATUS	Unique	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.
19DH	413	MSR_THERM2_CTL	Shared	Thermal Monitor 2 Control
		15:0		Reserved.
		16		<b>TM_SELECT (R/W)</b> Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:17		Reserved.
1A0H	416	IA32_MISC_ENABLE	Unique	<b>Enable Misc. Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.
		0		<b>Fast-Strings Enable</b> See Table 2-2.
		2:1		Reserved.
		3	Unique	<b>Automatic Thermal Control Circuit Enable (R/W)</b> See Table 2-2. Default value is 0.
		6:4		Reserved.
		7	Shared	<b>Performance Monitoring Available (R)</b> See Table 2-2.
		8		Reserved.
		9		Reserved.

Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		10	Shared	<b>FERR# Multiplexing Enable (R/W)</b> 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	<b>Branch Trace Storage Unavailable (RO)</b> See Table 2-2.
		12	Shared	<b>Processor Event Based Sampling Unavailable (RO)</b> See Table 2-2.
		13	Shared	<b>TM2 Enable (R/W)</b> 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	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> See Table 2-2.
		18	Shared	<b>ENABLE MONITOR FSM (R/W)</b> See Table 2-2.
		19		Reserved.
		20	Shared	<b>Enhanced Intel SpeedStep Technology Select Lock (R/WO)</b> When set, this bit causes the following bits to become read-only: <ul style="list-style-type: none"> <li>▪ Enhanced Intel SpeedStep Technology Select Lock (this bit),</li> <li>▪ Enhanced Intel SpeedStep Technology Enable bit.</li> </ul> The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved.
		22	Unique	<b>Limit CPUID Maxval (R/W)</b> See Table 2-2.
		23	Shared	<b>xTPR Message Disable (R/W)</b> See Table 2-2.
		33:24		Reserved.

**Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		34	Unique	<b>XD Bit Disable (R/W)</b> See Table 2-2.
		63:35		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Unique	<b>Last Branch Record Stack TOS (R/W)</b> 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	<b>Debug Control (R/W)</b> See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	<b>Last Exception Record From Linear IP (R)</b> 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	<b>Last Exception Record To Linear IP (R)</b> This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Shared	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Shared	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Shared	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Shared	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Shared	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Shared	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Shared	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Shared	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Shared	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Shared	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Shared	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Shared	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Shared	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Shared	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Shared	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Shared	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Shared	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Shared	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Shared	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Shared	See Table 2-2.

Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
269H	617	IA32_MTRR_FIX4K_C8000	Shared	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Shared	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Shared	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Shared	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Shared	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Shared	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Shared	See Table 2-2.
277H	631	IA32_PAT	Unique	See Table 2-2.
309H	777	IA32_FIXED_CTR0	Unique	<b>Fixed-Function Performance Counter Register 0 (R/W)</b> See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Unique	<b>Fixed-Function Performance Counter Register 1 (R/W)</b> See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Unique	<b>Fixed-Function Performance Counter Register 2 (R/W)</b> See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Shared	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Unique	<b>Fixed-Function-Counter Control Register (R/W)</b> See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Unique	See Table 2-2. See Section 18.6.2.4, "Processor 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."
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."

**Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
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	IA32_MC3_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	IA32_MC3_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the 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	IA32_MC4_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	IA32_MC4_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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	<b>Reporting Register of Basic VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Unique	<b>Capability Reporting Register of Pin-based VM-execution Controls (R/O)</b> See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Unique	<b>Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Unique	<b>Capability Reporting Register of VM-exit Controls (R/O)</b> See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Unique	<b>Capability Reporting Register of VM-entry Controls (R/O)</b> See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	<b>Reporting Register of Miscellaneous VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.6, "Miscellaneous Data."

Table 2-4. MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
486H	1158	IA32_VMX_CRO_FIXED0	Unique	<b>Capability Reporting Register of CRO Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Unique	<b>Capability Reporting Register of CRO Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	<b>Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	<b>Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	<b>Capability Reporting Register of VMCS Field Enumeration (R/O)</b> See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_ CTLS2	Unique	<b>Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	<b>DS Save Area (R/W)</b> See Table 2-2. See Section 18.6.3.4, "Debug Store (DS) Mechanism."
C000_0080H		IA32_EFER	Unique	<b>Extended Feature Enables</b> See Table 2-2.
C000_0081H		IA32_STAR	Unique	<b>System Call Target Address (R/W)</b> See Table 2-2.
C000_0082H		IA32_LSTAR	Unique	<b>IA-32e Mode System Call Target Address (R/W)</b> See Table 2-2.
C000_0084H		IA32_FMASK	Unique	<b>System Call Flag Mask (R/W)</b> See Table 2-2.
C000_0100H		IA32_FS_BASE	Unique	<b>Map of BASE Address of FS (R/W)</b> See Table 2-2.
C000_0101H		IA32_GS_BASE	Unique	<b>Map of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Unique	<b>Swap Target of BASE Address of GS (R/W)</b> See Table 2-2.

Table 2-5 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor with the CPUID signature with DisplayFamily\_DisplayModel of 06\_27H.

**Table 2-5. MSRs Supported by Intel® Atom™ Processors with CPUID Signature 06\_27H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3F8H	1016	MSR_PKG_C2_RESIDENCY	Package	<b>Package C2 Residency</b> 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	<b>Package C4 Residency</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States
		63:0	Package	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	<b>Package C6 Residency</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States
		63:0	Package	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.

## 2.4 MSRS IN INTEL PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 2-6 lists model-specific registers (MSRs) common to Intel processors based on the Silvermont microarchitecture. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_37H, 06\_4AH, 06\_4DH, 06\_5AH, and 06\_5DH; see Table 2-1. The MSRs listed in Table 2-6 are also common to processors based on the Airmont microarchitecture and newer microarchitectures for next generation Intel Atom processors.

Table 2-7 lists MSRs common to processors based on the Silvermont and Airmont microarchitectures, but not newer microarchitectures.

Table 2-8, Table 2-9, and Table 2-10 lists MSRs that are model-specific across processors based on the Silvermont microarchitecture.

In the Silvermont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a pair of processor cores in the physical package. “Package” means all processor cores in the physical package share the same MSR or bit interface.

**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 2.22, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Module	See Section 2.22, “MSRs in Pentium Processors.”



**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
6H	6	IA32_MONITOR_FILTER_SIZE	Core	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2
10H	16	IA32_TIME_STAMP_COUNTER	Core	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
1BH	27	IA32_APIC_BASE	Core	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Module	<b>Processor Hard Power-On Configuration (R/W)</b> Writes ignored.
		63:0		Reserved (R/O)
34H	52	MSR_SMI_COUNT	Core	<b>SMI Counter (R/O)</b>
		31:0		<b>SMI Count (R/O)</b> Running count of SMI events since last RESET.
		63:32		Reserved.
79H	121	IA32_BIOS_UPDT_TRIG	Core	<b>BIOS Update Trigger Register (W)</b> See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Core	<b>BIOS Update Signature ID (RO)</b> See Table 2-2.
C1H	193	IA32_PMC0	Core	<b>Performance counter register</b> See Table 2-2.
C2H	194	IA32_PMC1	Core	<b>Performance Counter Register</b> See Table 2-2.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Module	<b>Power Management IO Redirection in C-state (R/W)</b> See <a href="http://biosbits.org">http://biosbits.org</a> .
		15:0		<b>LVL_2 Base Address (R/W)</b> 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		<b>C-state Range (R/W)</b> Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 100b - C4 is the max C-State to include 110b - C6 is the max C-State to include 111b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Core	<b>Maximum Performance Frequency Clock Count (RW)</b> See Table 2-2.
E8H	232	IA32_APERF	Core	<b>Actual Performance Frequency Clock Count (RW)</b> See Table 2-2.

**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
FEH	254	IA32_MTRRCAP	Core	<b>Memory Type Range Register (R)</b> See Table 2-2.
13CH	52	MSR_FEATURE_CONFIG	Core	<b>AES Configuration (RW-L)</b> Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		<b>AES Configuration (RW-L)</b> Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 1 1b: AES instructions are not available until next RESET. otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
174H	372	IA32_SYSENTER_CS	Core	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Core	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Core	See Table 2-2.
179H	377	IA32_MCG_CAP	Core	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Core	Global Machine Check Status
		0		<b>RIPV</b> 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		<b>EIPV</b> 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		<b>MCIP</b> 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_PERFVTSELO	Core	See Table 2-2.
		7:0		<b>Event Select</b>
		15:8		<b>UMask</b>
		16		<b>USR</b>
		17		<b>OS</b>
		18		<b>Edge</b>
		19		<b>PC</b>
		20		<b>INT</b>

**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		21		<b>Reserved</b>
		22		<b>EN</b>
		23		<b>INV</b>
		31:24		<b>CMASK</b>
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Core	See Table 2-2.
198H	408	IA32_PERF_STATUS	Module	See Table 2-2.
199H	409	IA32_PERF_CTL	Core	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Core	<b>Clock Modulation (R/W)</b> See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Core	<b>Thermal Interrupt Control (R/W)</b> See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved.
		23:16		<b>Temperature Target (R)</b> The default thermal throttling or PROCHOT# activation temperature in degree C, The effective temperature for thermal throttling or PROCHOT# activation is "Temperature Target" + "Target Offset"
		29:24		<b>Target Offset (R/W)</b> Specifies an offset in degrees C to adjust the throttling and PROCHOT# activation temperature from the default target specified in TEMPERATURE_TARGET (bits 23:16).
		63:30		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Module	<b>Offcore Response Event Select Register (R/W)</b>
1A7H	423	MSR_OFFCORE_RSP_1	Module	<b>Offcore Response Event Select Register (R/W)</b>
1B0H	432	IA32_ENERGY_PERF_BIAS	Core	See Table 2-2.
1D9H	473	IA32_DEBUGCTL	Core	<b>Debug Control (R/W)</b> See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Core	<b>Last Exception Record From Linear IP (R)</b> Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.

**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
1DEH	478	MSR_LER_TO_LIP	Core	<b>Last Exception Record To Linear IP (R)</b> This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 2-2.
277H	631	IA32_PAT	Core	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	<b>Default Memory Types (R/W)</b> See Table 2-2.

**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
309H	777	IA32_FIXED_CTR0	Core	<b>Fixed-Function Performance Counter Register 0 (R/W)</b> See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Core	<b>Fixed-Function Performance Counter Register 1 (R/W)</b> See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Core	<b>Fixed-Function Performance Counter Register 2 (R/W)</b> See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Core	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Core	<b>Fixed-Function-Counter Control Register (R/W)</b> See Table 2-2.
38FH	911	IA32_PERF_GLOBAL_CTRL	Core	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Counts at the TSC Frequency.
400H	1024	IA32_MCO_CTL	Module	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Module	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	Module	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	Module	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Module	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Module	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Module	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Module	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the 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	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."

**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the 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	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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	Core	<b>Reporting Register of Basic VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Core	<b>Capability Reporting Register of Pin-based VM-execution Controls (R/O)</b> See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Core	<b>Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Core	<b>Capability Reporting Register of VM-exit Controls (R/O)</b> See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Core	<b>Capability Reporting Register of VM-entry Controls (R/O)</b> See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Core	<b>Reporting Register of Miscellaneous VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.6, "Miscellaneous Data."

**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
486H	1158	IA32_VMX_CR0_FIXED0	Core	<b>Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Core	<b>Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Core	<b>Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Core	<b>Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Core	<b>Capability Reporting Register of VMCS Field Enumeration (R/O)</b> See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLD2	Core	<b>Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	<b>Capability Reporting Register of EPT and VPID (R/O)</b> See Table 2-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLD	Core	<b>Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)</b> See Table 2-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLD	Core	<b>Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)</b> See Table 2-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLD	Core	<b>Capability Reporting Register of VM-exit Flex Controls (R/O)</b> See Table 2-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLD	Core	<b>Capability Reporting Register of VM-entry Flex Controls (R/O)</b> See Table 2-2
491H	1169	IA32_VMX_FMFUNC	Core	<b>Capability Reporting Register of VM-function Controls (R/O)</b> See Table 2-2
4C1H	1217	IA32_A_PMC0	Core	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Core	See Table 2-2.
600H	1536	IA32_DS_AREA	Core	<b>DS Save Area (R/W)</b> See Table 2-2. See Section 18.6.3.4, "Debug Store (DS) Mechanism."
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

**Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:0		CORE C1 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C1 states. Counts at the TSC frequency.
6E0H	1760	IA32_TSC_DEADLINE	Core	<b>TSC Target of Local APIC's TSC Deadline Mode (R/W)</b> See Table 2-2
C000_0080H		IA32_EFER	Core	<b>Extended Feature Enables</b> See Table 2-2.
C000_0081H		IA32_STAR	Core	<b>System Call Target Address (R/W)</b> See Table 2-2.
C000_0082H		IA32_LSTAR	Core	<b>IA-32e Mode System Call Target Address (R/W)</b> See Table 2-2.
C000_0084H		IA32_FMASK	Core	<b>System Call Flag Mask (R/W)</b> See Table 2-2.
C000_0100H		IA32_FS_BASE	Core	<b>Map of BASE Address of FS (R/W)</b> See Table 2-2.
C000_0101H		IA32_GS_BASE	Core	<b>Map of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Core	<b>Swap Target of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0103H		IA32_TSC_AUX	Core	<b>AUXILIARY TSC Signature. (R/W)</b> See Table 2-2

Table 2-7 lists model-specific registers (MSRs) that are common to Intel® Atom™ processors based on the Silvermont and Airmont microarchitectures but not newer microarchitectures.

**Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	<b>Model Specific Platform ID (R)</b>
		7:0		Reserved.
		13:8		<b>Maximum Qualified Ratio (R)</b> The maximum allowed bus ratio.
		49:13		Reserved.
		52:50		See Table 2-2
		63:33		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Core	<b>Control Features in Intel 64Processor (R/W)</b> See Table 2-2.
		0		<b>Lock (R/WL)</b>
		1		<b>Reserved</b>



Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		<b>Enable VMX outside SMX operation (R/WL)</b>
40H	64	MSR_LASTBRANCH_0_FROM_IP	Core	<b>Last Branch Record 0 From IP (R/W)</b> One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the <b>source instruction</b> . See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.5 and record format in Section 17.4.8.1</li> </ul>
41H	65	MSR_LASTBRANCH_1_FROM_IP	Core	<b>Last Branch Record 1 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Core	<b>Last Branch Record 2 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Core	<b>Last Branch Record 3 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_LASTBRANCH_4_FROM_IP	Core	<b>Last Branch Record 4 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_LASTBRANCH_5_FROM_IP	Core	<b>Last Branch Record 5 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_LASTBRANCH_6_FROM_IP	Core	<b>Last Branch Record 6 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_LASTBRANCH_7_FROM_IP	Core	<b>Last Branch Record 7 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Core	<b>Last Branch Record 0 To IP (R/W)</b> One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Core	<b>Last Branch Record 1 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Core	<b>Last Branch Record 2 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Core	<b>Last Branch Record 3 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Core	<b>Last Branch Record 4 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Core	<b>Last Branch Record 5 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Core	<b>Last Branch Record 6 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
67H	103	MSR_LASTBRANCH_7_TO_IP	Core	<b>Last Branch Record 7 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.

**Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Module	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See <a href="http://biosbits.org">http://biosbits.org</a> .
		2:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power), for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 100b: C4 110b: C6 111b: C7 (Silvermont only).
		9:3		Reserved.
		10		<b>I/O MWAIT Redirection Enable (R/W)</b> 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		<b>CFG Lock (R/WO)</b> When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
11EH	281	MSR_BBL_CR_CTL3	Module	Control register 3. Used to configure the L2 Cache.
		0		<b>L2 Hardware Enabled (RO)</b> 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		<b>L2 Enabled. (R/W)</b> 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		<b>L2 Not Present (RO)</b> 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
1A0H	416	IA32_MISC_ENABLE		<b>Enable Misc. Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0	Core	<b>Fast-Strings Enable</b> See Table 2-2.
		2:1		Reserved.
		3	Module	<b>Automatic Thermal Control Circuit Enable (R/W)</b> See Table 2-2. Default value is 0.
		6:4		Reserved.
		7	Core	<b>Performance Monitoring Available (R)</b> See Table 2-2.
		10:8		Reserved.
		11	Core	<b>Branch Trace Storage Unavailable (RO)</b> See Table 2-2.
		12	Core	<b>Processor Event Based Sampling Unavailable (RO)</b> See Table 2-2.
		15:13		Reserved.
		16	Module	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> See Table 2-2.
		18	Core	<b>ENABLE MONITOR FSM (R/W)</b> See Table 2-2.
		21:19		Reserved.
		22	Core	<b>Limit CPUID Maxval (R/W)</b> See Table 2-2.
		23	Module	<b>xTPR Message Disable (R/W)</b> See Table 2-2.
		33:24		Reserved.
		34	Core	<b>XD Bit Disable (R/W)</b> See Table 2-2.
		37:35		Reserved.
		38	Module	<b>Turbo Mode Disable (R/W)</b> 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. <b>Note:</b> 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.
1C8H	456	MSR_LBR_SELECT	Core	<b>Last Branch Record Filtering Select Register (R/W)</b> See Section 17.9.2, "Filtering of Last Branch Records."

**Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
		63:9		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Core	<p><b>Last Branch Record Stack TOS (R/W)</b></p> <p>Contains an index (bits 0-2) that points to the MSR containing the most recent branch record.</p> <p>See MSR_LASTBRANCH_0_FROM_IP.</p>
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Core	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Core	See Table 2-2. See Section 18.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS for precise event on IA32_PMC0. (R/W)
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Counts at the TSC Frequency.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	<p><b>Module C6 Residency Counter (R/O)</b></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>
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.
		15:8	Package	<p><b>Maximum Non-Turbo Ratio (R/O)</b></p> <p>The is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * Scalable Bus Frequency.</p>
		63:16		Reserved.

## 2.4.1 MSRs with Model-Specific Behavior in the Silvermont Microarchitecture

Table 2-8 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor E3000 Series (CPUID signature with DisplayFamily\_DisplayModel of 06\_37H) and Intel Atom processors (CPUID signatures with DisplayFamily\_DisplayModel of 06\_4AH, 06\_5AH, 06\_5DH).

**Table 2-8. Specific MSRs Supported by Intel® Atom™ Processors with CPUID Signatures 06\_37H, 06\_4AH, 06\_5AH, 06\_5DH**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
606H	1542	MSR_RAPL_POWER_UNIT	Package	<b>Unit Multipliers used in RAPL Interfaces (R/O)</b> See Section 14.9.1, "RAPL Interfaces."
		3:0		Power Units. Power related information (in milliWatts) is based on the multiplier, $2^{\text{PU}}$ ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.
		7:4		Reserved
		12:8		Energy Status Units. Energy related information (in microJoules) is based on the multiplier, $2^{\text{ESU}}$ ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.
		15:13		Reserved
		19:16		Time Unit. The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	<b>PKG RAPL Power Limit Control (R/W)</b>
		14:0		Package Power Limit #1. (R/W) See Section 14.9.3, "Package RAPL Domain." and MSR_RAPL_POWER_UNIT in Table 2-8.
		15		Enable Power Limit #1. (R/W) See Section 14.9.3, "Package RAPL Domain."
		16		Package Clamping Limitation #1. (R/W) See Section 14.9.3, "Package RAPL Domain."
		23:17		Time Window for Power Limit #1. (R/W) in unit of second. If 0 is specified in bits [23:17], defaults to 1 second window.
		63:24		Reserved
611H	1553	MSR_PKG_ENERGY_STATUS	Package	<b>PKG Energy Status (R/O)</b> See Section 14.9.3, "Package RAPL Domain." and MSR_RAPL_POWER_UNIT in Table 2-8

**Table 2-8. Specific MSRs Supported by Intel® Atom™ Processors with CPUID Signatures 06\_37H, 06\_4AH, 06\_5AH, 06\_5DH**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
639H	1593	MSR_PP0_ENERGY_STATUS	Package	<b>PP0 Energy Status (R/O)</b> See Section 14.9.4, “PP0/PP1 RAPL Domains.” and MSR_RAPL_POWER_UNIT in Table 2-8
CDH	205	MSR_FSB_FREQ	Module	<b>Scaleable Bus Speed(R/O)</b> This field indicates the intended scaleable bus clock speed for processors based on Silvermont microarchitecture.
		2:0		<ul style="list-style-type: none"> <li>▪ 100B: 080.0 MHz</li> <li>▪ 000B: 083.3 MHz</li> <li>▪ 001B: 100.0 MHz</li> <li>▪ 010B: 133.3 MHz</li> <li>▪ 011B: 116.7 MHz</li> </ul>
		63:3		Reserved.

Table 2-9 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor E3000 Series (CPUID signature with DisplayFamily\_DisplayModel of 06\_37H).

**Table 2-9. Specific MSRs Supported by Intel® Atom™ Processor E3000 Series with CPUID Signature 06\_37H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
668H	1640	MSR_CC6_DEMOTION_POLICY_CONFIG	Package	<b>Core C6 demotion policy config MSR</b>
		63:0		Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.
669H	1641	MSR_MC6_DEMOTION_POLICY_CONFIG	Package	<b>Module C6 demotion policy config MSR</b>
		63:0		Controls module (i.e. two cores sharing the second-level cache) C6 demotion policy. Writing a value of 0 disables module level HW demotion policy.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	<b>Module C6 Residency Counter (R/O)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.

Table 2-10 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor C2000 Series (CPUID signature with DisplayFamily\_DisplayModel of 06\_4DH).

Table 2-10. Specific MSRs Supported by Intel® Atom™ Processor C2000 Series with CPUID Signature 06\_4DH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A4H	420	MSR_MISC_FEATURE_CONTROL		<b>Miscellaneous Feature Control (R/W)</b>
		0	Core	<b>L2 Hardware Prefetcher Disable (R/W)</b> If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1		<b>Reserved</b>
		2	Core	<b>DCU Hardware Prefetcher Disable (R/W)</b> If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		63:3		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode (RW)</b>
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		39:32	Package	<b>Maximum Ratio Limit for 5C</b> Maximum turbo ratio limit of 5 core active.
		47:40	Package	<b>Maximum Ratio Limit for 6C</b> Maximum turbo ratio limit of 6 core active.
		55:48	Package	<b>Maximum Ratio Limit for 7C</b> Maximum turbo ratio limit of 7 core active.
		63:56	Package	<b>Maximum Ratio Limit for 8C</b> Maximum turbo ratio limit of 8 core active.
606H	1542	MSR_RAPL_POWER_UNIT	Package	<b>Unit Multipliers used in RAPL Interfaces (R/O)</b> See Section 14.9.1, "RAPL Interfaces."
		3:0		Power Units. Power related information (in milliwatts) is based on the multiplier, $2^{\text{PU}}$ ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliwatts increment.
		7:4		Reserved
		12:8		Energy Status Units. Energy related information (in microjoules) is based on the multiplier, $2^{\text{ESU}}$ ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microjoules increment.
		15:13		Reserved

**Table 2-10. Specific MSRs Supported by Intel® Atom™ Processor C2000 Series (Contd.)with CPUID Signature**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		19:16		Time Unit. The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	<b>PKG RAPL Power Limit Control (R/W)</b> See Section 14.9.3, "Package RAPL Domain."
66EH	1646	MSR_PKG_POWER_INFO	Package	<b>PKG RAPL Parameter (R/O)</b>
		14:0		Thermal Spec Power. (R/O) The unsigned integer value is the equivalent of thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT
		63:15		Reserved

### 2.4.2 MSRs In Intel Atom Processors Based on Airmont Microarchitecture

Intel Atom processor X7-Z8000 and X5-Z8000 series are based on the Airmont microarchitecture. These processors support MSRs listed in Table 2-6, Table 2-7, Table 2-8, and Table 2-11. These processors have a CPUID signature with DisplayFamily\_DisplayModel including 06\_4CH; see Table 2-1.

**Table 2-11. MSRs in Intel Atom Processors Based on the Airmont Microarchitecture**

Address		Register Name	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Module	<b>Scaleable Bus Speed(R0)</b> This field indicates the intended scaleable bus clock speed for processors based on Airmont microarchitecture:
		3:0		<ul style="list-style-type: none"> <li>▪ 0000B: 083.3 MHz</li> <li>▪ 0001B: 100.0 MHz</li> <li>▪ 0010B: 133.3 MHz</li> <li>▪ 0011B: 116.7 MHz</li> <li>▪ 0100B: 080.0 MHz</li> <li>▪ 0101B: 093.3 MHz</li> <li>▪ 0110B: 090.0 MHz</li> <li>▪ 0111B: 088.9 MHz</li> <li>▪ 1000B: 087.5 MHz</li> </ul>
		63:5		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Module	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See <a href="http://biosbits.org">http://biosbits.org</a> .



Table 2-11. MSRs in Intel Atom Processors Based on the Airmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power), for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: No limit 001b: C1 010b: C2 110b: C6 111b: C7
		9:3		Reserved.
		10		<b>I/O MWAIT Redirection Enable (R/W)</b> 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		<b>CFG Lock (R/WO)</b> When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Module	<b>Power Management IO Redirection in C-state (R/W)</b> See <a href="http://biosbits.org">http://biosbits.org</a> .
		15:0		<b>LVL_2 Base Address (R/W)</b> 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		<b>C-state Range (R/W)</b> Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - Deep Power Down Technology is the max C-State 010b - C7 is the max C-State to include
		63:19		Reserved.
638H	1592	MSR_PPO_POWER_LIMIT	Package	<b>PPO RAPL Power Limit Control (R/W)</b>
		14:0		PPO Power Limit #1. (R/W) See Section 14.9.4, "PPO/PP1 RAPL Domains," and MSR_RAPL_POWER_UNIT in Table 2-8.
		15		Enable Power Limit #1. (R/W) See Section 14.9.4, "PPO/PP1 RAPL Domains."
		16		Reserved

**Table 2-11. MSRs in Intel Atom Processors Based on the Airmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		23:17		Time Window for Power Limit #1. (R/W) Specifies the time duration over which the average power must remain below PPO_POWER_LIMIT #1(14:0). Supported Encodings: 0x0: 1 second time duration. 0x1: 5 second time duration (Default). 0x2: 10 second time duration. 0x3: 15 second time duration. 0x4: 20 second time duration. 0x5: 25 second time duration. 0x6: 30 second time duration. 0x7: 35 second time duration. 0x8: 40 second time duration. 0x9: 45 second time duration. 0xA: 50 second time duration. 0xB-0x7F - reserved.
		63:24		Reserved

## 2.5 MSRS IN INTEL ATOM PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Intel Atom processors based on the Goldmont microarchitecture support MSRs listed in Table 2-6 and Table 2-12. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_5CH; see Table 2-1.

In the Goldmont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a pair of processor cores in the physical package. “Package” means all processor cores in the physical package share the same MSR or bit interface.

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture**

Address		Register Name	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	<b>Model Specific Platform ID (R)</b>
		49:0		Reserved.
		52:50		See Table 2-2.
		63:33		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Core	<b>Control Features in Intel 64Processor (R/W)</b> See Table 2-2.
		0		<b>Lock (R/WL)</b>
		1		<b>Enable VMX inside SMX operation (R/WL)</b>
		2		<b>Enable VMX outside SMX operation (R/WL)</b>
		14:8		<b>SENTER local functions enables (R/WL)</b>

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		15		<b>SENTER global functions enable (R/WL)</b>
		18		<b>SGX global functions enable (R/WL)</b>
		63:19		Reserved.
3BH	59	IA32_TSC_ADJUST	Core	<b>Per-Core TSC ADJUST (R/W)</b> See Table 2-2.
C3H	195	IA32_PMC2	Core	<b>Performance Counter Register</b> See Table 2-2.
C4H	196	IA32_PMC3	Core	<b>Performance Counter Register</b> See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> The is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> 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	<b>Programmable TDP Limit for Turbo Mode (R/O)</b> 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.
		30	Package	<b>Programmable TJ OFFSET (R/O)</b> When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify an temperature offset.
		39:31		Reserved.
		47:40	Package	<b>Maximum Efficiency Ratio (R/O)</b> 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	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See <a href="http://biosbits.org">http://biosbits.org</a> .

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		3:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power), for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: No limit 0001b: C1 0010b: C3 0011b: C6 0100b: C7 0101b: C7S 0110b: C8 0111b: C9 1000b: C10
		9:3		Reserved.
		10		<b>I/O MWAIT Redirection Enable (R/W)</b> 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		<b>CFG Lock (R/WO)</b> When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
17DH	381	MSR_SMM_MCA_CAP	Core	<b>Enhanced SMM Capabilities (SMM-RO)</b> Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		<b>Reserved</b>
		58		<b>SMM_Code_Access_Chk (SMM-RO)</b> If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.
		59		<b>Long_Flow_Indication (SMM-RO)</b> If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.
		63:60		Reserved
188H	392	IA32_PERFEVTSEL2	Core	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Core	See Table 2-2.
1A0H	416	IA32_MISC_ENABLE		<b>Enable Misc. Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.
		0	Core	<b>Fast-Strings Enable</b> See Table 2-2.
		2:1		Reserved.
		3	Package	<b>Automatic Thermal Control Circuit Enable (R/W)</b> See Table 2-2. Default value is 1.

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		6:4		Reserved.
		7	Core	<b>Performance Monitoring Available (R)</b> See Table 2-2.
		10:8		Reserved.
		11	Core	<b>Branch Trace Storage Unavailable (RO)</b> See Table 2-2.
		12	Core	<b>Processor Event Based Sampling Unavailable (RO)</b> See Table 2-2.
		15:13		Reserved.
		16	Package	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> See Table 2-2.
		18	Core	<b>ENABLE MONITOR FSM (R/W)</b> See Table 2-2.
		21:19		Reserved.
		22	Core	<b>Limit CPUID Maxval (R/W)</b> See Table 2-2.
		23	Package	<b>xTPR Message Disable (R/W)</b> See Table 2-2.
		33:24		Reserved.
		34	Core	<b>XD Bit Disable (R/W)</b> See Table 2-2.
		37:35		Reserved.
		38	Package	<b>Turbo Mode Disable (R/W)</b> 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. <b>Note:</b> 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.		
1A4H	420	MSR_MISC_FEATURE_CONTROL		<b>Miscellaneous Feature Control (R/W)</b>
		0	Core	<b>L2 Hardware Prefetcher Disable (R/W)</b> If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1		<b>Reserved</b>
		2	Core	<b>DCU Hardware Prefetcher Disable (R/W)</b> If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:3		Reserved.
1AAH	426	MSR_MISC_PWR_MGMT	Package	Miscellaneous Power Management Control; various model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		0		<b>EIST Hardware Coordination Disable (R/W)</b> When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores; When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.
		21:1		Reserved.
		22		<b>Thermal Interrupt Coordination Enable (R/W)</b> If set, then thermal interrupt on one core is routed to all cores.
		63:23		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode by Core Groups (RW)</b> Specifies Maximum Ratio Limit for each Core Group. Max ratio for groups with more cores must decrease monotonically. <b>For groups with less than 4 cores, the max ratio must be 32 or less. For groups with 4-5 cores, the max ratio must be 22 or less. For groups with more than 5 cores, the max ratio must be 16 or less.</b>
		7:0	Package	<b>Maximum Ratio Limit for Active cores in Group 0</b> Maximum turbo ratio limit when number of active cores is less or equal to Group 0 threshold.
		15:8	Package	<b>Maximum Ratio Limit for Active cores in Group 1</b> Maximum turbo ratio limit when number of active cores is less or equal to Group 1 threshold and greater than Group 0 threshold.
		23:16	Package	<b>Maximum Ratio Limit for Active cores in Group 2</b> Maximum turbo ratio limit when number of active cores is less or equal to Group 2 threshold and greater than Group 1 threshold.
		31:24	Package	<b>Maximum Ratio Limit for Active cores in Group 3</b> Maximum turbo ratio limit when number of active cores is less or equal to Group 3 threshold and greater than Group 2 threshold.
		39:32	Package	<b>Maximum Ratio Limit for Active cores in Group 4</b> Maximum turbo ratio limit when number of active cores is less or equal to Group 4 threshold and greater than Group 3 threshold.
		47:40	Package	<b>Maximum Ratio Limit for Active cores in Group 5</b> Maximum turbo ratio limit when number of active cores is less or equal to Group 5 threshold and greater than Group 4 threshold.
		55:48	Package	<b>Maximum Ratio Limit for Active cores in Group 6</b> Maximum turbo ratio limit when number of active cores is less or equal to Group 6 threshold and greater than Group 5 threshold.
		63:56	Package	<b>Maximum Ratio Limit for Active cores in Group 7</b> Maximum turbo ratio limit when number of active cores is less or equal to Group 7 threshold and greater than Group 6 threshold.

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
1AEH	430	MSR_TURBO_GROUP_CORE CNT	Package	<b>Group Size of Active Cores for Turbo Mode Operation (RW)</b> Writes of 0 threshold is ignored
		7:0	Package	<b>Group 0 Core Count Threshold</b> Maximum number of active cores to operate under Group 0 Max Turbo Ratio limit.
		15:8	Package	<b>Group 1 Core Count Threshold</b> Maximum number of active cores to operate under Group 1 Max Turbo Ratio limit. Must be greater than Group 0 Core Count.
		23:16	Package	<b>Group 2 Core Count Threshold</b> Maximum number of active cores to operate under Group 2 Max Turbo Ratio limit. Must be greater than Group 1 Core Count.
		31:24	Package	<b>Group 3 Core Count Threshold</b> Maximum number of active cores to operate under Group 3 Max Turbo Ratio limit. Must be greater than Group 2 Core Count.
		39:32	Package	<b>Group 4 Core Count Threshold</b> Maximum number of active cores to operate under Group 4 Max Turbo Ratio limit. Must be greater than Group 3 Core Count.
		47:40	Package	<b>Group 5 Core Count Threshold</b> Maximum number of active cores to operate under Group 5 Max Turbo Ratio limit. Must be greater than Group 4 Core Count.
		55:48	Package	<b>Group 6 Core Count Threshold</b> Maximum number of active cores to operate under Group 6 Max Turbo Ratio limit. Must be greater than Group 5 Core Count.
		63:56	Package	<b>Group 7 Core Count Threshold</b> Maximum number of active cores to operate under Group 7 Max Turbo Ratio limit. Must be greater than Group 6 Core Count and not less than the total number of processor cores in the package. E.g. specify 255.
1C8H	456	MSR_LBR_SELECT	Core	<b>Last Branch Record Filtering Select Register (R/W)</b> See Section 17.9.2, "Filtering of Last Branch Records."
		0		<b>CPL_EQ_0</b>
		1		<b>CPL_NEQ_0</b>
		2		<b>JCC</b>
		3		<b>NEAR_REL_CALL</b>
		4		<b>NEAR_IND_CALL</b>
		5		<b>NEAR_RET</b>
		6		<b>NEAR_IND_JMP</b>
		7		<b>NEAR_REL_JMP</b>
		8		<b>FAR_BRANCH</b>
		9		<b>EN_CALL_STACK</b>
63:10		Reserved.		

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
1C9H	457	MSR_LASTBRANCH_TOS	Core	<b>Last Branch Record Stack TOS (R/W)</b> Contains an index (bits 0-4) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_O_FROM_IP.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See <a href="http://biosbits.org">http://biosbits.org</a> .
		0		Reserved.
		1	Package	<b>C1E Enable (R/W)</b> When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		63:2		Reserved.
210H	528	IA32_MTRR_PHYSBASE8	Core	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Core	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Core	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Core	See Table 2-2.
280H	640	IA32_MC0_CTL2	Module	See Table 2-2.
281H	641	IA32_MC1_CTL2	Module	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Module	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	See Table 2-2.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
300H	768	MSR_SGXOWNEREPOCH0	Package	<b>Lower 64 Bit CR_SGXOWNEREPOCH.</b> Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.
301H	769	MSR_SGXOWNEREPOCH1	Package	<b>Upper 64 Bit CR_SGXOWNEREPOCH.</b> Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 2-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0		<b>Ovf_PMC0</b>
		1		<b>Ovf_PMC1</b>
		2		<b>Ovf_PMC2</b>
		3		<b>Ovf_PMC3</b>
		31:4		Reserved.
		32		<b>Ovf_FixedCtr0</b>
		33		<b>Ovf_FixedCtr1</b>



Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		34		<b>Ovf_FixedCtr2</b>
		54:35		Reserved.
		55		<b>Trace_ToPA_PMI.</b>
		57:56		Reserved.
		58		<b>LBR_Frz.</b>
		59		<b>CTR_Frz.</b>
		60		<b>ASCI.</b>
		61		<b>Ovf_Uncore</b>
		62		<b>Ovf_BufDSSAVE</b>
		63		<b>CondChgd</b>
390H	912	IA32_PERF_GLOBAL_STAT US_RESET	Core	See Table 2-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0		<b>Set 1 to clear Ovf_PMC0</b>
		1		<b>Set 1 to clear Ovf_PMC1</b>
		2		<b>Set 1 to clear Ovf_PMC2</b>
		3		<b>Set 1 to clear Ovf_PMC3</b>
		31:4		Reserved.
		32		<b>Set 1 to clear Ovf_FixedCtr0</b>
		33		<b>Set 1 to clear Ovf_FixedCtr1</b>
		34		<b>Set 1 to clear Ovf_FixedCtr2</b>
		54:35		Reserved.
		55		<b>Set 1 to clear Trace_ToPA_PMI.</b>
		57:56		Reserved.
		58		<b>Set 1 to clear LBR_Frz.</b>
		59		<b>Set 1 to clear CTR_Frz.</b>
		60		<b>Set 1 to clear ASCI.</b>
		61		<b>Set 1 to clear Ovf_Uncore</b>
		62		<b>Set 1 to clear Ovf_BufDSSAVE</b>
63		<b>Set 1 to clear CondChgd</b>		
391H	913	IA32_PERF_GLOBAL_STAT US_SET	Core	See Table 2-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0		<b>Set 1 to cause Ovf_PMC0 = 1</b>
		1		<b>Set 1 to cause Ovf_PMC1 = 1</b>
		2		<b>Set 1 to cause Ovf_PMC2 = 1</b>
		3		<b>Set 1 to cause Ovf_PMC3 = 1</b>
		31:4		Reserved.
		32		<b>Set 1 to cause Ovf_FixedCtr0 = 1</b>
		33		<b>Set 1 to cause Ovf_FixedCtr1 = 1</b>

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		34		Set 1 to cause Ovf_FixedCtr2 = 1
		54:35		Reserved.
		55		Set 1 to cause Trace_ToPA_PMI = 1
		57:56		Reserved.
		58		Set 1 to cause LBR_Frz = 1
		59		Set 1 to cause CTR_Frz = 1
		60		Set 1 to cause ASCI = 1
		61		Set 1 to cause Ovf_Uncore
		62		Set 1 to cause Ovf_BufDSSAVE
		63		Reserved.
392H	914	IA32_PERF_GLOBAL_INUSE		See Table 2-2.
3F1H	1009	MSR_PEBS_ENABLE	Core	See Table 2-2. See Section 18.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0. (R/W)
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
406H	1030	IA32_MC1_ADDR	Module	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the 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.
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
4C3H	1219	IA32_A_PMC2	Core	See Table 2-2.
4C4H	1220	IA32_A_PMC3	Core	See Table 2-2.
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	<b>Enhanced SMM Feature Control (SMM-Rw)</b> Reports SMM capability Enhancement. Accessible only while in SMM.
		0		<b>Lock (SMM-RwO)</b> When set to '1' locks this register from further changes
		1		Reserved
		2		<b>SMM_Code_Chk_En (SMM-Rw)</b> This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	<b>SMM Delayed (SMM-RO)</b> Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.
		N-1:0		<b>LOG_PROC_STATE (SMM-RO)</b> Each bit represents a processor core of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	<b>SMM Blocked (SMM-RO)</b> Reports the blocked state of all logical processors in the package. Available only while in SMM.
		N-1:0		<b>LOG_PROC_STATE (SMM-RO)</b> Each bit represents a processor core of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is 0FFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
500H	1280	IA32_SGX_SVN_STATUS	Core	<b>Status and SVN Threshold of SGX Support for ACM (RO).</b>
		0		<b>Lock.</b> See Section 4.1.1.3, "Interactions with Authenticated Code Modules (ACMs)"

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:1		Reserved.
		23:16		<b>SGX_SVN_SINIT</b> . See Section 41.11.3, "Interactions with Authenticated Code Modules (ACMs)"
		63:24		Reserved.
560H	1376	IA32_RTIT_OUTPUT_BASE	Core	<b>Trace Output Base Register (R/W)</b> . See Table 2-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Core	<b>Trace Output Mask Pointers Register (R/W)</b> . See Table 2-2.
570H	1392	IA32_RTIT_CTL	Core	<b>Trace Control Register (R/W)</b>
		0		<b>TraceEn</b>
		1		<b>CYCEn</b>
		2		<b>OS</b>
		3		<b>User</b>
		6:4		Reserved, MBZ
		7		<b>CR3 filter</b>
		8		<b>ToPA; writing 0 will #GP if also setting TraceEn</b>
		9		<b>MTCEn</b>
		10		<b>TSCEn</b>
		11		<b>DisRETC</b>
		12		Reserved, MBZ
		13		<b>BranchEn</b>
		17:14		<b>MTCFreq</b>
		18		Reserved, MBZ
		22:19		<b>CYCThresh</b>
		23		Reserved, MBZ
		27:24		<b>PSBFreq</b>
		31:28		Reserved, MBZ
		35:32		<b>ADDR0_CFG</b>
39:36		<b>ADDR1_CFG</b>		
63:40		Reserved, MBZ.		
571H	1393	IA32_RTIT_STATUS	Core	<b>Tracing Status Register (R/W)</b>
		0		<b>FilterEn</b> , writes ignored.
		1		<b>ContexEn</b> , writes ignored.
		2		<b>TriggerEn</b> , writes ignored.
		3		Reserved
		4		<b>Error (R/W)</b>
		5		<b>Stopped</b>
		31:6		Reserved. MBZ
		48:32		<b>PacketByteCnt</b>

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:49		Reserved, MBZ.
572H	1394	IA32_RTIT_CR3_MATCH	Core	<b>Trace Filter CR3 Match Register (R/W)</b>
		4:0		Reserved
		63:5		CR3[63:5] value to match
580H	1408	IA32_RTIT_ADDR0_A	Core	<b>Region 0 Start Address (R/W)</b>
		63:0		See Table 2-2.
581H	1409	IA32_RTIT_ADDR0_B	Core	<b>Region 0 End Address (R/W)</b>
		63:0		See Table 2-2.
582H	1410	IA32_RTIT_ADDR1_A	Core	<b>Region 1 Start Address (R/W)</b>
		63:0		See Table 2-2.
583H	1411	IA32_RTIT_ADDR1_B	Core	<b>Region 1 End Address (R/W)</b>
		63:0		See Table 2-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	<b>Unit Multipliers used in RAPL Interfaces (R/O)</b> See Section 14.9.1, "RAPL Interfaces."
		3:0		Power Units. Power related information (in Watts) is in unit of, $1W/2^{PU}$ ; where PU is an unsigned integer represented by bits 3:0. Default value is 1000b, indicating power unit is in 3.9 milliWatts increment.
		7:4		Reserved
		12:8		Energy Status Units. Energy related information (in Joules) is in unit of, $1\text{Joule}/(2^{ESU})$ ; where ESU is an unsigned integer represented by bits 12:8. Default value is 01110b, indicating energy unit is in 61 microJoules.
		15:13		Reserved
		19:16		Time Unit. Time related information (in seconds) is in unit of, $1S/2^{TU}$ ; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating power unit is in 0.977 millisecond.
		63:20		Reserved
60AH	1546	MSR_PKGC3_IRTL	Package	<b>Package C3 Interrupt Response Limit (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		<b>Interrupt response time limit (R/W)</b> Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		<b>Time Unit (R/W)</b> Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-19 for supported time unit encodings.
		14:13		Reserved.

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		15		<b>Valid (R/W)</b> 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.
60BH	1547	MSR_PKGC_IRTL1	Package	<b>Package C6/C7S Interrupt Response Limit 1 (R/W)</b> This MSR defines the interrupt response time limit used by the processor to manage transition to package C6 or C7S state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		<b>Interrupt response time limit (R/W)</b> Specifies the limit that should be used to decide if the package should be put into a package C6 or C7S state.
		12:10		<b>Time Unit (R/W)</b> Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-19 for supported time unit encodings
		14:13		Reserved.
		15		<b>Valid (R/W)</b> 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.
60CH	1548	MSR_PKGC_IRTL2	Package	<b>Package C7 Interrupt Response Limit 2 (R/W)</b> This MSR defines the interrupt response time limit used by the processor to manage transition to package C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		<b>Interrupt response time limit (R/W)</b> Specifies the limit that should be used to decide if the package should be put into a package C7 state.
		12:10		<b>Time Unit (R/W)</b> Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-19 for supported time unit encodings
		14:13		Reserved.
		15		<b>Valid (R/W)</b> 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		<b>Package C2 Residency Counter. (R/O)</b> Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
610H	1552	MSR_PKG_POWER_LIMIT	Package	<b>PKG RAPL Power Limit Control (R/W)</b> See Section 14.9.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	<b>PKG Energy Status (R/O)</b> See Section 14.9.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	<b>PKG Perf Status (R/O)</b> See Section 14.9.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	<b>PKG RAPL Parameters (R/W)</b>
		14:0		<b>Thermal Spec Power (R/W)</b> See Section 14.9.3, "Package RAPL Domain."
		15		Reserved.
		30:16		<b>Minimum Power (R/W)</b> See Section 14.9.3, "Package RAPL Domain."
		31		Reserved.
		46:32		<b>Maximum Power (R/W)</b> See Section 14.9.3, "Package RAPL Domain."
		47		Reserved.
		54:48		<b>Maximum Time Window (R/W)</b> Specified by $2^Y * (1.0 + Z/4.0) * \text{Time\_Unit}$ , where "Y" is the unsigned integer value represented by bits 52:48, "Z" is an unsigned integer represented by bits 54:53. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT
63:55		Reserved.		
618H	1560	MSR_DRAM_POWER_LIMIT	Package	<b>DRAM RAPL Power Limit Control (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	<b>DRAM Energy Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	<b>DRAM Performance Throttling Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	<b>DRAM RAPL Parameters (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names,
		63:0		<b>Package C10 Residency Counter. (R/O)</b> Value since last reset that the entire SOC is in an S0i3 state. Count at the same frequency as the TSC.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	<b>PP0 Energy Status (R/O)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	<b>PP1 Energy Status (R/O)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	<b>ConfigTDP Control (R/W)</b>

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		7:0		<b>MAX_NON_TURBO_RATIO (RW/L)</b> System BIOS can program this field.
		30:8		Reserved.
		31		<b>TURBO_ACTIVATION_RATIO_Lock (RW/L)</b> When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in Processor Cores (R/W) (frequency refers to processor core frequency)</b>
		0		<b>PROCHOT Status (R0)</b> When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal event.
		2		<b>Package-Level Power Limiting PL1 Status (R0)</b> When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		3		<b>Package-Level PL2 Power Limiting Status (R0)</b> When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		8:4		Reserved.
		9		<b>Core Power Limiting Status (R0)</b> When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		11		<b>Max Turbo Limit Status (R0)</b> When set, frequency is reduced below the operating system request due to multi-core turbo limits.
		12		<b>Electrical Design Point Status (R0)</b> When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		13		<b>Turbo Transition Attenuation Status (R0)</b> When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
		14		<b>Maximum Efficiency Frequency Status (R0)</b> When set, frequency is reduced below the maximum efficiency frequency.
15		<b>Reserved</b>		



Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		<b>Package-Level PL1 Power Limiting Log</b> When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		<b>Package-Level PL2 Power Limiting Log</b> When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24:20		Reserved.
		25		<b>Core Power Limiting Log</b> When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		<b>Max Turbo Limit Log</b> When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		<b>Electrical Design Point Log</b> When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		<b>Turbo Transition Attenuation Log</b> When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		<b>Maximum Efficiency Frequency Log</b> When set, indicates that the Maximum Efficiency Frequency Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:31		Reserved.

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description	
Hex	Dec				
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Core	<b>Last Branch Record 0 From IP (R/W)</b> One of 32 pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the <b>source instruction</b> . See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.6 and record format in Section 17.4.8.1</li> </ul>	
				0:47	<b>From Linear Address (R/W)</b>
				62:48	Signed extension of bits 47:0.
				63	Mispred
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Core	<b>Last Branch Record 1 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Core	<b>Last Branch Record 2 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Core	<b>Last Branch Record 3 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Core	<b>Last Branch Record 4 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Core	<b>Last Branch Record 5 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Core	<b>Last Branch Record 6 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Core	<b>Last Branch Record 7 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Core	<b>Last Branch Record 8 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Core	<b>Last Branch Record 9 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Core	<b>Last Branch Record 10 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Core	<b>Last Branch Record 11 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Core	<b>Last Branch Record 12 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Core	<b>Last Branch Record 13 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Core	<b>Last Branch Record 14 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Core	<b>Last Branch Record 15 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
690H	1680	MSR_LASTBRANCH_16_FROM_IP	Core	<b>Last Branch Record 16 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description	
Hex	Dec				
691H	1681	MSR_LASTBRANCH_17_FROM_IP	Core	<b>Last Branch Record 17 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
692H	1682	MSR_LASTBRANCH_18_FROM_IP	Core	<b>Last Branch Record 18 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
693H	1683	MSR_LASTBRANCH_19_FROM_IP	Core	<b>Last Branch Record 19 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
694H	1684	MSR_LASTBRANCH_20_FROM_IP	Core	<b>Last Branch Record 20 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
695H	1685	MSR_LASTBRANCH_21_FROM_IP	Core	<b>Last Branch Record 21 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
696H	1686	MSR_LASTBRANCH_22_FROM_IP	Core	<b>Last Branch Record 22 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
697H	1687	MSR_LASTBRANCH_23_FROM_IP	Core	<b>Last Branch Record 23 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
698H	1688	MSR_LASTBRANCH_24_FROM_IP	Core	<b>Last Branch Record 24 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
699H	1689	MSR_LASTBRANCH_25_FROM_IP	Core	<b>Last Branch Record 25 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
69AH	1690	MSR_LASTBRANCH_26_FROM_IP	Core	<b>Last Branch Record 26 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
69BH	1691	MSR_LASTBRANCH_27_FROM_IP	Core	<b>Last Branch Record 27 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
69CH	1692	MSR_LASTBRANCH_28_FROM_IP	Core	<b>Last Branch Record 28 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
69DH	1693	MSR_LASTBRANCH_29_FROM_IP	Core	<b>Last Branch Record 29 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
69EH	1694	MSR_LASTBRANCH_30_FROM_IP	Core	<b>Last Branch Record 30 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
69FH	1695	MSR_LASTBRANCH_31_FROM_IP	Core	<b>Last Branch Record 31 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.	
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Core	<b>Last Branch Record 0 To IP (R/W)</b> One of 32 pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the <b>Destination instruction</b> and elapsed cycles from last LBR update. See also: <ul style="list-style-type: none"> <li>▪ Section 17.6</li> </ul>	
				0:47	<b>Target Linear Address (R/W)</b>
				63:48	Elapsed cycles from last update to the LBR.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Core	<b>Last Branch Record 1 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.	

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Core	<b>Last Branch Record 2 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Core	<b>Last Branch Record 3 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Core	<b>Last Branch Record 4 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Core	<b>Last Branch Record 5 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Core	<b>Last Branch Record 6 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Core	<b>Last Branch Record 7 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Core	<b>Last Branch Record 8 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Core	<b>Last Branch Record 9 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Core	<b>Last Branch Record 10 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Core	<b>Last Branch Record 11 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Core	<b>Last Branch Record 12 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Core	<b>Last Branch Record 13 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Core	<b>Last Branch Record 14 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Core	<b>Last Branch Record 15 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D0H	1744	MSR_LASTBRANCH_16_TO_IP	Core	<b>Last Branch Record 16 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D1H	1745	MSR_LASTBRANCH_17_TO_IP	Core	<b>Last Branch Record 17 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_LASTBRANCH_18_TO_IP	Core	<b>Last Branch Record 18 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_LASTBRANCH_19_TO_IP	Core	<b>Last Branch Record 19 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_LASTBRANCH_20_TO_IP	Core	<b>Last Branch Record 20 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_LASTBRANCH_21_TO_IP	Core	<b>Last Branch Record 21 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
6D6H	1750	MSR_LASTBRANCH_22_TO_IP	Core	<b>Last Branch Record 22 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_LASTBRANCH_23_TO_IP	Core	<b>Last Branch Record 23 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_LASTBRANCH_24_TO_IP	Core	<b>Last Branch Record 24 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_LASTBRANCH_25_TO_IP	Core	<b>Last Branch Record 25 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_LASTBRANCH_26_TO_IP	Core	<b>Last Branch Record 26 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DBH	1755	MSR_LASTBRANCH_27_TO_IP	Core	<b>Last Branch Record 27 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_LASTBRANCH_28_TO_IP	Core	<b>Last Branch Record 28 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_LASTBRANCH_29_TO_IP	Core	<b>Last Branch Record 29 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DEH	1758	MSR_LASTBRANCH_30_TO_IP	Core	<b>Last Branch Record 30 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_LASTBRANCH_31_TO_IP	Core	<b>Last Branch Record 31 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Core	x2APIC ID register (R/O) See x2APIC Specification.
803H	2051	IA32_X2APIC_VERSION	Core	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Core	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Core	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Core	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Core	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Core	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR0	Core	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Core	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Core	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Core	x2APIC In-Service register bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Core	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Core	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Core	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Core	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Core	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Core	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Core	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Core	x2APIC Trigger Mode register bits [127:96] (R/O)

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
81CH	2076	IA32_X2APIC_TMR4	Core	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Core	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Core	x2APIC Trigger Mode register bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Core	x2APIC Trigger Mode register bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Core	x2APIC Interrupt Request register bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Core	x2APIC Interrupt Request register bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Core	x2APIC Interrupt Request register bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Core	x2APIC Interrupt Request register bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Core	x2APIC Interrupt Request register bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Core	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Core	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Core	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Core	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Core	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Core	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Core	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERM AL	Core	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Core	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Core	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Core	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Core	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Core	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Core	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Core	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Core	x2APIC Self IPI register (w/o)
C8FH	3215	IA32_PQR_ASSOC	Core	<b>Resource Association Register (R/W)</b>
		31:0		<b>Reserved</b>
		33:32		<b>COS (R/W).</b>
		63: 34		<b>Reserved</b>
D10H	3344	IA32_L2_QOS_MASK_0	Module	<b>L2 Class Of Service Mask - COS 0 (R/W)</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0
		0:7		<b>CBM: Bit vector of available L2 ways for COS 0 enforcement</b>
		63:8		<b>Reserved</b>
D11H	3345	IA32_L2_QOS_MASK_1	Module	<b>L2 Class Of Service Mask - COS 1 (R/W)</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1
		0:7		<b>CBM: Bit vector of available L2 ways for COS 0 enforcement</b>
		63:8		<b>Reserved</b>

**Table 2-12. MSRs in Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
D12H	3346	IA32_L2_QOS_MASK_2	Module	<b>L2 Class Of Service Mask - COS 2 (R/W)</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2
		0:7		<b>CBM: Bit vector of available L2 ways for COS 0 enforcement</b>
		63:8		<b>Reserved</b>
D13H	3347	IA32_L2_QOS_MASK_3	Package	<b>L2 Class Of Service Mask - COS 3 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3
		0:19		<b>CBM: Bit vector of available L2 ways for COS 3 enforcement</b>
		63:20		<b>Reserved</b>
D90H	3472	IA32_BNDCFGS	Core	See Table 2-2.
DA0H	3488	IA32_XSS	Core	See Table 2-2.

See Table 2-6, and Table 2-12 for MSR definitions applicable to processors with CPUID signature 06\_5CH.

## 2.6 MSRS IN INTEL ATOM PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Intel Atom processors based on the Goldmont Plus microarchitecture support MSRs listed in Table 2-6, Table 2-12 and Table 2-13. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_7AH; see Table 2-1. For an MSR listed in Table 2-13 that also appears in the model-specific tables of prior generations, Table 2-13 supercede prior generation tables.

In the Goldmont Plus microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a pair of processor cores in the physical package. “Package” means all processor cores in the physical package share the same MSR or bit interface.

**Table 2-13. MSRs in Intel Atom Processors Based on the Goldmont Plus Microarchitecture**

Address		Register Name	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Core	<b>Control Features in Intel 64Processor (R/W)</b> See Table 2-2.
		0		<b>Lock (R/WL)</b>
		1		<b>Enable VMX inside SMX operation (R/WL)</b>
		2		<b>Enable VMX outside SMX operation (R/WL)</b>
		14:8		<b>SENTER local functions enables (R/WL)</b>
		15		<b>SENTER global functions enable (R/WL)</b>
		17		<b>SGX Launch Control Enable (R/WL)</b> This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR.  Valid if CPUID.(EAX=07H, ECX=0H): ECX[30] = 1.
		18		<b>SGX global functions enable (R/WL)</b>
		63:19		Reserved.



**Table 2-13. MSRs in Intel Atom Processors Based on the Goldmont Plus Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
8CH	140	IA32_SGXLEPUBKEYHASH0	Core	See Table 2-2.
8DH	141	IA32_SGXLEPUBKEYHASH1	Core	See Table 2-2.
8EH	142	IA32_SGXLEPUBKEYHASH2	Core	See Table 2-2.
8FH	143	IA32_SGXLEPUBKEYHASH3	Core	See Table 2-2.
3F1H	1009	MSR_PEBS_ENABLE	Core	See Table 2-2. See Section 18.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0. (R/W)
		1		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC1.
		2		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC2.
		3		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC3.
		31:4		Reserved.
		32		Enable PEBS trigger and recording for IA32_FIXED_CTR0.
		33		Enable PEBS trigger and recording for IA32_FIXED_CTR1.
		34		Enable PEBS trigger and recording for IA32_FIXED_CTR2.
		63:35		Reserved.
570H	1392	IA32_RTIT_CTL	Core	<b>Trace Control Register (R/W)</b>
		0		<b>TraceEn</b>
		1		<b>CYCEn</b>
		2		<b>OS</b>
		3		<b>User</b>
		4		<b>PwrEvtEn</b>
		5		<b>FUPonPTW</b>
		6		<b>FabricEn</b>
		7		<b>CR3 filter</b>
		8		<b>ToPA; writing 0 will #GP if also setting TraceEn</b>
		9		<b>MTCEn</b>
		10		<b>TSCEn</b>
		11		<b>DisRETC</b>
		12		<b>PTWEn</b>
		13		<b>BranchEn</b>
		17:14		<b>MTCFreq</b>
		18		Reserved, MBZ
		22:19		<b>CYCThresh</b>
23		Reserved, MBZ		
27:24		<b>PSBFreq</b>		



Table 2-13. MSRs in Intel Atom Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:28		Reserved, MBZ
		35:32		<b>ADDR0_CFG</b>
		39:36		<b>ADDR1_CFG</b>
		63:40		Reserved, MBZ.
680H	1664	MSR_ LASTBRANCH_0_FROM_IP	Core	<b>Last Branch Record 0 From IP (R/W)</b> One of the three MSRs that make up the first entry of the 32-entry LBR stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> <li>Last Branch Record Stack TOS at 1C9H</li> <li>Section 17.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."</li> </ul>
681H - 69FH	1665 - 1695	MSR_ LASTBRANCH_i_FROM_IP	Core	<b>Last Branch Record i From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP; i = 1-31.
6C0H	1728	MSR_ LASTBRANCH_0_TO_IP	Core	<b>Last Branch Record 0 To IP (R/W)</b> One of the 3 MSRs that make up the first entry of the 32-entry LBR stack. The To_IP part of the stack contains pointers to the Destination instruction. See also: <ul style="list-style-type: none"> <li>Section 17.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."</li> </ul>
6C1H - 6DFH	1729 - 1759	MSR_ LASTBRANCH_i_TO_IP	Core	<b>Last Branch Record i To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP; i = 1-31.
DC0H	3520	MSR_LASTBRANCH_INFO_ 0	Core	<b>Last Branch Record 0 Additional Information (R/W)</b> One of the 3 MSRs that make up the first entry of the 32-entry LBR stack. This part of the stack contains flag and elapsed cycle information. See also: <ul style="list-style-type: none"> <li>Last Branch Record Stack TOS at 1C9H</li> <li>Section 17.9.1, "LBR Stack."</li> </ul>
DC1H	3521	MSR_LASTBRANCH_INFO_ 1	Core	<b>Last Branch Record 1 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DC2H	3522	MSR_LASTBRANCH_INFO_ 2	Core	<b>Last Branch Record 2 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DC3H	3523	MSR_LASTBRANCH_INFO_ 3	Core	<b>Last Branch Record 3 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DC4H	3524	MSR_LASTBRANCH_INFO_ 4	Core	<b>Last Branch Record 4 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DC5H	3525	MSR_LASTBRANCH_INFO_ 5	Core	<b>Last Branch Record 5 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DC6H	3526	MSR_LASTBRANCH_INFO_ 6	Core	<b>Last Branch Record 6 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.

Table 2-13. MSRs in Intel Atom Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
DC7H	3527	MSR_LASTBRANCH_INFO_7	Core	<b>Last Branch Record 7 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DC8H	3528	MSR_LASTBRANCH_INFO_8	Core	<b>Last Branch Record 8 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DC9H	3529	MSR_LASTBRANCH_INFO_9	Core	<b>Last Branch Record 9 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DCAH	3530	MSR_LASTBRANCH_INFO_10	Core	<b>Last Branch Record 10 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DCBH	3531	MSR_LASTBRANCH_INFO_11	Core	<b>Last Branch Record 11 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DCCH	3532	MSR_LASTBRANCH_INFO_12	Core	<b>Last Branch Record 12 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DCDH	3533	MSR_LASTBRANCH_INFO_13	Core	<b>Last Branch Record 13 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DCEH	3534	MSR_LASTBRANCH_INFO_14	Core	<b>Last Branch Record 14 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DCFH	3535	MSR_LASTBRANCH_INFO_15	Core	<b>Last Branch Record 15 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD0H	3536	MSR_LASTBRANCH_INFO_16	Core	<b>Last Branch Record 16 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD1H	3537	MSR_LASTBRANCH_INFO_17	Core	<b>Last Branch Record 17 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD2H	3538	MSR_LASTBRANCH_INFO_18	Core	<b>Last Branch Record 18 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD3H	3539	MSR_LASTBRANCH_INFO_19	Core	<b>Last Branch Record 19 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD4H	3520	MSR_LASTBRANCH_INFO_20	Core	<b>Last Branch Record 20 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD5H	3521	MSR_LASTBRANCH_INFO_21	Core	<b>Last Branch Record 21 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD6H	3522	MSR_LASTBRANCH_INFO_22	Core	<b>Last Branch Record 22 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD7H	3523	MSR_LASTBRANCH_INFO_23	Core	<b>Last Branch Record 23 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD8H	3524	MSR_LASTBRANCH_INFO_24	Core	<b>Last Branch Record 24 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DD9H	3525	MSR_LASTBRANCH_INFO_25	Core	<b>Last Branch Record 25 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.
DDAH	3526	MSR_LASTBRANCH_INFO_26	Core	<b>Last Branch Record 26 Additional Information (R/w)</b> See description of MSR_LASTBRANCH_INFO_0.

**Table 2-13. MSRs in Intel Atom Processors Based on the Goldmont Plus Microarchitecture (Contd.)**

Address		Register Name	Scope	Bit Description
Hex	Dec			
DDBH	3527	MSR_LASTBRANCH_INFO_27	Core	<b>Last Branch Record 27 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DDCH	3528	MSR_LASTBRANCH_INFO_28	Core	<b>Last Branch Record 28 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DDDH	3529	MSR_LASTBRANCH_INFO_29	Core	<b>Last Branch Record 29 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DDEH	3530	MSR_LASTBRANCH_INFO_30	Core	<b>Last Branch Record 30 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.
DDFH	3531	MSR_LASTBRANCH_INFO_31	Core	<b>Last Branch Record 31 Additional Information (R/W)</b> See description of MSR_LASTBRANCH_INFO_0.

See Table 2-6, Table 2-12 and Table 2-13 for MSR definitions applicable to processors with CPUID signature 06\_7AH.

## 2.7 MSRS IN THE INTEL® MICROARCHITECTURE CODE NAME NEHALEM

Table 2-14 lists model-specific registers (MSRs) that are common for Intel® microarchitecture code name Nehalem. These include Intel Core i7 and i5 processor family. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_1AH, 06\_1EH, 06\_1FH, 06\_2EH, see Table 2-1. Additional MSRs specific to 06\_1AH, 06\_1EH, 06\_1FH are listed in Table 2-15. Some MSRs listed in these tables are used by BIOS. More information about these MSR can be found at <http://biosbits.org>.

The column "Scope" represents the package/core/thread scope of individual bit field of an MSR. "Thread" means this bit field must be programmed on each logical processor independently. "Core" means the bit field must be programmed on each processor core independently, logical processors in the same core will be affected by change of this bit on the other logical processor in the same core. "Package" means the bit field must be programmed once for each physical package. Change of a bit filed with a package scope will affect all logical processors in that physical package.

**Table 2-14. 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 2.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 2.22, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	<b>Platform ID (R)</b> See Table 2-2.
17H	23	MSR_PLATFORM_ID	Package	<b>Model Specific Platform ID (R)</b>
		49:0		Reserved.
		52:50		See Table 2-2.

**Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	<b>SMI Counter (R/O)</b>
		31:0		<b>SMI Count (R/O)</b> Running count of SMI events since last RESET.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	<b>Control Features in Intel 64Processor (R/W)</b> See Table 2-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	<b>BIOS Update Trigger Register (W)</b> See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	<b>BIOS Update Signature ID (RO)</b> See Table 2-2.
C1H	193	IA32_PMC0	Thread	<b>Performance Counter Register</b> See Table 2-2.
C2H	194	IA32_PMC1	Thread	<b>Performance Counter Register</b> See Table 2-2.
C3H	195	IA32_PMC2	Thread	<b>Performance Counter Register</b> See Table 2-2.
C4H	196	IA32_PMC3	Thread	<b>Performance Counter Register</b> See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> 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	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> 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	<b>Programmable TDC-TDP Limit for Turbo Mode (R/O)</b> 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.

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		47:40	Package	<b>Maximum Efficiency Ratio (R/O)</b> 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	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See <a href="http://biosbits.org">http://biosbits.org</a> .
		2:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power), for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-sate support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		<b>I/O MWAIT Redirection Enable (R/W)</b> 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		<b>CFG Lock (R/WO)</b> When set, lock bits 15:0 of this register until next reset.
		23:16		Reserved.
		24		<b>Interrupt filtering enable (R/W)</b> 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		<b>C3 state auto demotion enable (R/W)</b> When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		<b>C1 state auto demotion enable (R/W)</b> When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		<b>Enable C3 Undemotion (R/W)</b>
		28		<b>Enable C1 Undemotion (R/W)</b>

**Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		29		<b>Package C State Demotion Enable (R/W)</b>
		30		<b>Package C State UnDemotion Enable (R/W)</b>
		63:31		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	<b>Power Management IO Redirection in C-state (R/W)</b> See <a href="http://biosbits.org">http://biosbits.org</a> .
		15:0		<b>LVL_2 Base Address (R/W)</b> 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		<b>C-state Range (R/W)</b> Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	<b>Maximum Performance Frequency Clock Count (RW)</b> See Table 2-2.
E8H	232	IA32_APERF	Thread	<b>Actual Performance Frequency Clock Count (RW)</b> See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 2-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status
		0		<b>RIPV</b> 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		<b>EIPV</b> When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		<b>MCIP</b> 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	Thread	See Table 2-2.
		7:0		<b>Event Select</b>
		15:8		<b>UMask</b>
		16		<b>USR</b>
		17		<b>OS</b>
		18		<b>Edge</b>
		19		<b>PC</b>
		20		<b>INT</b>
		21		<b>AnyThread</b>
		22		<b>EN</b>
		23		<b>INV</b>
		31:24		<b>CMASK</b>
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 2-2.
198H	408	IA32_PERF_STATUS	Core	See Table 2-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	<b>Clock Modulation (R/W)</b> See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		0		Reserved.
		3:1		<b>On demand Clock Modulation Duty Cycle (R/W)</b>
		4		<b>On demand Clock Modulation Enable (R/W)</b>
		63:5		Reserved.
19BH	411	IA32_THERM_INTERRUPT	Core	<b>Thermal Interrupt Control (R/W)</b> See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.

**Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A0H	416	IA32_MISC_ENABLE		<b>Enable Misc. Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.
		0	Thread	<b>Fast-Strings Enable</b> See Table 2-2.
		2:1		Reserved.
		3	Thread	<b>Automatic Thermal Control Circuit Enable (R/W)</b> See Table 2-2. Default value is 1.
		6:4		Reserved.
		7	Thread	<b>Performance Monitoring Available (R)</b> See Table 2-2.
		10:8		Reserved.
		11	Thread	<b>Branch Trace Storage Unavailable (RO)</b> See Table 2-2.
		12	Thread	<b>Processor Event Based Sampling Unavailable (RO)</b> See Table 2-2.
		15:13		Reserved.
		16	Package	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> See Table 2-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 2-2.
		21:19		Reserved.
		22	Thread	<b>Limit CPUID Maxval (R/W)</b> See Table 2-2.
		23	Thread	<b>xTPR Message Disable (R/W)</b> See Table 2-2.
		33:24		Reserved.
		34	Thread	<b>XD Bit Disable (R/W)</b> See Table 2-2.
		37:35		Reserved.
		38	Package	<b>Turbo Mode Disable (R/W)</b> 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. <b>Note:</b> 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 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A2H	418	MSR_TEMPERATURE_TARGET	Thread	Temperature Target
		15:0		Reserved.
		23:16		<b>Temperature Target (R)</b> The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A4H	420	MSR_MISC_FEATURE_CONTROL		<b>Miscellaneous Feature Control (R/W)</b>
		0	Core	<b>L2 Hardware Prefetcher Disable (R/W)</b> If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1	Core	<b>L2 Adjacent Cache Line Prefetcher Disable (R/W)</b> If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).
		2	Core	<b>DCU Hardware Prefetcher Disable (R/W)</b> If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3	Core	<b>DCU IP Prefetcher Disable (R/W)</b> If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction Pointer of previous loads) to determine whether to prefetch additional lines.
		63:4		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	<b>Offcore Response Event Select Register (R/W)</b>
1AAH	426	MSR_MISC_PWR_MGMT		Miscellaneous Power Management Control; Various model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		0	Package	<b>EIST Hardware Coordination Disable (R/W)</b> When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores; When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.
		1	Thread	<b>Energy/Performance Bias Enable (R/W)</b> This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.(EAX=06h):ECX[3].
	63:2		Reserved.	
1ACH	428	MSR_TURBO_POWER_CURRENT_LIMIT		See <a href="http://biosbits.org">http://biosbits.org</a> .
		14:0	Package	<b>TDP Limit (R/W)</b> TDP limit in 1/8 Watt granularity.

**Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15	Package	<b>TDP Limit Override Enable (R/W)</b> A value = 0 indicates override is not active, and a value = 1 indicates active.
		30:16	Package	<b>TDC Limit (R/W)</b> TDC limit in 1/8 Amp granularity.
		31	Package	<b>TDC Limit Override Enable (R/W)</b> A value = 0 indicates override is not active, and a value = 1 indicates active.
		63:32		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> <b>RO</b> if MSR_PLATFORM_INFO.[28] = 0, <b>RW</b> if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
1C8H	456	MSR_LBR_SELECT	Core	<b>Last Branch Record Filtering Select Register (R/W)</b> See Section 17.9.2, "Filtering of Last Branch Records."
		0		<b>CPL_EQ_0</b>
		1		<b>CPL_NEQ_0</b>
		2		<b>JCC</b>
		3		<b>NEAR_REL_CALL</b>
		4		<b>NEAR_IND_CALL</b>
		5		<b>NEAR_RET</b>
		6		<b>NEAR_IND_JMP</b>
		7		<b>NEAR_REL_JMP</b>
		8		<b>FAR_BRANCH</b>
63:9		Reserved.		
1C9H	457	MSR_LASTBRANCH_TOS	Thread	<b>Last Branch Record Stack TOS (R/W)</b> 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).

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Thread	<b>Debug Control (R/W)</b> See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	<b>Last Exception Record From Linear IP (R)</b> 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	<b>Last Exception Record To Linear IP (R)</b> This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See <a href="http://biosbits.org">http://biosbits.org</a> .
		0		Reserved.
		1	Package	<b>C1E Enable (R/W)</b> 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 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 2-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 2-2.

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 2-2.
277H	631	IA32_PAT	Thread	See Table 2-2.
280H	640	IA32_MC0_CTL2	Package	See Table 2-2.
281H	641	IA32_MC1_CTL2	Package	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Core	See Table 2-2.
285H	645	IA32_MC5_CTL2	Core	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	<b>Default Memory Types (R/W)</b> See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	<b>Fixed-Function Performance Counter Register 0 (R/W)</b> See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	<b>Fixed-Function Performance Counter Register 1 (R/W)</b> See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	<b>Fixed-Function Performance Counter Register 2 (R/W)</b> See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format. See Table 2-2.

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 2-2.
		11:8		PEBS_REC_FORMAT. See Table 2-2.
		12		SMM_FREEZE. See Table 2-2.
		63:13		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	<b>Fixed-Function-Counter Control Register (R/W)</b> See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Thread	Provides single-bit status used by software to query the overflow condition of each performance counter. (RO)
		61		<b>UNC_Ovf</b> Uncore overflowed if 1.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities." Allows software to clear counter overflow conditions on any combination of fixed-function PMCs (MSR_PERF_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Thread	<b>(R/W)</b>
		61		<b>CLR_UNC_Ovf</b> Set 1 to clear UNC_Ovf.
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
63:36		Reserved.		
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 18.3.1.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
		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	IA32_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."

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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	IA32_MC1_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
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	IA32_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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.
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the 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	IA32_MC4_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
414H	1044	IA32_MC5_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	IA32_MC5_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	IA32_MC5_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
41BH	1051	IA32_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRs."
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCI_CTL MSRs."
41DH	1053	IA32_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
41EH	1054	IA32_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRs."
41FH	1055	IA32_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRs."
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCI_CTL MSRs."
421H	1057	IA32_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
422H	1058	IA32_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRs."
423H	1059	IA32_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRs."
480H	1152	IA32_VMX_BASIC	Thread	<b>Reporting Register of Basic VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Thread	<b>Capability Reporting Register of Pin-based VM-execution Controls (R/O)</b> See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Thread	<b>Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Thread	<b>Capability Reporting Register of VM-exit Controls (R/O)</b> See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Thread	<b>Capability Reporting Register of VM-entry Controls (R/O)</b> See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	<b>Reporting Register of Miscellaneous VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Thread	<b>Capability Reporting Register of CRO Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Thread	<b>Capability Reporting Register of CRO Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	<b>Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."



Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
489H	1161	IA32_VMX_CR4_FIXED1	Thread	<b>Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	<b>Capability Reporting Register of VMCS Field Enumeration (R/O).</b> See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLDS2	Thread	<b>Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Thread	<b>DS Save Area (R/W)</b> See Table 2-2. See Section 18.6.3.4, "Debug Store (DS) Mechanism."
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	<b>Last Branch Record 0 From IP (R/W)</b> One of sixteen pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the <b>source instruction</b> . See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.9.1 and record format in Section 17.4.8.1</li> </ul>
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	<b>Last Branch Record 1 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	<b>Last Branch Record 2 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	<b>Last Branch Record 3 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	<b>Last Branch Record 4 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	<b>Last Branch Record 5 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	<b>Last Branch Record 6 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	<b>Last Branch Record 7 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	<b>Last Branch Record 8 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	<b>Last Branch Record 9 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	<b>Last Branch Record 10 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	<b>Last Branch Record 11 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	<b>Last Branch Record 12 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	<b>Last Branch Record 13 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	<b>Last Branch Record 14 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	<b>Last Branch Record 15 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	<b>Last Branch Record 0 To IP (R/W)</b> One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	<b>Last Branch Record 1 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	<b>Last Branch Record 2 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	<b>Last Branch Record 3 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	<b>Last Branch Record 4 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	<b>Last Branch Record 5 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	<b>Last Branch Record 6 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	<b>Last Branch Record 7 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	<b>Last Branch Record 8 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	<b>Last Branch Record 9 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	<b>Last Branch Record 10 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	<b>Last Branch Record 11 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	<b>Last Branch Record 12 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	<b>Last Branch Record 13 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	<b>Last Branch Record 14 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	<b>Last Branch Record 15 To IP (R/W)</b> 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)

Table 2-14. MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI register (w/O)
C000_0080H		IA32_EFER	Thread	<b>Extended Feature Enables</b> See Table 2-2.
C000_0081H		IA32_STAR	Thread	<b>System Call Target Address (R/W)</b> See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	<b>IA-32e Mode System Call Target Address (R/W)</b> See Table 2-2.
C000_0084H		IA32_FMASK	Thread	<b>System Call Flag Mask (R/W)</b> See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	<b>Map of BASE Address of FS (R/W)</b> See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	<b>Map of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	<b>Swap Target of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	<b>AUXILIARY TSC Signature. (R/W)</b> See Table 2-2 and Section 17.17.2, "IA32_TSC_AUX Register and RDTSCP Support."

## 2.7.1 Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

Intel Xeon Processor 5500 and 3400 series support additional model-specific registers listed in Table 2-15. These MSRs also apply to Intel Core i7 and i5 processor family CPUID signature with DisplayFamily\_DisplayModel of 06\_1AH, 06\_1EH and 06\_1FH, see Table 2-1.

Table 2-15. Additional MSRs in Intel® Xeon® Processor 5500 and 3400 Series

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Actual maximum turbo frequency is multiplied by 133.33MHz. (not available to model 06_2EH)
		7:0		<b>Maximum Turbo Ratio Limit 1C (R/O)</b> Maximum Turbo mode ratio limit with 1 core active.
		15:8		<b>Maximum Turbo Ratio Limit 2C (R/O)</b> Maximum Turbo mode ratio limit with 2 cores active.
		23:16		<b>Maximum Turbo Ratio Limit 3C (R/O)</b> Maximum Turbo mode ratio limit with 3 cores active.
		31:24		<b>Maximum Turbo Ratio Limit 4C (R/O)</b> Maximum Turbo mode ratio limit with 4 cores active.
		63:32		Reserved.
301H	769	MSR_GQ_SNOOP_MESF	Package	
		0		<b>From M to S (R/W)</b>
		1		<b>From E to S (R/W)</b>
		2		<b>From S to S (R/W)</b>
		3		<b>From F to S (R/W)</b>
		4		<b>From M to I (R/W)</b>
		5		<b>From E to I (R/W)</b>
		6		<b>From S to I (R/W)</b>
		7		<b>From F to I (R/W)</b>
63:8		Reserved.		
391H	913	MSR_UNCORE_PERF_GLOBAL_CTRL	Package	See Section 18.3.1.2.1, "Uncore Performance Monitoring Management Facility."
392H	914	MSR_UNCORE_PERF_GLOBAL_STATUS	Package	See Section 18.3.1.2.1, "Uncore Performance Monitoring Management Facility."
393H	915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	Package	See Section 18.3.1.2.1, "Uncore Performance Monitoring Management Facility."
394H	916	MSR_UNCORE_FIXED_CTR0	Package	See Section 18.3.1.2.1, "Uncore Performance Monitoring Management Facility."
395H	917	MSR_UNCORE_FIXED_CTR_CTRL	Package	See Section 18.3.1.2.1, "Uncore Performance Monitoring Management Facility."
396H	918	MSR_UNCORE_ADDR_OPCODE_MATCH	Package	See Section 18.3.1.2.3, "Uncore Address/Opcod Match MSR."
3B0H	960	MSR_UNCORE_PMC0	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B1H	961	MSR_UNCORE_PMC1	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B2H	962	MSR_UNCORE_PMC2	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."

**Table 2-15. Additional MSRs in Intel® Xeon® Processor 5500 and 3400 Series (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3B3H	963	MSR_UNCORE_PMC3	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B4H	964	MSR_UNCORE_PMC4	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B6H	966	MSR_UNCORE_PMC6	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B7H	967	MSR_UNCORE_PMC7	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C0H	944	MSR_UNCORE_PERFEVTSEL0	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C1H	945	MSR_UNCORE_PERFEVTSEL1	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C2H	946	MSR_UNCORE_PERFEVTSEL2	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C3H	947	MSR_UNCORE_PERFEVTSEL3	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C4H	948	MSR_UNCORE_PERFEVTSEL4	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C5H	949	MSR_UNCORE_PERFEVTSEL5	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C6H	950	MSR_UNCORE_PERFEVTSEL6	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C7H	951	MSR_UNCORE_PERFEVTSEL7	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."

### 2.7.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

Intel Xeon Processor 7500 series support MSRs listed in Table 2-14 (except MSR address 1ADH) and additional model-specific registers listed in Table 2-16. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_2EH.

**Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Reserved</b> Attempt to read/write will cause #UD.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
394H	816	MSR_W_PMON_FIXED_CTR	Package	Uncore W-box perfmon fixed counter
395H	817	MSR_W_PMON_FIXED_CTR_CTL	Package	Uncore U-box perfmon fixed counter control MSR
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	IA32_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
426H	1062	IA32_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	IA32_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	IA32_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42AH	1066	IA32_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	IA32_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	IA32_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42EH	1070	IA32_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	IA32_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	IA32_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
432H	1074	IA32_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	IA32_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	IA32_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
436H	1078	IA32_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	IA32_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	IA32_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43AH	1082	IA32_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	IA32_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."



Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
43DH	1085	IA32_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43EH	1086	IA32_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	IA32_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	IA32_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
442H	1090	IA32_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	IA32_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	IA32_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
446H	1094	IA32_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	IA32_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	IA32_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44AH	1098	IA32_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	IA32_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	IA32_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44EH	1102	IA32_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	IA32_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
451H	1105	IA32_MC20_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
452H	1106	IA32_MC20_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
453H	1107	IA32_MC20_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
455H	1109	IA32_MC21_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
456H	1110	IA32_MC21_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
457H	1111	IA32_MC21_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
C00H	3072	MSR_U_PMON_GLOBAL_CTRL	Package	Uncore U-box perfmon global control MSR.
C01H	3073	MSR_U_PMON_GLOBAL_STATUS	Package	Uncore U-box perfmon global status MSR.
C02H	3074	MSR_U_PMON_GLOBAL_OVF_CTRL	Package	Uncore U-box perfmon global overflow control MSR.
C10H	3088	MSR_U_PMON_EVNT_SEL	Package	Uncore U-box perfmon event select MSR.
C11H	3089	MSR_U_PMON_CTR	Package	Uncore U-box perfmon counter MSR.
C20H	3104	MSR_B0_PMON_BOX_CTRL	Package	Uncore B-box 0 perfmon local box control MSR.



Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C21H	3105	MSR_B0_PMON_BOX_STATUS	Package	Uncore B-box 0 perfmon local box status MSR.
C22H	3106	MSR_B0_PMON_BOX_OVF_CTRL	Package	Uncore B-box 0 perfmon local box overflow control MSR.
C30H	3120	MSR_B0_PMON_EVNT_SELO	Package	Uncore B-box 0 perfmon event select MSR.
C31H	3121	MSR_B0_PMON_CTR0	Package	Uncore B-box 0 perfmon counter MSR.
C32H	3122	MSR_B0_PMON_EVNT_SEL1	Package	Uncore B-box 0 perfmon event select MSR.
C33H	3123	MSR_B0_PMON_CTR1	Package	Uncore B-box 0 perfmon counter MSR.
C34H	3124	MSR_B0_PMON_EVNT_SEL2	Package	Uncore B-box 0 perfmon event select MSR.
C35H	3125	MSR_B0_PMON_CTR2	Package	Uncore B-box 0 perfmon counter MSR.
C36H	3126	MSR_B0_PMON_EVNT_SEL3	Package	Uncore B-box 0 perfmon event select MSR.
C37H	3127	MSR_B0_PMON_CTR3	Package	Uncore B-box 0 perfmon counter MSR.
C40H	3136	MSR_S0_PMON_BOX_CTRL	Package	Uncore S-box 0 perfmon local box control MSR.
C41H	3137	MSR_S0_PMON_BOX_STATUS	Package	Uncore S-box 0 perfmon local box status MSR.
C42H	3138	MSR_S0_PMON_BOX_OVF_CTRL	Package	Uncore S-box 0 perfmon local box overflow control MSR.
C50H	3152	MSR_S0_PMON_EVNT_SELO	Package	Uncore S-box 0 perfmon event select MSR.
C51H	3153	MSR_S0_PMON_CTR0	Package	Uncore S-box 0 perfmon counter MSR.
C52H	3154	MSR_S0_PMON_EVNT_SEL1	Package	Uncore S-box 0 perfmon event select MSR.
C53H	3155	MSR_S0_PMON_CTR1	Package	Uncore S-box 0 perfmon counter MSR.
C54H	3156	MSR_S0_PMON_EVNT_SEL2	Package	Uncore S-box 0 perfmon event select MSR.
C55H	3157	MSR_S0_PMON_CTR2	Package	Uncore S-box 0 perfmon counter MSR.
C56H	3158	MSR_S0_PMON_EVNT_SEL3	Package	Uncore S-box 0 perfmon event select MSR.
C57H	3159	MSR_S0_PMON_CTR3	Package	Uncore S-box 0 perfmon counter MSR.
C60H	3168	MSR_B1_PMON_BOX_CTRL	Package	Uncore B-box 1 perfmon local box control MSR.
C61H	3169	MSR_B1_PMON_BOX_STATUS	Package	Uncore B-box 1 perfmon local box status MSR.
C62H	3170	MSR_B1_PMON_BOX_OVF_CTRL	Package	Uncore B-box 1 perfmon local box overflow control MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C70H	3184	MSR_B1_PMON_EVNT_SELO	Package	Uncore B-box 1 perfmon event select MSR.
C71H	3185	MSR_B1_PMON_CTRL0	Package	Uncore B-box 1 perfmon counter MSR.
C72H	3186	MSR_B1_PMON_EVNT_SEL1	Package	Uncore B-box 1 perfmon event select MSR.
C73H	3187	MSR_B1_PMON_CTRL1	Package	Uncore B-box 1 perfmon counter MSR.
C74H	3188	MSR_B1_PMON_EVNT_SEL2	Package	Uncore B-box 1 perfmon event select MSR.
C75H	3189	MSR_B1_PMON_CTRL2	Package	Uncore B-box 1 perfmon counter MSR.
C76H	3190	MSR_B1_PMON_EVNT_SEL3	Package	Uncore B-box 1 vperfmon event select MSR.
C77H	3191	MSR_B1_PMON_CTRL3	Package	Uncore B-box 1 perfmon counter MSR.
C80H	3120	MSR_W_PMON_BOX_CTRL	Package	Uncore W-box perfmon local box control MSR.
C81H	3121	MSR_W_PMON_BOX_STATUS	Package	Uncore W-box perfmon local box status MSR.
C82H	3122	MSR_W_PMON_BOX_OVF_CTRL	Package	Uncore W-box perfmon local box overflow control MSR.
C90H	3136	MSR_W_PMON_EVNT_SELO	Package	Uncore W-box perfmon event select MSR.
C91H	3137	MSR_W_PMON_CTRL0	Package	Uncore W-box perfmon counter MSR.
C92H	3138	MSR_W_PMON_EVNT_SEL1	Package	Uncore W-box perfmon event select MSR.
C93H	3139	MSR_W_PMON_CTRL1	Package	Uncore W-box perfmon counter MSR.
C94H	3140	MSR_W_PMON_EVNT_SEL2	Package	Uncore W-box perfmon event select MSR.
C95H	3141	MSR_W_PMON_CTRL2	Package	Uncore W-box perfmon counter MSR.
C96H	3142	MSR_W_PMON_EVNT_SEL3	Package	Uncore W-box perfmon event select MSR.
C97H	3143	MSR_W_PMON_CTRL3	Package	Uncore W-box perfmon counter MSR.
CA0H	3232	MSR_M0_PMON_BOX_CTRL	Package	Uncore M-box 0 perfmon local box control MSR.
CA1H	3233	MSR_M0_PMON_BOX_STATUS	Package	Uncore M-box 0 perfmon local box status MSR.
CA2H	3234	MSR_M0_PMON_BOX_OVF_CTRL	Package	Uncore M-box 0 perfmon local box overflow control MSR.
CA4H	3236	MSR_M0_PMON_TIMESTAMP	Package	Uncore M-box 0 perfmon time stamp unit select MSR.
CA5H	3237	MSR_M0_PMON_DSP	Package	Uncore M-box 0 perfmon DSP unit select MSR.
CA6H	3238	MSR_M0_PMON_ISS	Package	Uncore M-box 0 perfmon ISS unit select MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CA7H	3239	MSR_M0_PMON_MAP	Package	Uncore M-box 0 perfmon MAP unit select MSR.
CA8H	3240	MSR_M0_PMON_MSC_THR	Package	Uncore M-box 0 perfmon MIC THR select MSR.
CA9H	3241	MSR_M0_PMON_PGT	Package	Uncore M-box 0 perfmon PGT unit select MSR.
CAAH	3242	MSR_M0_PMON_PLD	Package	Uncore M-box 0 perfmon PLD unit select MSR.
CABH	3243	MSR_M0_PMON_ZDP	Package	Uncore M-box 0 perfmon ZDP unit select MSR.
CB0H	3248	MSR_M0_PMON_EVNT_SEL0	Package	Uncore M-box 0 perfmon event select MSR.
CB1H	3249	MSR_M0_PMON_CTRL0	Package	Uncore M-box 0 perfmon counter MSR.
CB2H	3250	MSR_M0_PMON_EVNT_SEL1	Package	Uncore M-box 0 perfmon event select MSR.
CB3H	3251	MSR_M0_PMON_CTRL1	Package	Uncore M-box 0 perfmon counter MSR.
CB4H	3252	MSR_M0_PMON_EVNT_SEL2	Package	Uncore M-box 0 perfmon event select MSR.
CB5H	3253	MSR_M0_PMON_CTRL2	Package	Uncore M-box 0 perfmon counter MSR.
CB6H	3254	MSR_M0_PMON_EVNT_SEL3	Package	Uncore M-box 0 perfmon event select MSR.
CB7H	3255	MSR_M0_PMON_CTRL3	Package	Uncore M-box 0 perfmon counter MSR.
CB8H	3256	MSR_M0_PMON_EVNT_SEL4	Package	Uncore M-box 0 perfmon event select MSR.
CB9H	3257	MSR_M0_PMON_CTRL4	Package	Uncore M-box 0 perfmon counter MSR.
CBAH	3258	MSR_M0_PMON_EVNT_SEL5	Package	Uncore M-box 0 perfmon event select MSR.
CBBH	3259	MSR_M0_PMON_CTRL5	Package	Uncore M-box 0 perfmon counter MSR.
CC0H	3264	MSR_S1_PMON_BOX_CTRL	Package	Uncore S-box 1 perfmon local box control MSR.
CC1H	3265	MSR_S1_PMON_BOX_STATUS	Package	Uncore S-box 1 perfmon local box status MSR.
CC2H	3266	MSR_S1_PMON_BOX_OVF_CTRL	Package	Uncore S-box 1 perfmon local box overflow control MSR.
CD0H	3280	MSR_S1_PMON_EVNT_SEL0	Package	Uncore S-box 1 perfmon event select MSR.
CD1H	3281	MSR_S1_PMON_CTRL0	Package	Uncore S-box 1 perfmon counter MSR.
CD2H	3282	MSR_S1_PMON_EVNT_SEL1	Package	Uncore S-box 1 perfmon event select MSR.
CD3H	3283	MSR_S1_PMON_CTRL1	Package	Uncore S-box 1 perfmon counter MSR.
CD4H	3284	MSR_S1_PMON_EVNT_SEL2	Package	Uncore S-box 1 perfmon event select MSR.
CD5H	3285	MSR_S1_PMON_CTRL2	Package	Uncore S-box 1 perfmon counter MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CD6H	3286	MSR_S1_PMON_EVNT_SEL3	Package	Uncore S-box 1 perfmon event select MSR.
CD7H	3287	MSR_S1_PMON_CTR3	Package	Uncore S-box 1 perfmon counter MSR.
CE0H	3296	MSR_M1_PMON_BOX_CTRL	Package	Uncore M-box 1 perfmon local box control MSR.
CE1H	3297	MSR_M1_PMON_BOX_STATUS	Package	Uncore M-box 1 perfmon local box status MSR.
CE2H	3298	MSR_M1_PMON_BOX_OVF_CTRL	Package	Uncore M-box 1 perfmon local box overflow control MSR.
CE4H	3300	MSR_M1_PMON_TIMESTAMP	Package	Uncore M-box 1 perfmon time stamp unit select MSR.
CE5H	3301	MSR_M1_PMON_DSP	Package	Uncore M-box 1 perfmon DSP unit select MSR.
CE6H	3302	MSR_M1_PMON_ISS	Package	Uncore M-box 1 perfmon ISS unit select MSR.
CE7H	3303	MSR_M1_PMON_MAP	Package	Uncore M-box 1 perfmon MAP unit select MSR.
CE8H	3304	MSR_M1_PMON_MSC_THR	Package	Uncore M-box 1 perfmon MIC THR select MSR.
CE9H	3305	MSR_M1_PMON_PGT	Package	Uncore M-box 1 perfmon PGT unit select MSR.
CEAH	3306	MSR_M1_PMON_PLD	Package	Uncore M-box 1 perfmon PLD unit select MSR.
CEBH	3307	MSR_M1_PMON_ZDP	Package	Uncore M-box 1 perfmon ZDP unit select MSR.
CF0H	3312	MSR_M1_PMON_EVNT_SELO	Package	Uncore M-box 1 perfmon event select MSR.
CF1H	3313	MSR_M1_PMON_CTR0	Package	Uncore M-box 1 perfmon counter MSR.
CF2H	3314	MSR_M1_PMON_EVNT_SEL1	Package	Uncore M-box 1 perfmon event select MSR.
CF3H	3315	MSR_M1_PMON_CTR1	Package	Uncore M-box 1 perfmon counter MSR.
CF4H	3316	MSR_M1_PMON_EVNT_SEL2	Package	Uncore M-box 1 perfmon event select MSR.
CF5H	3317	MSR_M1_PMON_CTR2	Package	Uncore M-box 1 perfmon counter MSR.
CF6H	3318	MSR_M1_PMON_EVNT_SEL3	Package	Uncore M-box 1 perfmon event select MSR.
CF7H	3319	MSR_M1_PMON_CTR3	Package	Uncore M-box 1 perfmon counter MSR.
CF8H	3320	MSR_M1_PMON_EVNT_SEL4	Package	Uncore M-box 1 perfmon event select MSR.
CF9H	3321	MSR_M1_PMON_CTR4	Package	Uncore M-box 1 perfmon counter MSR.
CFAH	3322	MSR_M1_PMON_EVNT_SEL5	Package	Uncore M-box 1 perfmon event select MSR.
CFBH	3323	MSR_M1_PMON_CTR5	Package	Uncore M-box 1 perfmon counter MSR.
DO0H	3328	MSR_C0_PMON_BOX_CTRL	Package	Uncore C-box 0 perfmon local box control MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D01H	3329	MSR_CO_PMON_BOX_STATUS	Package	Uncore C-box 0 perfmon local box status MSR.
D02H	3330	MSR_CO_PMON_BOX_OVF_CTRL	Package	Uncore C-box 0 perfmon local box overflow control MSR.
D10H	3344	MSR_CO_PMON_EVNT_SELO	Package	Uncore C-box 0 perfmon event select MSR.
D11H	3345	MSR_CO_PMON_CTR0	Package	Uncore C-box 0 perfmon counter MSR.
D12H	3346	MSR_CO_PMON_EVNT_SEL1	Package	Uncore C-box 0 perfmon event select MSR.
D13H	3347	MSR_CO_PMON_CTR1	Package	Uncore C-box 0 perfmon counter MSR.
D14H	3348	MSR_CO_PMON_EVNT_SEL2	Package	Uncore C-box 0 perfmon event select MSR.
D15H	3349	MSR_CO_PMON_CTR2	Package	Uncore C-box 0 perfmon counter MSR.
D16H	3350	MSR_CO_PMON_EVNT_SEL3	Package	Uncore C-box 0 perfmon event select MSR.
D17H	3351	MSR_CO_PMON_CTR3	Package	Uncore C-box 0 perfmon counter MSR.
D18H	3352	MSR_CO_PMON_EVNT_SEL4	Package	Uncore C-box 0 perfmon event select MSR.
D19H	3353	MSR_CO_PMON_CTR4	Package	Uncore C-box 0 perfmon counter MSR.
D1AH	3354	MSR_CO_PMON_EVNT_SEL5	Package	Uncore C-box 0 perfmon event select MSR.
D1BH	3355	MSR_CO_PMON_CTR5	Package	Uncore C-box 0 perfmon counter MSR.
D20H	3360	MSR_C4_PMON_BOX_CTRL	Package	Uncore C-box 4 perfmon local box control MSR.
D21H	3361	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-box 4 perfmon local box status MSR.
D22H	3362	MSR_C4_PMON_BOX_OVF_CTRL	Package	Uncore C-box 4 perfmon local box overflow control MSR.
D30H	3376	MSR_C4_PMON_EVNT_SELO	Package	Uncore C-box 4 perfmon event select MSR.
D31H	3377	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter MSR.
D32H	3378	MSR_C4_PMON_EVNT_SEL1	Package	Uncore C-box 4 perfmon event select MSR.
D33H	3379	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter MSR.
D34H	3380	MSR_C4_PMON_EVNT_SEL2	Package	Uncore C-box 4 perfmon event select MSR.
D35H	3381	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter MSR.
D36H	3382	MSR_C4_PMON_EVNT_SEL3	Package	Uncore C-box 4 perfmon event select MSR.
D37H	3383	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D38H	3384	MSR_C4_PMON_EVNT_SEL4	Package	Uncore C-box 4 perfmon event select MSR.
D39H	3385	MSR_C4_PMON_CTRL4	Package	Uncore C-box 4 perfmon counter MSR.
D3AH	3386	MSR_C4_PMON_EVNT_SEL5	Package	Uncore C-box 4 perfmon event select MSR.
D3BH	3387	MSR_C4_PMON_CTRL5	Package	Uncore C-box 4 perfmon counter MSR.
D40H	3392	MSR_C2_PMON_BOX_CTRL	Package	Uncore C-box 2 perfmon local box control MSR.
D41H	3393	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-box 2 perfmon local box status MSR.
D42H	3394	MSR_C2_PMON_BOX_OVF_CTRL	Package	Uncore C-box 2 perfmon local box overflow control MSR.
D50H	3408	MSR_C2_PMON_EVNT_SELO	Package	Uncore C-box 2 perfmon event select MSR.
D51H	3409	MSR_C2_PMON_CTRL0	Package	Uncore C-box 2 perfmon counter MSR.
D52H	3410	MSR_C2_PMON_EVNT_SEL1	Package	Uncore C-box 2 perfmon event select MSR.
D53H	3411	MSR_C2_PMON_CTRL1	Package	Uncore C-box 2 perfmon counter MSR.
D54H	3412	MSR_C2_PMON_EVNT_SEL2	Package	Uncore C-box 2 perfmon event select MSR.
D55H	3413	MSR_C2_PMON_CTRL2	Package	Uncore C-box 2 perfmon counter MSR.
D56H	3414	MSR_C2_PMON_EVNT_SEL3	Package	Uncore C-box 2 perfmon event select MSR.
D57H	3415	MSR_C2_PMON_CTRL3	Package	Uncore C-box 2 perfmon counter MSR.
D58H	3416	MSR_C2_PMON_EVNT_SEL4	Package	Uncore C-box 2 perfmon event select MSR.
D59H	3417	MSR_C2_PMON_CTRL4	Package	Uncore C-box 2 perfmon counter MSR.
D5AH	3418	MSR_C2_PMON_EVNT_SEL5	Package	Uncore C-box 2 perfmon event select MSR.
D5BH	3419	MSR_C2_PMON_CTRL5	Package	Uncore C-box 2 perfmon counter MSR.
D60H	3424	MSR_C6_PMON_BOX_CTRL	Package	Uncore C-box 6 perfmon local box control MSR.
D61H	3425	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-box 6 perfmon local box status MSR.
D62H	3426	MSR_C6_PMON_BOX_OVF_CTRL	Package	Uncore C-box 6 perfmon local box overflow control MSR.
D70H	3440	MSR_C6_PMON_EVNT_SELO	Package	Uncore C-box 6 perfmon event select MSR.
D71H	3441	MSR_C6_PMON_CTRL0	Package	Uncore C-box 6 perfmon counter MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D72H	3442	MSR_C6_PMON_EVNT_SEL1	Package	Uncore C-box 6 perfmon event select MSR.
D73H	3443	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter MSR.
D74H	3444	MSR_C6_PMON_EVNT_SEL2	Package	Uncore C-box 6 perfmon event select MSR.
D75H	3445	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter MSR.
D76H	3446	MSR_C6_PMON_EVNT_SEL3	Package	Uncore C-box 6 perfmon event select MSR.
D77H	3447	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter MSR.
D78H	3448	MSR_C6_PMON_EVNT_SEL4	Package	Uncore C-box 6 perfmon event select MSR.
D79H	3449	MSR_C6_PMON_CTR4	Package	Uncore C-box 6 perfmon counter MSR.
D7AH	3450	MSR_C6_PMON_EVNT_SEL5	Package	Uncore C-box 6 perfmon event select MSR.
D7BH	3451	MSR_C6_PMON_CTR5	Package	Uncore C-box 6 perfmon counter MSR.
D80H	3456	MSR_C1_PMON_BOX_CTRL	Package	Uncore C-box 1 perfmon local box control MSR.
D81H	3457	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-box 1 perfmon local box status MSR.
D82H	3458	MSR_C1_PMON_BOX_OVF_CTRL	Package	Uncore C-box 1 perfmon local box overflow control MSR.
D90H	3472	MSR_C1_PMON_EVNT_SELO	Package	Uncore C-box 1 perfmon event select MSR.
D91H	3473	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter MSR.
D92H	3474	MSR_C1_PMON_EVNT_SEL1	Package	Uncore C-box 1 perfmon event select MSR.
D93H	3475	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter MSR.
D94H	3476	MSR_C1_PMON_EVNT_SEL2	Package	Uncore C-box 1 perfmon event select MSR.
D95H	3477	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter MSR.
D96H	3478	MSR_C1_PMON_EVNT_SEL3	Package	Uncore C-box 1 perfmon event select MSR.
D97H	3479	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter MSR.
D98H	3480	MSR_C1_PMON_EVNT_SEL4	Package	Uncore C-box 1 perfmon event select MSR.
D99H	3481	MSR_C1_PMON_CTR4	Package	Uncore C-box 1 perfmon counter MSR.
D9AH	3482	MSR_C1_PMON_EVNT_SEL5	Package	Uncore C-box 1 perfmon event select MSR.
D9BH	3483	MSR_C1_PMON_CTR5	Package	Uncore C-box 1 perfmon counter MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DA0H	3488	MSR_C5_PMON_BOX_CTRL	Package	Uncore C-box 5 perfmon local box control MSR.
DA1H	3489	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon local box status MSR.
DA2H	3490	MSR_C5_PMON_BOX_OVF_CTRL	Package	Uncore C-box 5 perfmon local box overflow control MSR.
DB0H	3504	MSR_C5_PMON_EVNT_SELO	Package	Uncore C-box 5 perfmon event select MSR.
DB1H	3505	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter MSR.
DB2H	3506	MSR_C5_PMON_EVNT_SEL1	Package	Uncore C-box 5 perfmon event select MSR.
DB3H	3507	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter MSR.
DB4H	3508	MSR_C5_PMON_EVNT_SEL2	Package	Uncore C-box 5 perfmon event select MSR.
DB5H	3509	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter MSR.
DB6H	3510	MSR_C5_PMON_EVNT_SEL3	Package	Uncore C-box 5 perfmon event select MSR.
DB7H	3511	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter MSR.
DB8H	3512	MSR_C5_PMON_EVNT_SEL4	Package	Uncore C-box 5 perfmon event select MSR.
DB9H	3513	MSR_C5_PMON_CTR4	Package	Uncore C-box 5 perfmon counter MSR.
DBAH	3514	MSR_C5_PMON_EVNT_SEL5	Package	Uncore C-box 5 perfmon event select MSR.
DBBH	3515	MSR_C5_PMON_CTR5	Package	Uncore C-box 5 perfmon counter MSR.
DC0H	3520	MSR_C3_PMON_BOX_CTRL	Package	Uncore C-box 3 perfmon local box control MSR.
DC1H	3521	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-box 3 perfmon local box status MSR.
DC2H	3522	MSR_C3_PMON_BOX_OVF_CTRL	Package	Uncore C-box 3 perfmon local box overflow control MSR.
DD0H	3536	MSR_C3_PMON_EVNT_SELO	Package	Uncore C-box 3 perfmon event select MSR.
DD1H	3537	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter MSR.
DD2H	3538	MSR_C3_PMON_EVNT_SEL1	Package	Uncore C-box 3 perfmon event select MSR.
DD3H	3539	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter MSR.
DD4H	3540	MSR_C3_PMON_EVNT_SEL2	Package	Uncore C-box 3 perfmon event select MSR.
DD5H	3541	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter MSR.



Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DD6H	3542	MSR_C3_PMON_EVNT_SEL3	Package	Uncore C-box 3 perfmon event select MSR.
DD7H	3543	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter MSR.
DD8H	3544	MSR_C3_PMON_EVNT_SEL4	Package	Uncore C-box 3 perfmon event select MSR.
DD9H	3545	MSR_C3_PMON_CTR4	Package	Uncore C-box 3 perfmon counter MSR.
DDAH	3546	MSR_C3_PMON_EVNT_SEL5	Package	Uncore C-box 3 perfmon event select MSR.
DDBH	3547	MSR_C3_PMON_CTR5	Package	Uncore C-box 3 perfmon counter MSR.
DE0H	3552	MSR_C7_PMON_BOX_CTRL	Package	Uncore C-box 7 perfmon local box control MSR.
DE1H	3553	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-box 7 perfmon local box status MSR.
DE2H	3554	MSR_C7_PMON_BOX_OVF_CTRL	Package	Uncore C-box 7 perfmon local box overflow control MSR.
DF0H	3568	MSR_C7_PMON_EVNT_SELO	Package	Uncore C-box 7 perfmon event select MSR.
DF1H	3569	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter MSR.
DF2H	3570	MSR_C7_PMON_EVNT_SEL1	Package	Uncore C-box 7 perfmon event select MSR.
DF3H	3571	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter MSR.
DF4H	3572	MSR_C7_PMON_EVNT_SEL2	Package	Uncore C-box 7 perfmon event select MSR.
DF5H	3573	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter MSR.
DF6H	3574	MSR_C7_PMON_EVNT_SEL3	Package	Uncore C-box 7 perfmon event select MSR.
DF7H	3575	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter MSR.
DF8H	3576	MSR_C7_PMON_EVNT_SEL4	Package	Uncore C-box 7 perfmon event select MSR.
DF9H	3577	MSR_C7_PMON_CTR4	Package	Uncore C-box 7 perfmon counter MSR.
DFAH	3578	MSR_C7_PMON_EVNT_SEL5	Package	Uncore C-box 7 perfmon event select MSR.
DFBH	3579	MSR_C7_PMON_CTR5	Package	Uncore C-box 7 perfmon counter MSR.
E00H	3584	MSR_R0_PMON_BOX_CTRL	Package	Uncore R-box 0 perfmon local box control MSR.
E01H	3585	MSR_R0_PMON_BOX_STATUS	Package	Uncore R-box 0 perfmon local box status MSR.
E02H	3586	MSR_R0_PMON_BOX_OVF_CTRL	Package	Uncore R-box 0 perfmon local box overflow control MSR.
E04H	3588	MSR_R0_PMON_IPERFO_PO	Package	Uncore R-box 0 perfmon IPERFO unit Port 0 select MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E05H	3589	MSR_R0_PMON_IPERF0_P1	Package	Uncore R-box 0 perfmon IPERF0 unit Port 1 select MSR.
E06H	3590	MSR_R0_PMON_IPERF0_P2	Package	Uncore R-box 0 perfmon IPERF0 unit Port 2 select MSR.
E07H	3591	MSR_R0_PMON_IPERF0_P3	Package	Uncore R-box 0 perfmon IPERF0 unit Port 3 select MSR.
E08H	3592	MSR_R0_PMON_IPERF0_P4	Package	Uncore R-box 0 perfmon IPERF0 unit Port 4 select MSR.
E09H	3593	MSR_R0_PMON_IPERF0_P5	Package	Uncore R-box 0 perfmon IPERF0 unit Port 5 select MSR.
E0AH	3594	MSR_R0_PMON_IPERF0_P6	Package	Uncore R-box 0 perfmon IPERF0 unit Port 6 select MSR.
E0BH	3595	MSR_R0_PMON_IPERF0_P7	Package	Uncore R-box 0 perfmon IPERF0 unit Port 7 select MSR.
E0CH	3596	MSR_R0_PMON_QLX_P0	Package	Uncore R-box 0 perfmon QLX unit Port 0 select MSR.
E0DH	3597	MSR_R0_PMON_QLX_P1	Package	Uncore R-box 0 perfmon QLX unit Port 1 select MSR.
E0EH	3598	MSR_R0_PMON_QLX_P2	Package	Uncore R-box 0 perfmon QLX unit Port 2 select MSR.
E0FH	3599	MSR_R0_PMON_QLX_P3	Package	Uncore R-box 0 perfmon QLX unit Port 3 select MSR.
E10H	3600	MSR_R0_PMON_EVNT_SELO	Package	Uncore R-box 0 perfmon event select MSR.
E11H	3601	MSR_R0_PMON_CTR0	Package	Uncore R-box 0 perfmon counter MSR.
E12H	3602	MSR_R0_PMON_EVNT_SEL1	Package	Uncore R-box 0 perfmon event select MSR.
E13H	3603	MSR_R0_PMON_CTR1	Package	Uncore R-box 0 perfmon counter MSR.
E14H	3604	MSR_R0_PMON_EVNT_SEL2	Package	Uncore R-box 0 perfmon event select MSR.
E15H	3605	MSR_R0_PMON_CTR2	Package	Uncore R-box 0 perfmon counter MSR.
E16H	3606	MSR_R0_PMON_EVNT_SEL3	Package	Uncore R-box 0 perfmon event select MSR.
E17H	3607	MSR_R0_PMON_CTR3	Package	Uncore R-box 0 perfmon counter MSR.
E18H	3608	MSR_R0_PMON_EVNT_SEL4	Package	Uncore R-box 0 perfmon event select MSR.
E19H	3609	MSR_R0_PMON_CTR4	Package	Uncore R-box 0 perfmon counter MSR.
E1AH	3610	MSR_R0_PMON_EVNT_SEL5	Package	Uncore R-box 0 perfmon event select MSR.
E1BH	3611	MSR_R0_PMON_CTR5	Package	Uncore R-box 0 perfmon counter MSR.
E1CH	3612	MSR_R0_PMON_EVNT_SEL6	Package	Uncore R-box 0 perfmon event select MSR.
E1DH	3613	MSR_R0_PMON_CTR6	Package	Uncore R-box 0 perfmon counter MSR.
E1EH	3614	MSR_R0_PMON_EVNT_SEL7	Package	Uncore R-box 0 perfmon event select MSR.
E1FH	3615	MSR_R0_PMON_CTR7	Package	Uncore R-box 0 perfmon counter MSR.
E20H	3616	MSR_R1_PMON_BOX_CTRL	Package	Uncore R-box 1 perfmon local box control MSR.
E21H	3617	MSR_R1_PMON_BOX_STATUS	Package	Uncore R-box 1 perfmon local box status MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E22H	3618	MSR_R1_PMON_BOX_OVF_CTRL	Package	Uncore R-box 1 perfmon local box overflow control MSR.
E24H	3620	MSR_R1_PMON_IPERF1_P8	Package	Uncore R-box 1 perfmon IPERF1 unit Port 8 select MSR.
E25H	3621	MSR_R1_PMON_IPERF1_P9	Package	Uncore R-box 1 perfmon IPERF1 unit Port 9 select MSR.
E26H	3622	MSR_R1_PMON_IPERF1_P10	Package	Uncore R-box 1 perfmon IPERF1 unit Port 10 select MSR.
E27H	3623	MSR_R1_PMON_IPERF1_P11	Package	Uncore R-box 1 perfmon IPERF1 unit Port 11 select MSR.
E28H	3624	MSR_R1_PMON_IPERF1_P12	Package	Uncore R-box 1 perfmon IPERF1 unit Port 12 select MSR.
E29H	3625	MSR_R1_PMON_IPERF1_P13	Package	Uncore R-box 1 perfmon IPERF1 unit Port 13 select MSR.
E2AH	3626	MSR_R1_PMON_IPERF1_P14	Package	Uncore R-box 1 perfmon IPERF1 unit Port 14 select MSR.
E2BH	3627	MSR_R1_PMON_IPERF1_P15	Package	Uncore R-box 1 perfmon IPERF1 unit Port 15 select MSR.
E2CH	3628	MSR_R1_PMON_QLX_P4	Package	Uncore R-box 1 perfmon QLX unit Port 4 select MSR.
E2DH	3629	MSR_R1_PMON_QLX_P5	Package	Uncore R-box 1 perfmon QLX unit Port 5 select MSR.
E2EH	3630	MSR_R1_PMON_QLX_P6	Package	Uncore R-box 1 perfmon QLX unit Port 6 select MSR.
E2FH	3631	MSR_R1_PMON_QLX_P7	Package	Uncore R-box 1 perfmon QLX unit Port 7 select MSR.
E30H	3632	MSR_R1_PMON_EVNT_SEL8	Package	Uncore R-box 1 perfmon event select MSR.
E31H	3633	MSR_R1_PMON_CTR8	Package	Uncore R-box 1 perfmon counter MSR.
E32H	3634	MSR_R1_PMON_EVNT_SEL9	Package	Uncore R-box 1 perfmon event select MSR.
E33H	3635	MSR_R1_PMON_CTR9	Package	Uncore R-box 1 perfmon counter MSR.
E34H	3636	MSR_R1_PMON_EVNT_SEL10	Package	Uncore R-box 1 perfmon event select MSR.
E35H	3637	MSR_R1_PMON_CTR10	Package	Uncore R-box 1 perfmon counter MSR.
E36H	3638	MSR_R1_PMON_EVNT_SEL11	Package	Uncore R-box 1 perfmon event select MSR.
E37H	3639	MSR_R1_PMON_CTR11	Package	Uncore R-box 1 perfmon counter MSR.
E38H	3640	MSR_R1_PMON_EVNT_SEL12	Package	Uncore R-box 1 perfmon event select MSR.
E39H	3641	MSR_R1_PMON_CTR12	Package	Uncore R-box 1 perfmon counter MSR.
E3AH	3642	MSR_R1_PMON_EVNT_SEL13	Package	Uncore R-box 1 perfmon event select MSR.
E3BH	3643	MSR_R1_PMON_CTR13	Package	Uncore R-box 1 perfmon counter MSR.
E3CH	3644	MSR_R1_PMON_EVNT_SEL14	Package	Uncore R-box 1 perfmon event select MSR.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E3DH	3645	MSR_R1_PMON_CTR14	Package	Uncore R-box 1 perfmon counter MSR.
E3EH	3646	MSR_R1_PMON_EVNT_SEL15	Package	Uncore R-box 1 perfmon event select MSR.
E3FH	3647	MSR_R1_PMON_CTR15	Package	Uncore R-box 1 perfmon counter MSR.
E45H	3653	MSR_B0_PMON_MATCH	Package	Uncore B-box 0 perfmon local box match MSR.
E46H	3654	MSR_B0_PMON_MASK	Package	Uncore B-box 0 perfmon local box mask MSR.
E49H	3657	MSR_S0_PMON_MATCH	Package	Uncore S-box 0 perfmon local box match MSR.
E4AH	3658	MSR_S0_PMON_MASK	Package	Uncore S-box 0 perfmon local box mask MSR.
E4DH	3661	MSR_B1_PMON_MATCH	Package	Uncore B-box 1 perfmon local box match MSR.
E4EH	3662	MSR_B1_PMON_MASK	Package	Uncore B-box 1 perfmon local box mask MSR.
E54H	3668	MSR_M0_PMON_MM_CONFIG	Package	Uncore M-box 0 perfmon local box address match/mask config MSR.
E55H	3669	MSR_M0_PMON_ADDR_MATCH	Package	Uncore M-box 0 perfmon local box address match MSR.
E56H	3670	MSR_M0_PMON_ADDR_MASK	Package	Uncore M-box 0 perfmon local box address mask MSR.
E59H	3673	MSR_S1_PMON_MATCH	Package	Uncore S-box 1 perfmon local box match MSR.
E5AH	3674	MSR_S1_PMON_MASK	Package	Uncore S-box 1 perfmon local box mask MSR.
E5CH	3676	MSR_M1_PMON_MM_CONFIG	Package	Uncore M-box 1 perfmon local box address match/mask config MSR.
E5DH	3677	MSR_M1_PMON_ADDR_MATCH	Package	Uncore M-box 1 perfmon local box address match MSR.
E5EH	3678	MSR_M1_PMON_ADDR_MASK	Package	Uncore M-box 1 perfmon local box address mask MSR.
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 18.3.1.2.2, "Uncore Performance Event Configuration Facility."

## 2.8 MSRS IN THE INTEL® XEON® PROCESSOR 5600 SERIES (BASED ON INTEL® MICROARCHITECTURE CODE NAME WESTMERE)

Intel® Xeon® Processor 5600 Series (based on Intel® microarchitecture code name Westmere) supports the MSR interfaces listed in Table 2-14, Table 2-15, plus additional MSR listed in Table 2-17. These MSRs apply to Intel Core i7, i5 and i3 processor family with CPUID signature DisplayFamily\_DisplayModel of 06\_25H and 06\_2CH, see Table 2-1.

**Table 2-17. Additional MSRs Supported by Intel Processors  
(Based on Intel® Microarchitecture Code Name Westmere)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
13CH	52	MSR_FEATURE_CONFIG	Core	<b>AES Configuration (RW-L)</b> Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		<b>AES Configuration (RW-L)</b> Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
1A7H	423	MSR_OFFCORE_RSP_1	Thread	<b>Offcore Response Event Select Register (R/W)</b>
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> <b>RO</b> if MSR_PLATFORM_INFO.[28] = 0, <b>RW</b> if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		39:32	Package	<b>Maximum Ratio Limit for 5C</b> Maximum turbo ratio limit of 5 core active.
		47:40	Package	<b>Maximum Ratio Limit for 6C</b> Maximum turbo ratio limit of 6 core active.
		63:48		Reserved.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.

## 2.9 MSRS IN THE INTEL® XEON® PROCESSOR E7 FAMILY (BASED ON INTEL® MICROARCHITECTURE CODE NAME WESTMERE)

Intel® Xeon® Processor E7 Family (based on Intel® microarchitecture code name Westmere) supports the MSR interfaces listed in Table 2-14 (except MSR address 1ADH), Table 2-15, plus additional MSR listed in Table 2-18. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_2FH.

**Table 2-18. Additional MSRs Supported by Intel® Xeon® Processor E7 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
13CH	52	MSR_FEATURE_CONFIG	Core	<b>AES Configuration (RW-L)</b> Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		<b>AES Configuration (RW-L)</b> Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
1A7H	423	MSR_OFFCORE_RSP_1	Thread	<b>Offcore Response Event Select Register (R/W)</b>
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Reserved</b> Attempt to read/write will cause #UD.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.
F40H	3904	MSR_C8_PMON_BOX_CTRL	Package	Uncore C-box 8 perfmon local box control MSR.
F41H	3905	MSR_C8_PMON_BOX_STATUS	Package	Uncore C-box 8 perfmon local box status MSR.
F42H	3906	MSR_C8_PMON_BOX_OVF_CTRL	Package	Uncore C-box 8 perfmon local box overflow control MSR.
F50H	3920	MSR_C8_PMON_EVNT_SELO	Package	Uncore C-box 8 perfmon event select MSR.
F51H	3921	MSR_C8_PMON_CTR0	Package	Uncore C-box 8 perfmon counter MSR.
F52H	3922	MSR_C8_PMON_EVNT_SEL1	Package	Uncore C-box 8 perfmon event select MSR.
F53H	3923	MSR_C8_PMON_CTR1	Package	Uncore C-box 8 perfmon counter MSR.
F54H	3924	MSR_C8_PMON_EVNT_SEL2	Package	Uncore C-box 8 perfmon event select MSR.
F55H	3925	MSR_C8_PMON_CTR2	Package	Uncore C-box 8 perfmon counter MSR.
F56H	3926	MSR_C8_PMON_EVNT_SEL3	Package	Uncore C-box 8 perfmon event select MSR.
F57H	3927	MSR_C8_PMON_CTR3	Package	Uncore C-box 8 perfmon counter MSR.
F58H	3928	MSR_C8_PMON_EVNT_SEL4	Package	Uncore C-box 8 perfmon event select MSR.
F59H	3929	MSR_C8_PMON_CTR4	Package	Uncore C-box 8 perfmon counter MSR.
F5AH	3930	MSR_C8_PMON_EVNT_SEL5	Package	Uncore C-box 8 perfmon event select MSR.
F5BH	3931	MSR_C8_PMON_CTR5	Package	Uncore C-box 8 perfmon counter MSR.

**Table 2-18. Additional MSRs Supported by Intel® Xeon® Processor E7 Family (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
FC0H	4032	MSR_C9_PMON_BOX_CTRL	Package	Uncore C-box 9 perfmon local box control MSR.
FC1H	4033	MSR_C9_PMON_BOX_STATUS	Package	Uncore C-box 9 perfmon local box status MSR.
FC2H	4034	MSR_C9_PMON_BOX_OVF_CTRL	Package	Uncore C-box 9 perfmon local box overflow control MSR.
FD0H	4048	MSR_C9_PMON_EVNT_SEL0	Package	Uncore C-box 9 perfmon event select MSR.
FD1H	4049	MSR_C9_PMON_CTR0	Package	Uncore C-box 9 perfmon counter MSR.
FD2H	4050	MSR_C9_PMON_EVNT_SEL1	Package	Uncore C-box 9 perfmon event select MSR.
FD3H	4051	MSR_C9_PMON_CTR1	Package	Uncore C-box 9 perfmon counter MSR.
FD4H	4052	MSR_C9_PMON_EVNT_SEL2	Package	Uncore C-box 9 perfmon event select MSR.
FD5H	4053	MSR_C9_PMON_CTR2	Package	Uncore C-box 9 perfmon counter MSR.
FD6H	4054	MSR_C9_PMON_EVNT_SEL3	Package	Uncore C-box 9 perfmon event select MSR.
FD7H	4055	MSR_C9_PMON_CTR3	Package	Uncore C-box 9 perfmon counter MSR.
FD8H	4056	MSR_C9_PMON_EVNT_SEL4	Package	Uncore C-box 9 perfmon event select MSR.
FD9H	4057	MSR_C9_PMON_CTR4	Package	Uncore C-box 9 perfmon counter MSR.
FDAH	4058	MSR_C9_PMON_EVNT_SEL5	Package	Uncore C-box 9 perfmon event select MSR.
FDBH	4059	MSR_C9_PMON_CTR5	Package	Uncore C-box 9 perfmon counter MSR.

## 2.10 MSRS IN INTEL® PROCESSOR FAMILY BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Table 2-19 lists model-specific registers (MSRs) that are common to Intel® processor family based on Intel micro-architecture code name Sandy Bridge. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_2AH, 06\_2DH, see Table 2-1. Additional MSRs specific to 06\_2AH are listed in Table 2-20.

**Table 2-19. 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 2.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 2.22, "MSRs in Pentium Processors."

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	<b>Platform ID (R)</b> See Table 2-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	<b>SMI Counter (R/O)</b>
		31:0		<b>SMI Count (R/O)</b> Count SMIs.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	<b>Control Features in Intel 64 Processor (R/W)</b> See Table 2-2.
		0		<b>Lock (R/WL)</b>
		1		<b>Enable VMX inside SMX operation (R/WL)</b>
		2		<b>Enable VMX outside SMX operation (R/WL)</b>
		14:8		<b>SENTER local functions enables (R/WL)</b>
		15		<b>SENTER global functions enable (R/WL)</b>
79H	121	IA32_BIOS_UPDT_TRIG	Core	<b>BIOS Update Trigger Register (w)</b> See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	<b>BIOS Update Signature ID (RO)</b> See Table 2-2.
C1H	193	IA32_PMC0	Thread	<b>Performance Counter Register</b> See Table 2-2.
C2H	194	IA32_PMC1	Thread	<b>Performance Counter Register</b> See Table 2-2.
C3H	195	IA32_PMC2	Thread	<b>Performance Counter Register</b> See Table 2-2.
C4H	196	IA32_PMC3	Thread	<b>Performance Counter Register</b> See Table 2-2.
C5H	197	IA32_PMC4	Core	<b>Performance Counter Register (if core not shared by threads)</b>
C6H	198	IA32_PMC5	Core	<b>Performance Counter Register (if core not shared by threads)</b>
C7H	199	IA32_PMC6	Core	<b>Performance Counter Register (if core not shared by threads)</b>
C8H	200	IA32_PMC7	Core	<b>Performance Counter Register (if core not shared by threads)</b>
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.



**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> 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	<b>Programmable TDP Limit for Turbo Mode (R/O)</b> 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	<b>Maximum Efficiency Ratio (R/O)</b> 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	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See <a href="http://biosbits.org">http://biosbits.org</a> .
		2:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		<b>I/O MWAIT Redirection Enable (R/W)</b> 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		<b>CFG Lock (R/WO)</b> When set, lock bits 15:0 of this register until next reset.

**Table 2-19. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		24:16		Reserved.
		25		<b>C3 state auto demotion enable (R/W)</b> When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		<b>C1 state auto demotion enable (R/W)</b> When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		<b>Enable C3 undemotion (R/W)</b> When set, enables undemotion from demoted C3.
		28		<b>Enable C1 undemotion (R/W)</b> When set, enables undemotion from demoted C1.
		63:29		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	<b>Power Management IO Redirection in C-state (R/W)</b> See <a href="http://biosbits.org">http://biosbits.org</a> .
		15:0		<b>LVL_2 Base Address (R/W)</b> 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		<b>C-state Range (R/W)</b> Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	<b>Maximum Performance Frequency Clock Count (RW)</b> See Table 2-2.
E8H	232	IA32_APERF	Thread	<b>Actual Performance Frequency Clock Count (RW)</b> See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 2-2.
13CH	52	MSR_FEATURE_CONFIG	Core	<b>AES Configuration (RW-L)</b> Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		1:0		<b>AES Configuration (RW-L)</b> Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status
		0		<b>RIPV</b> 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		<b>EIPV</b> 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		<b>MCIP</b> 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_PERFVTSEL0	Thread	See Table 2-2.
187H	391	IA32_PERFVTSEL1	Thread	See Table 2-2.
188H	392	IA32_PERFVTSEL2	Thread	See Table 2-2.
189H	393	IA32_PERFVTSEL3	Thread	See Table 2-2.
18AH	394	IA32_PERFVTSEL4	Core	See Table 2-2; If CPUID.0AH:EAX[15:8] = 8
18BH	395	IA32_PERFVTSEL5	Core	See Table 2-2; If CPUID.0AH:EAX[15:8] = 8
18CH	396	IA32_PERFVTSEL6	Core	See Table 2-2; If CPUID.0AH:EAX[15:8] = 8

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
18DH	397	IA32_PERFEVTSEL7	Core	See Table 2-2; If CPUID.0AH:EAX[15:8] = 8
198H	408	IA32_PERF_STATUS	Package	See Table 2-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
198H	408	MSR_PERF_STATUS	Package	Performance Status
		47:32		Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2 <sup>13</sup> ).
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	<b>Clock Modulation (R/W)</b> See Table 2-2 IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		3:0		<b>On demand Clock Modulation Duty Cycle (R/W)</b> In 6.25% increment
		4		<b>On demand Clock Modulation Enable (R/W)</b>
		63:5		Reserved.
19BH	411	IA32_THERM_INTERRUPT	Core	<b>Thermal Interrupt Control (R/W)</b> See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.
		0		<b>Thermal status (RO)</b> See Table 2-2.
		1		<b>Thermal status log (R/WCO)</b> See Table 2-2.
		2		<b>PROTCHOT # or FORCEPR# status (RO)</b> See Table 2-2.
		3		<b>PROTCHOT # or FORCEPR# log (R/WCO)</b> See Table 2-2.
		4		<b>Critical Temperature status (RO)</b> See Table 2-2.
		5		<b>Critical Temperature status log (R/WCO)</b> See Table 2-2.
		6		<b>Thermal threshold #1 status (RO)</b> See Table 2-2.
		7		<b>Thermal threshold #1 log (R/WCO)</b> See Table 2-2.

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		8		<b>Thermal threshold #2 status (RO)</b> See Table 2-2.
		9		<b>Thermal threshold #2 log (R/WCO)</b> See Table 2-2.
		10		<b>Power Limitation status (RO)</b> See Table 2-2.
		11		<b>Power Limitation log (R/WCO)</b> See Table 2-2.
		15:12		Reserved.
		22:16		<b>Digital Readout (RO)</b> See Table 2-2.
		26:23		Reserved.
		30:27		<b>Resolution in degrees Celsius (RO)</b> See Table 2-2.
		31		<b>Reading Valid (RO)</b> See Table 2-2.
		63:32		Reserved.
1A0H	416	IA32_MISC_ENABLE		<b>Enable Misc. Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.
		0	Thread	<b>Fast-Strings Enable</b> See Table 2-2
		6:1		Reserved.
		7	Thread	<b>Performance Monitoring Available (R)</b> See Table 2-2.
		10:8		Reserved.
		11	Thread	<b>Branch Trace Storage Unavailable (RO)</b> See Table 2-2.
		12	Thread	<b>Processor Event Based Sampling Unavailable (RO)</b> See Table 2-2.
		15:13		Reserved.
		16	Package	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> See Table 2-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 2-2.
		21:19		Reserved.
		22	Thread	<b>Limit CPUID Maxval (R/W)</b> See Table 2-2.
		23	Thread	<b>xTPR Message Disable (R/W)</b> See Table 2-2.

**Table 2-19. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		33:24		Reserved.
		34	Thread	<b>XD Bit Disable (R/W)</b> See Table 2-2.
		37:35		Reserved.
		38	Package	<b>Turbo Mode Disable (R/W)</b> 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. <b>Note:</b> 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	Unique	Temperature Target
		15:0		Reserved.
		23:16		<b>Temperature Target (R)</b> The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A4H	420	MSR_MISC_FEATURE_CONTROL		<b>Miscellaneous Feature Control (R/W)</b>
		0	Core	<b>L2 Hardware Prefetcher Disable (R/W)</b> If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1	Core	<b>L2 Adjacent Cache Line Prefetcher Disable (R/W)</b> If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).
		2	Core	<b>DCU Hardware Prefetcher Disable (R/W)</b> If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3	Core	<b>DCU IP Prefetcher Disable (R/W)</b> If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction Pointer of previous loads) to determine whether to prefetch additional lines.
		63:4		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	<b>Offcore Response Event Select Register (R/W)</b>
1A7H	422	MSR_OFFCORE_RSP_1	Thread	<b>Offcore Response Event Select Register (R/W)</b>
1AAH	426	MSR_MISC_PWR_MGMT		Miscellaneous Power Management Control; various model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 2-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 2-2.
1C8H	456	MSR_LBR_SELECT	Thread	<b>Last Branch Record Filtering Select Register (R/W)</b> See Section 17.9.2, "Filtering of Last Branch Records."
		0		<b>CPL_EQ_0</b>
		1		<b>CPL_NEQ_0</b>
		2		<b>JCC</b>
		3		<b>NEAR_REL_CALL</b>
		4		<b>NEAR_IND_CALL</b>
		5		<b>NEAR_RET</b>
		6		<b>NEAR_IND_JMP</b>
		7		<b>NEAR_REL_JMP</b>
		8		<b>FAR_BRANCH</b>
63:9		Reserved.		
1C9H	457	MSR_LASTBRANCH_TOS	Thread	<b>Last Branch Record Stack TOS (R/W)</b> 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	<b>Debug Control (R/W)</b> See Table 2-2.
		0		<b>LBR: Last Branch Record</b>
		1		<b>BTF</b>
		5:2		Reserved.
		6		<b>TR: Branch Trace</b>
		7		<b>BTS: Log Branch Trace Message to BTS buffer</b>
		8		<b>BTINT</b>
		9		<b>BTS_OFF_OS</b>
		10		<b>BTS_OFF_USER</b>
		11		<b>FREEZE_LBR_ON_PMI</b>
		12		<b>FREEZE_PERFMON_ON_PMI</b>
		13		<b>ENABLE_UNCORE_PMI</b>
		14		<b>FREEZE_WHILE_SMM</b>
		63:15		Reserved.

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1DDH	477	MSR_LER_FROM_LIP	Thread	<b>Last Exception Record From Linear IP (R)</b> 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	<b>Last Exception Record To Linear IP (R)</b> This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
1FCH	508	MSR_POWER_CTL	Core	See <a href="http://biosbits.org">http://biosbits.org</a> .
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 2-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 2-2.



**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 2-2.
277H	631	IA32_PAT	Thread	See Table 2-2.
280H	640	IA32_MC0_CTL2	Core	See Table 2-2.
281H	641	IA32_MC1_CTL2	Core	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	Always 0 (CMCI not supported).
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	<b>Default Memory Types (R/W)</b> See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	<b>Fixed-Function Performance Counter Register 0 (R/W)</b> See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	<b>Fixed-Function Performance Counter Register 1 (R/W)</b> See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	<b>Fixed-Function Performance Counter Register 2 (R/W)</b> See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format. See Table 2-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 2-2.
		11:8		PEBS_REC_FORMAT. See Table 2-2.
		12		SMM_FREEZE. See Table 2-2.
		63:13		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	<b>Fixed-Function-Counter Control Register (R/W)</b> See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
		0	Thread	<b>Ovf_PMC0</b>
		1	Thread	<b>Ovf_PMC1</b>
		2	Thread	<b>Ovf_PMC2</b>

**Table 2-19. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		3	Thread	Ovf_PMC3
		4	Core	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)
		5	Core	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)
		6	Core	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)
		7	Core	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)
		31:8		Reserved.
		32	Thread	Ovf_FixedCtr0
		33	Thread	Ovf_FixedCtr1
		34	Thread	Ovf_FixedCtr2
		60:35		Reserved.
		61	Thread	Ovf_Uncore
		62	Thread	Ovf_BufDSSAVE
		63	Thread	CondChgd
		38FH	911	IA32_PERF_GLOBAL_CTRL
0	Thread			Set 1 to enable PMC0 to count
1	Thread			Set 1 to enable PMC1 to count
2	Thread			Set 1 to enable PMC2 to count
3	Thread			Set 1 to enable PMC3 to count
4	Core			Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4)
5	Core			Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5)
6	Core			Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6)
7	Core			Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7)
31:8				Reserved.
32	Thread			Set 1 to enable FixedCtr0 to count
33	Thread			Set 1 to enable FixedCtr1 to count
34	Thread			Set 1 to enable FixedCtr2 to count
63:35				Reserved.
390H	912	IA32_PERF_GLOBAL_OVF_CTRL		See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
		0	Thread	Set 1 to clear Ovf_PMC0
		1	Thread	Set 1 to clear Ovf_PMC1
		2	Thread	Set 1 to clear Ovf_PMC2
		3	Thread	Set 1 to clear Ovf_PMC3
		4	Core	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)
		5	Core	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)
		6	Core	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		7	Core	<b>Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] &gt; 7)</b>
		31:8		Reserved.
		32	Thread	<b>Set 1 to clear Ovf_FixedCtr0</b>
		33	Thread	<b>Set 1 to clear Ovf_FixedCtr1</b>
		34	Thread	<b>Set 1 to clear Ovf_FixedCtr2</b>
		60:35		Reserved.
		61	Thread	<b>Set 1 to clear Ovf_Uncore</b>
		62	Thread	<b>Set 1 to clear Ovf_BufDSSAVE</b>
		63	Thread	<b>Set 1 to clear CondChgd</b>
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		62:36		Reserved.
		63		Enable Precise Store. (R/W)
3F6H	1014	MSR_PEBS_LD_LAT	Thread	see See Section 18.3.1.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
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."

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCI_CTL MSRs."
		0		<b>PCU Hardware Error (R/W)</b> When set, enables signaling of PCU hardware detected errors.
		1		<b>PCU Controller Error (R/W)</b> When set, enables signaling of PCU controller detected errors
		2		<b>PCU Firmware Error (R/W)</b> When set, enables signaling of PCU firmware detected errors
		63:2		Reserved.
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
480H	1152	IA32_VMX_BASIC	Thread	<b>Reporting Register of Basic VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Thread	<b>Capability Reporting Register of Pin-based VM-execution Controls (R/O)</b> See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Thread	<b>Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Thread	<b>Capability Reporting Register of VM-exit Controls (R/O)</b> See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Thread	<b>Capability Reporting Register of VM-entry Controls (R/O)</b> See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	<b>Reporting Register of Miscellaneous VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	<b>Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	<b>Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	<b>Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
489H	1161	IA32_VMX_CR4_FIXED1	Thread	<b>Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)</b> See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	<b>Capability Reporting Register of VMCS Field Enumeration (R/O)</b> See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLD2	Thread	<b>Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Thread	<b>Capability Reporting Register of EPT and VPID (R/O)</b> See Table 2-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLD	Thread	<b>Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)</b> See Table 2-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLD	Thread	<b>Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)</b> See Table 2-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLD	Thread	<b>Capability Reporting Register of VM-exit Flex Controls (R/O)</b> See Table 2-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLD	Thread	<b>Capability Reporting Register of VM-entry Flex Controls (R/O)</b> See Table 2-2
4C1H	1217	IA32_A_PMC0	Thread	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 2-2.
4C3H	1219	IA32_A_PMC2	Thread	See Table 2-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 2-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 2-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 2-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 2-2.
4C8H	1224	IA32_A_PMC7	Core	See Table 2-2.
600H	1536	IA32_DS_AREA	Thread	<b>DS Save Area (R/W)</b> See Table 2-2. See Section 18.6.3.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	<b>Unit Multipliers used in RAPL Interfaces (R/O)</b> See Section 14.9.1, "RAPL Interfaces."
60AH	1546	MSR_PKG_C3_INTERRUPT_RESPONSE_LIMIT	Package	<b>Package C3 Interrupt Response Limit (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		9:0		<b>Interrupt response time limit (R/W)</b> Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		<b>Time Unit (R/W)</b> 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		<b>Valid (R/W)</b> 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.
60BH	1547	MSR_PKGC6_IRTL	Package	<b>Package C6 Interrupt Response Limit (R/W)</b> 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. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		<b>Interrupt response time limit (R/W)</b> Specifies the limit that should be used to decide if the package should be put into a package C6 state.
		12:10		<b>Time Unit (R/W)</b> 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		<b>Valid (R/W)</b> 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.

**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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		<b>Package C2 Residency Counter. (R/O)</b> Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_POWER_LIMIT	Package	<b>PKG RAPL Power Limit Control (R/W)</b> See Section 14.9.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	<b>PKG Energy Status (R/O)</b> See Section 14.9.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	<b>PKG RAPL Parameters (R/W)</b> See Section 14.9.3, "Package RAPL Domain."
638H	1592	MSR_PPO_POWER_LIMIT	Package	<b>PPO RAPL Power Limit Control (R/W)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	<b>Last Branch Record 0 From IP (R/W)</b> One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the <b>source instruction</b> . See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.9.1 and record format in Section 17.4.8.1</li> </ul>
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	<b>Last Branch Record 1 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	<b>Last Branch Record 2 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	<b>Last Branch Record 3 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	<b>Last Branch Record 4 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	<b>Last Branch Record 5 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	<b>Last Branch Record 6 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	<b>Last Branch Record 7 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	<b>Last Branch Record 8 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	<b>Last Branch Record 9 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	<b>Last Branch Record 10 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.



**Table 2-19. MSRs Supported by Intel® Processors  
based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	<b>Last Branch Record 11 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	<b>Last Branch Record 12 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	<b>Last Branch Record 13 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	<b>Last Branch Record 14 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	<b>Last Branch Record 15 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	<b>Last Branch Record 0 To IP (R/W)</b> One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	<b>Last Branch Record 1 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	<b>Last Branch Record 2 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	<b>Last Branch Record 3 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	<b>Last Branch Record 4 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	<b>Last Branch Record 5 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	<b>Last Branch Record 6 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	<b>Last Branch Record 7 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	<b>Last Branch Record 8 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	<b>Last Branch Record 9 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	<b>Last Branch Record 10 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	<b>Last Branch Record 11 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	<b>Last Branch Record 12 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.

**Table 2-19. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	<b>Last Branch Record 13 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	<b>Last Branch Record 14 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	<b>Last Branch Record 15 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6E0H	1760	IA32_TSC_DEADLINE	Thread	See Table 2-2.
802H-83FH		X2APIC MSRs	Thread	See Table 2-2.
C000_0080H		IA32_EFER	Thread	<b>Extended Feature Enables</b> See Table 2-2.
C000_0081H		IA32_STAR	Thread	<b>System Call Target Address (R/W)</b> See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	<b>IA-32e Mode System Call Target Address (R/W)</b> See Table 2-2.
C000_0084H		IA32_FMASK	Thread	<b>System Call Flag Mask (R/W)</b> See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	<b>Map of BASE Address of FS (R/W)</b> See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	<b>Map of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	<b>Swap Target of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	<b>AUXILIARY TSC Signature (R/W)</b> See Table 2-2 and Section 17.17.2, "IA32_TSC_AUX Register and RDTSCP Support."

### 2.10.1 MSRs In 2nd Generation Intel® Core™ Processor Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 2-20 and Table 2-21 list model-specific registers (MSRs) that are specific to the 2nd generation Intel® Core™ processor family (based on Intel microarchitecture code name Sandy Bridge). These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_2AH; see Table 2-1.

**Table 2-20. MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® microarchitecture code name Sandy Bridge)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
60CH	1548	MSR_PKGC7_IRTL	Package	<b>Package C7 Interrupt Response Limit (R/W)</b> 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. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		<b>Interrupt response time limit (R/W)</b> Specifies the limit that should be used to decide if the package should be put into a package C7 state.
		12:10		<b>Time Unit (R/W)</b> 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		<b>Valid (R/W)</b> 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.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>PPO Energy Status (R/O)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."

**Table 2-20. MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® microarchitecture code name Sandy Bridge) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
63AH	1594	MSR_PP0_POLICY	Package	<b>PP0 Balance Policy (R/W)</b> See Section 14.9.4, "PP0/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	<b>PP1 RAPL Power Limit Control (R/W)</b> See Section 14.9.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	<b>PP1 Energy Status (R/O)</b> See Section 14.9.4, "PP0/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	<b>PP1 Balance Policy (R/W)</b> See Section 14.9.4, "PP0/PP1 RAPL Domains."

See Table 2-19, Table 2-20, and Table 2-21 for MSR definitions applicable to processors with CPUID signature 06\_2AH.

Table 2-21 lists the MSRs of uncore PMU for Intel processors with CPUID signature 06\_2AH.

**Table 2-21. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Slice 0 select
		1		Slice 1 select
		2		Slice 2 select
		3		Slice 3 select
		4		Slice 4 select
		18:5		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
63:32	Reserved.			
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU main status
		0		Fixed counter overflowed
		1		An ARB counter overflowed
		2		Reserved
		3		A CBox counter overflowed (on any slice)
		63:4		Reserved.
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore fixed counter control (R/W)
		19:0		Reserved
		20		Enable overflow propagation

Table 2-21. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		21		Reserved
		22		Enable counting
		63:23		Reserved.
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box configuration information (R/O)
		3:0		Report the number of C-Box units with performance counters, including processor cores and processor graphics"
		63:4		Reserved.
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb unit, performance counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSEL0	Package	Uncore Arb unit, counter 0 event select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, counter 1 event select MSR
700H	1792	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, counter 0 event select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, counter 1 event select MSR
702H	1794	MSR_UNC_CBO_0_PERFEVTSEL2	Package	Uncore C-Box 0, counter 2 event select MSR.
703H	1795	MSR_UNC_CBO_0_PERFEVTSEL3	Package	Uncore C-Box 0, counter 3 event select MSR.
705H	1797	MSR_UNC_CBO_0_UNIT_STATUS	Package	Uncore C-Box 0, unit status for counter 0-3
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, performance counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, performance counter 1
708H	1800	MSR_UNC_CBO_0_PERFCTR2	Package	Uncore C-Box 0, performance counter 2.
709H	1801	MSR_UNC_CBO_0_PERFCTR3	Package	Uncore C-Box 0, performance counter 3.
710H	1808	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, counter 0 event select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, counter 1 event select MSR
712H	1810	MSR_UNC_CBO_1_PERFEVTSEL2	Package	Uncore C-Box 1, counter 2 event select MSR.
713H	1811	MSR_UNC_CBO_1_PERFEVTSEL3	Package	Uncore C-Box 1, counter 3 event select MSR.

Table 2-21. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
715H	1813	MSR_UNC_CBO_1_UNIT_STATUS	Package	Uncore C-Box 1, unit status for counter 0-3
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, performance counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, performance counter 1
718H	1816	MSR_UNC_CBO_1_PERFCTR2	Package	Uncore C-Box 1, performance counter 2.
719H	1817	MSR_UNC_CBO_1_PERFCTR3	Package	Uncore C-Box 1, performance counter 3.
720H	1824	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, counter 0 event select MSR
721H	1825	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, counter 1 event select MSR
722H	1826	MSR_UNC_CBO_2_PERFEVTSEL2	Package	Uncore C-Box 2, counter 2 event select MSR.
723H	1827	MSR_UNC_CBO_2_PERFEVTSEL3	Package	Uncore C-Box 2, counter 3 event select MSR.
725H	1829	MSR_UNC_CBO_2_UNIT_STATUS	Package	Uncore C-Box 2, unit status for counter 0-3
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, performance counter 1
728H	1832	MSR_UNC_CBO_3_PERFCTR2	Package	Uncore C-Box 3, performance counter 2.
729H	1833	MSR_UNC_CBO_3_PERFCTR3	Package	Uncore C-Box 3, performance counter 3.
730H	1840	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, counter 0 event select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, counter 1 event select MSR.
732H	1842	MSR_UNC_CBO_3_PERFEVTSEL2	Package	Uncore C-Box 3, counter 2 event select MSR.
733H	1843	MSR_UNC_CBO_3_PERFEVTSEL3	Package	Uncore C-Box 3, counter 3 event select MSR.
735H	1845	MSR_UNC_CBO_3_UNIT_STATUS	Package	Uncore C-Box 3, unit status for counter 0-3
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, performance counter 0.
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, performance counter 1.
738H	1848	MSR_UNC_CBO_3_PERFCTR2	Package	Uncore C-Box 3, performance counter 2.
739H	1849	MSR_UNC_CBO_3_PERFCTR3	Package	Uncore C-Box 3, performance counter 3.
740H	1856	MSR_UNC_CBO_4_PERFEVTSELO	Package	Uncore C-Box 4, counter 0 event select MSR
741H	1857	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, counter 1 event select MSR.
742H	1858	MSR_UNC_CBO_4_PERFEVTSEL2	Package	Uncore C-Box 4, counter 2 event select MSR.

**Table 2-21. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
743H	1859	MSR_UNC_CBO_4_PERFEVTSEL3	Package	Uncore C-Box 4, counter 3 event select MSR.
745H	1861	MSR_UNC_CBO_4_UNIT_STATUS	Package	Uncore C-Box 4, unit status for counter 0-3
746H	1862	MSR_UNC_CBO_4_PERFCTR0	Package	Uncore C-Box 4, performance counter 0.
747H	1863	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, performance counter 1.
748H	1864	MSR_UNC_CBO_4_PERFCTR2	Package	Uncore C-Box 4, performance counter 2.
749H	1865	MSR_UNC_CBO_4_PERFCTR3	Package	Uncore C-Box 4, performance counter 3.

## 2.10.2 MSRs In Intel® Xeon® Processor E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 2-22 lists additional 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, and also supports MSRs listed in Table 2-19 and Table 2-23.

**Table 2-22. Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17FH	383	MSR_ERROR_CONTROL	Package	<b>MC Bank Error Configuration (R/W)</b>
		0		Reserved
		1		<b>MemError Log Enable (R/W)</b> When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		39:32	Package	<b>Maximum Ratio Limit for 5C</b> Maximum turbo ratio limit of 5 core active.

**Table 2-22. Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		47:40	Package	<b>Maximum Ratio Limit for 6C</b> Maximum turbo ratio limit of 6 core active.
		55:48	Package	<b>Maximum Ratio Limit for 7C</b> Maximum turbo ratio limit of 7 core active.
		63:56	Package	<b>Maximum Ratio Limit for 8C</b> Maximum turbo ratio limit of 8 core active.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
39CH	924	MSR_PEBS_NUM_ALT	Package	ENABLE_PEBS_NUM_ALT (Rw)
		0		<b>ENABLE_PEBS_NUM_ALT (RW)</b> Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see Table 19-17
		63:1		Reserved (must be zero).
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	IA32_MC5_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	IA32_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	IA32_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41EH	1054	IA32_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."



**Table 2-22. Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
41FH	1055	IA32_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	IA32_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
422H	1058	IA32_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	IA32_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	IA32_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
426H	1062	IA32_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	IA32_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	IA32_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42AH	1066	IA32_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	IA32_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	IA32_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42EH	1070	IA32_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	IA32_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	IA32_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
432H	1074	IA32_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	IA32_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	IA32_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
436H	1078	IA32_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	IA32_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	IA32_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43AH	1082	IA32_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	IA32_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	IA32_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43EH	1086	IA32_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	IA32_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	IA32_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
442H	1090	IA32_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."

**Table 2-22. Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
443H	1091	IA32_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRs."
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCI_CTL MSRs."
445H	1093	IA32_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
446H	1094	IA32_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRs."
447H	1095	IA32_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRs."
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCI_CTL MSRs."
449H	1097	IA32_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
44AH	1098	IA32_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRs."
44BH	1099	IA32_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRs."
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCI_CTL MSRs."
44DH	1101	IA32_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 16.
44EH	1102	IA32_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCI_ADDR MSRs."
44FH	1103	IA32_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCI_MISC MSRs."
613H	1555	MSR_PKG_PERF_STATUS	Package	<b>Package RAPL Perf Status (R/O)</b>
618H	1560	MSR_DRAM_POWER_LIMIT	Package	<b>DRAM RAPL Power Limit Control (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	<b>DRAM Energy Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	<b>DRAM Performance Throttling Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	<b>DRAM RAPL Parameters (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>PPO Energy Status (R/O)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."

See Table 2-19, Table 2-22, and Table 2-23 for MSR definitions applicable to processors with CPUID signature 06\_2DH.

### 2.10.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 Family

Intel Xeon Processor E5 family is based on the Sandy Bridge microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-23. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_2DH

**Table 2-23. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C08H		MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-box UCLK fixed counter control
C09H		MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-box UCLK fixed counter

Table 2-23. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C10H		MSR_U_PMON_EVTSELO	Package	Uncore U-box perfmon event select for U-box counter 0.
C11H		MSR_U_PMON_EVTSEL1	Package	Uncore U-box perfmon event select for U-box counter 1.
C16H		MSR_U_PMON_CTR0	Package	Uncore U-box perfmon counter 0
C17H		MSR_U_PMON_CTR1	Package	Uncore U-box perfmon counter 1
C24H		MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU perfmon for PCU-box-wide control
C30H		MSR_PCU_PMON_EVTSELO	Package	Uncore PCU perfmon event select for PCU counter 0.
C31H		MSR_PCU_PMON_EVTSEL1	Package	Uncore PCU perfmon event select for PCU counter 1.
C32H		MSR_PCU_PMON_EVTSEL2	Package	Uncore PCU perfmon event select for PCU counter 2.
C33H		MSR_PCU_PMON_EVTSEL3	Package	Uncore PCU perfmon event select for PCU counter 3.
C34H		MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU perfmon box-wide filter.
C36H		MSR_PCU_PMON_CTR0	Package	Uncore PCU perfmon counter 0.
C37H		MSR_PCU_PMON_CTR1	Package	Uncore PCU perfmon counter 1.
C38H		MSR_PCU_PMON_CTR2	Package	Uncore PCU perfmon counter 2.
C39H		MSR_PCU_PMON_CTR3	Package	Uncore PCU perfmon counter 3.
D04H		MSR_C0_PMON_BOX_CTL	Package	Uncore C-box 0 perfmon local box wide control.
D10H		MSR_C0_PMON_EVTSELO	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 0.
D11H		MSR_C0_PMON_EVTSEL1	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 1.
D12H		MSR_C0_PMON_EVTSEL2	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 2.
D13H		MSR_C0_PMON_EVTSEL3	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 3.
D14H		MSR_C0_PMON_BOX_FILTER	Package	Uncore C-box 0 perfmon box wide filter.
D16H		MSR_C0_PMON_CTR0	Package	Uncore C-box 0 perfmon counter 0.
D17H		MSR_C0_PMON_CTR1	Package	Uncore C-box 0 perfmon counter 1.
D18H		MSR_C0_PMON_CTR2	Package	Uncore C-box 0 perfmon counter 2.
D19H		MSR_C0_PMON_CTR3	Package	Uncore C-box 0 perfmon counter 3.
D24H		MSR_C1_PMON_BOX_CTL	Package	Uncore C-box 1 perfmon local box wide control.
D30H		MSR_C1_PMON_EVTSELO	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 0.
D31H		MSR_C1_PMON_EVTSEL1	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 1.
D32H		MSR_C1_PMON_EVTSEL2	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 2.
D33H		MSR_C1_PMON_EVTSEL3	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 3.
D34H		MSR_C1_PMON_BOX_FILTER	Package	Uncore C-box 1 perfmon box wide filter.
D36H		MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter 0.
D37H		MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter 1.
D38H		MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter 2.
D39H		MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter 3.
D44H		MSR_C2_PMON_BOX_CTL	Package	Uncore C-box 2 perfmon local box wide control.
D50H		MSR_C2_PMON_EVTSELO	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 0.
D51H		MSR_C2_PMON_EVTSEL1	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 1.

Table 2-23. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D52H		MSR_C2_PMON_EVTSEL2	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 2.
D53H		MSR_C2_PMON_EVTSEL3	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 3.
D54H		MSR_C2_PMON_BOX_FILTER	Package	Uncore C-box 2 perfmon box wide filter.
D56H		MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter 0.
D57H		MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter 1.
D58H		MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter 2.
D59H		MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter 3.
D64H		MSR_C3_PMON_BOX_CTL	Package	Uncore C-box 3 perfmon local box wide control.
D70H		MSR_C3_PMON_EVTSEL0	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 0.
D71H		MSR_C3_PMON_EVTSEL1	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 1.
D72H		MSR_C3_PMON_EVTSEL2	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 2.
D73H		MSR_C3_PMON_EVTSEL3	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 3.
D74H		MSR_C3_PMON_BOX_FILTER	Package	Uncore C-box 3 perfmon box wide filter.
D76H		MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter 0.
D77H		MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter 1.
D78H		MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter 2.
D79H		MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter 3.
D84H		MSR_C4_PMON_BOX_CTL	Package	Uncore C-box 4 perfmon local box wide control.
D90H		MSR_C4_PMON_EVTSEL0	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 0.
D91H		MSR_C4_PMON_EVTSEL1	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 1.
D92H		MSR_C4_PMON_EVTSEL2	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 2.
D93H		MSR_C4_PMON_EVTSEL3	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 3.
D94H		MSR_C4_PMON_BOX_FILTER	Package	Uncore C-box 4 perfmon box wide filter.
D96H		MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter 0.
D97H		MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter 1.
D98H		MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter 2.
D99H		MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter 3.
DA4H		MSR_C5_PMON_BOX_CTL	Package	Uncore C-box 5 perfmon local box wide control.
DB0H		MSR_C5_PMON_EVTSEL0	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 0.
DB1H		MSR_C5_PMON_EVTSEL1	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 1.
DB2H		MSR_C5_PMON_EVTSEL2	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 2.
DB3H		MSR_C5_PMON_EVTSEL3	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 3.
DB4H		MSR_C5_PMON_BOX_FILTER	Package	Uncore C-box 5 perfmon box wide filter.
DB6H		MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter 0.
DB7H		MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter 1.
DB8H		MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter 2.
DB9H		MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter 3.

**Table 2-23. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DC4H		MSR_C6_PMON_BOX_CTL	Package	Uncore C-box 6 perfmon local box wide control.
DD0H		MSR_C6_PMON_EVTSEL0	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 0.
DD1H		MSR_C6_PMON_EVTSEL1	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 1.
DD2H		MSR_C6_PMON_EVTSEL2	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 2.
DD3H		MSR_C6_PMON_EVTSEL3	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 3.
DD4H		MSR_C6_PMON_BOX_FILTER	Package	Uncore C-box 6 perfmon box wide filter.
DD6H		MSR_C6_PMON_CTR0	Package	Uncore C-box 6 perfmon counter 0.
DD7H		MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter 1.
DD8H		MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter 2.
DD9H		MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter 3.
DE4H		MSR_C7_PMON_BOX_CTL	Package	Uncore C-box 7 perfmon local box wide control.
DF0H		MSR_C7_PMON_EVTSEL0	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 0.
DF1H		MSR_C7_PMON_EVTSEL1	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 1.
DF2H		MSR_C7_PMON_EVTSEL2	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 2.
DF3H		MSR_C7_PMON_EVTSEL3	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 3.
DF4H		MSR_C7_PMON_BOX_FILTER	Package	Uncore C-box 7 perfmon box wide filter.
DF6H		MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter 0.
DF7H		MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter 1.
DF8H		MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter 2.
DF9H		MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter 3.

## 2.11 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY (BASED ON INTEL® MICROARCHITECTURE CODE NAME IVY BRIDGE)

The 3rd generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v2 product family (based on Intel microarchitecture code name Ivy Bridge) support the MSR interfaces listed in Table 2-19, Table 2-20, Table 2-21, and Table 2-24. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_3AH.

**Table 2-24. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.

**Table 2-24. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> 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	<b>Programmable TDP Limit for Turbo Mode (R/O)</b> 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.
		31:30		Reserved.
		32	Package	<b>Low Power Mode Support (LPM) (R/O)</b> When set to 1, indicates that LPM is supported, and when set to 0, indicates LPM is not supported.
		34:33	Package	<b>Number of ConfigTDP Levels (R/O)</b> 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 11: Reserved
		39:35		Reserved.
		47:40	Package	<b>Maximum Efficiency Ratio (R/O)</b> The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		55:48	Package	<b>Minimum Operating Ratio (R/O)</b> Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See <a href="http://biosbits.org">http://biosbits.org</a> .

**Table 2-24. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		<b>Package C-State Limit (R/W)</b> 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		<b>I/O MWAIT Redirection Enable (R/W)</b> 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		<b>CFG Lock (R/WO)</b> When set, lock bits 15:0 of this register until next reset.
		24:16		Reserved.
		25		<b>C3 state auto demotion enable (R/W)</b> When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		<b>C1 state auto demotion enable (R/W)</b> When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		<b>Enable C3 undemotion (R/W)</b> When set, enables undemotion from demoted C3.
		28		<b>Enable C1 undemotion (R/W)</b> When set, enables undemotion from demoted C1.
		63:29		Reserved.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>PPO Energy Status (R/O)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	<b>Base TDP Ratio (R/O)</b>
		7:0		<b>Config_TDP_Base</b> Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved.

**Table 2-24. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O)
		14:0		PKG_TDP_LVL1. Power setting for ConfigTDP Level 1.
		15		Reserved
		23:16		Config_TDP_LVL1_Ratio. ConfigTDP level 1 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL1. Max Power setting allowed for ConfigTDP Level 1.
		47		Reserved
		62:48		PKG_MIN_PWR_LVL1. MIN Power setting allowed for ConfigTDP Level 1.
		63		Reserved.
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O)
		14:0		PKG_TDP_LVL2. Power setting for ConfigTDP Level 2.
		15		Reserved
		23:16		Config_TDP_LVL2_Ratio. ConfigTDP level 2 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL2. Max Power setting allowed for ConfigTDP Level 2.
		47		Reserved
		62:48		PKG_MIN_PWR_LVL2. MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved.
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	<b>ConfigTDP Control (R/W)</b>
		1:0		<b>TDP_LEVEL (RW/L)</b> System BIOS can program this field.
		30:2		Reserved.
		31		<b>Config_TDP_Lock (RW/L)</b> When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	<b>ConfigTDP Control (R/W)</b>
		7:0		<b>MAX_NON_TURBO_RATIO (RW/L)</b> System BIOS can program this field.
		30:8		Reserved.



**Table 2-24. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31		<b>TURBO_ACTIVATION_RATIO_Lock (R/W/L)</b> When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
See Table 2-19, Table 2-20 and Table 2-21 for other MSR definitions applicable to processors with CPUID signature 06_3AH				

### 2.11.1 MSRs In Intel® Xeon® Processor E5 v2 Product Family (Based on Ivy Bridge-E Microarchitecture)

Table 2-25 lists model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 v2 Product Family (based on Ivy Bridge-E microarchitecture). These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_3EH, see Table 2-1. These processors supports the MSR interfaces listed in Table 2-19, and Table 2-25.

**Table 2-25. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
4EH	78	MSR_PPIN_CTL	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) Set 1 to prevent further writes to MSR_PPIN_CTL. Writing 1 to MSR_PPIN_CTL[bit 0] is permitted only if MSR_PPIN_CTL[bit 1] is clear, Default is 0. BIOS should provide an opt-in menu to enable the user to turn on MSR_PPIN_CTL[bit 1] for privileged inventory initialization agent to access MSR_PPIN. After reading MSR_PPIN, the privileged inventory initialization agent should write '01b' to MSR_PPIN_CTL to disable further access to MSR_PPIN and prevent unauthorized modification to MSR_PPIN_CTL.
		1		Enable_PPIN (R/W) If 1, enables MSR_PPIN to be accessible using RDMSR. Once set, attempt to write 1 to MSR_PPIN_CTL[bit 0] will cause #GP. If 0, an attempt to read MSR_PPIN will cause #GP. Default is 0.
		63:2		Reserved.
4FH	79	MSR_PPIN	Package	Protected Processor Inventory Number (R/O)

**Table 2-25. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:0		<b>Protected Processor Inventory Number (R/O)</b> A unique value within a given CUID family/model/stepping signature that a privileged inventory initialization agent can access to identify each physical processor, when access to MSR_PPIN is enabled. Access to MSR_PPIN is permitted only if MSR_PPIN_CTL[bits 1:0] = '10b'
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		22:16		Reserved.
		23	Package	<b>PPIN_CAP (R/O)</b> When set to 1, indicates that Protected Processor Inventory Number (PPIN) capability can be enabled for privileged system inventory agent to read PPIN from MSR_PPIN. When set to 0, PPIN capability is not supported. An attempt to access MSR_PPIN_CTL or MSR_PPIN will cause #GP.
		27:24		Reserved.
		28	Package	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> 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	<b>Programmable TDP Limit for Turbo Mode (R/O)</b> 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.
		30	Package	<b>Programmable TJ OFFSET (R/O)</b> When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify an temperature offset.
		39:31		Reserved.
		47:40	Package	<b>Maximum Efficiency Ratio (R/O)</b> 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	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWait extension C-state parameters or ACPI C-States. See <a href="http://biosbits.org">http://biosbits.org</a> .

**Table 2-25. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		<b>Package C-State Limit (R/W)</b> 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		<b>I/O MWAIT Redirection Enable (R/W)</b> 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		<b>CFG Lock (R/WO)</b> When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
179H	377	IA32_MCG_CAP	Thread	<b>Global Machine Check Capability (R/O)</b>
		7:0		<b>Count</b>
		8		<b>MCG_CTL_P</b>
		9		<b>MCG_EXT_P</b>
		10		<b>MCP_CMCI_P</b>
		11		<b>MCG_TES_P</b>
		15:12		Reserved.
		23:16		<b>MCG_EXT_CNT</b>
		24		<b>MCG_SER_P</b>
		25		Reserved.
		26		<b>MCG_ELOG_P</b>
63:27		Reserved.		
17FH	383	MSR_ERROR_CONTROL	Package	<b>MC Bank Error Configuration (R/W)</b>
		0		Reserved
		1		<b>MemError Log Enable (R/W)</b> When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.

**Table 2-25. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved.
		23:16		<b>Temperature Target (RO)</b> The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		27:24		<b>TCC Activation Offset (R/W)</b> Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only MSR_PLATFORM_INFO.[30] is set.
		63:28		Reserved.
1AEH	430	MSR_TURBO_RATIO_LIMIT 1	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 9C</b> Maximum turbo ratio limit of 9 core active.
		15:8	Package	<b>Maximum Ratio Limit for 10C</b> Maximum turbo ratio limit of 10core active.
		23:16	Package	<b>Maximum Ratio Limit for 11C</b> Maximum turbo ratio limit of 11 core active.
		31:24	Package	<b>Maximum Ratio Limit for 12C</b> Maximum turbo ratio limit of 12 core active.
		63:32		Reserved
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.

**Table 2-25. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
296H	662	IA32_MC22_CTL2	Package	See Table 2-2.
297H	663	IA32_MC23_CTL2	Package	See Table 2-2.
298H	664	IA32_MC24_CTL2	Package	See Table 2-2.
299H	665	IA32_MC25_CTL2	Package	See Table 2-2.
29AH	666	IA32_MC26_CTL2	Package	See Table 2-2.
29BH	667	IA32_MC27_CTL2	Package	See Table 2-2.
29CH	668	IA32_MC28_CTL2	Package	See Table 2-2.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC error from the Intel QPI module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC error from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC error from the two home agents.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC error from the two home agents.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

**Table 2-25. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
42DH	1069	IA32_MC11_STATUS	Package	Bank MC11 reports MC error from a specific channel of the integrated memory controller.
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	
435H	1077	IA32_MC13_STATUS	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	
439H	1081	IA32_MC14_STATUS	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	
43DH	1085	IA32_MC15_STATUS	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	
441H	1089	IA32_MC16_STATUS	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	
445H	1093	IA32_MC17_STATUS	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	
449H	1097	IA32_MC18_STATUS	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	
44DH	1101	IA32_MC19_STATUS	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	

**Table 2-25. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
451H	1105	IA32_MC20_STATUS	Package	Bank MC20 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
458H	1112	IA32_MC22_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
459H	1113	IA32_MC22_STATUS	Package	
45AH	1114	IA32_MC22_ADDR	Package	
45BH	1115	IA32_MC22_MISC	Package	
45CH	1116	IA32_MC23_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
45DH	1117	IA32_MC23_STATUS	Package	
45EH	1118	IA32_MC23_ADDR	Package	
45FH	1119	IA32_MC23_MISC	Package	
460H	1120	IA32_MC24_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
461H	1121	IA32_MC24_STATUS	Package	
462H	1122	IA32_MC24_ADDR	Package	
463H	1123	IA32_MC24_MISC	Package	
464H	1124	IA32_MC25_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC25 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
465H	1125	IA32_MC25_STATUS	Package	
466H	1126	IA32_MC25_ADDR	Package	
467H	1127	IA32_MC25_MISC	Package	
468H	1128	IA32_MC26_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC26 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
469H	1129	IA32_MC26_STATUS	Package	
46AH	1130	IA32_MC26_ADDR	Package	
46BH	1131	IA32_MC26_MISC	Package	
46CH	1132	IA32_MC27_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC27 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
46DH	1133	IA32_MC27_STATUS	Package	
46EH	1134	IA32_MC27_ADDR	Package	
46FH	1135	IA32_MC27_MISC	Package	

**Table 2-25. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
470H	1136	IA32_MC28_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC28 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
471H	1137	IA32_MC28_STATUS	Package	
472H	1138	IA32_MC28_ADDR	Package	
473H	1139	IA32_MC28_MISC	Package	
613H	1555	MSR_PKG_PERF_STATUS	Package	<b>Package RAPL Perf Status (R/O)</b>
618H	1560	MSR_DRAM_POWER_LIMIT	Package	<b>DRAM RAPL Power Limit Control (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	<b>DRAM Energy Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	<b>DRAM Performance Throttling Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	<b>DRAM RAPL Parameters (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
639H	1593	MSR_PP0_ENERGY_STATUS	Package	<b>PP0 Energy Status (R/O)</b> See Section 14.9.4, "PP0/PP1 RAPL Domains."
See Table 2-19, for other MSR definitions applicable to Intel Xeon processor E5 v2 with CPUID signature 06_3EH				

### 2.11.2 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family

Intel® Xeon® processor E7 v2 family (based on Ivy Bridge-E microarchitecture) with CPUID DisplayFamily\_DisplayModel signature 06\_3EH supports the MSR interfaces listed in Table 2-19, Table 2-25, and Table 2-26.

**Table 2-26. Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily\_DisplayModel Signature 06\_3EH**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	<b>Control Features in Intel 64 Processor (R/W)</b> See Table 2-2.
		0		<b>Lock (R/WL)</b>
		1		<b>Enable VMX inside SMX operation (R/WL)</b>
		2		<b>Enable VMX outside SMX operation (R/WL)</b>
		14:8		<b>SENTER local functions enables (R/WL)</b>
		15		<b>SENTER global functions enable (R/WL)</b>
		63:16		Reserved.
179H	377	IA32_MCG_CAP	Thread	<b>Global Machine Check Capability (R/O)</b>
		7:0		<b>Count</b>
		8		<b>MCG_CTL_P</b>



**Table 2-26. Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily\_DisplayModel Signature 06\_3EH**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved.
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		63:25		Reserved.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status (R/WO)
		0		RIPV
		1		EIPV
		2		MCIP
		3		LMCE signaled
		63:4		Reserved.
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 9C</b> Maximum turbo ratio limit of 9 core active.
		15:8	Package	<b>Maximum Ratio Limit for 10C</b> Maximum turbo ratio limit of 10core active.
		23:16	Package	<b>Maximum Ratio Limit for 11C</b> Maximum turbo ratio limit of 11 core active.
		31:24	Package	<b>Maximum Ratio Limit for 12C</b> Maximum turbo ratio limit of 12 core active.
		39:32	Package	<b>Maximum Ratio Limit for 13C</b> Maximum turbo ratio limit of 13 core active.
		47:40	Package	<b>Maximum Ratio Limit for 14C</b> Maximum turbo ratio limit of 14 core active.
		55:48	Package	<b>Maximum Ratio Limit for 15C</b> Maximum turbo ratio limit of 15 core active.
		62:56		Reserved
		63	Package	<b>Semaphore for Turbo Ratio Limit Configuration</b> If 1, the processor uses override configuration <sup>1</sup> specified in MSR_TURBO_RATIO_LIMIT and MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).
29DH	669	IA32_MC29_CTL2	Package	See Table 2-2.
29EH	670	IA32_MC30_CTL2	Package	See Table 2-2.
29FH	671	IA32_MC31_CTL2	Package	See Table 2-2.

**Table 2-26. Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily\_DisplayModel Signature 06\_3EH**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		63:36		Reserved.
41BH	1051	IA32_MC6_MISC	Package	Misc MAC information of Integrated I/O. (R/O) see Section 15.3.2.4
		5:0		Recoverable Address LSB
		8:6		Address Mode
		15:9		Reserved
		31:16		PCI Express Requestor ID
		39:32		PCI Express Segment Number
		63:32		Reserved
474H	1140	IA32_MC29_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
475H	1141	IA32_MC29_STATUS	Package	
476H	1142	IA32_MC29_ADDR	Package	
477H	1143	IA32_MC29_MISC	Package	
478H	1144	IA32_MC30_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
479H	1145	IA32_MC30_STATUS	Package	
47AH	1146	IA32_MC30_ADDR	Package	
47BH	1147	IA32_MC30_MISC	Package	
47CH	1148	IA32_MC31_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
47DH	1149	IA32_MC31_STATUS	Package	
47EH	1150	IA32_MC31_ADDR	Package	
47FH	1147	IA32_MC31_MISC	Package	

See Table 2-19, Table 2-25 for other MSR definitions applicable to Intel Xeon processor E7 v2 with CPUID signature 06\_3AH.

**NOTES:**

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

### 2.11.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Intel Xeon Processor E5 v2 and E7 v2 families are based on the Ivy Bridge-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-23 and Table 2-27. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 v2 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_3EH.

**Table 2-27. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C00H		MSR_PMON_GLOBAL_CTL	Package	Uncore perfmon per-socket global control.
C01H		MSR_PMON_GLOBAL_STATUS	Package	Uncore perfmon per-socket global status.
C06H		MSR_PMON_GLOBAL_CONFIG	Package	Uncore perfmon per-socket global configuration.
C15H		MSR_U_PMON_BOX_STATUS	Package	Uncore U-box perfmon U-box wide status.
C35H		MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU perfmon box wide status.
D1AH		MSR_C0_PMON_BOX_FILTER1	Package	Uncore C-box 0 perfmon box wide filter1.
D3AH		MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-box 1 perfmon box wide filter1.
D5AH		MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-box 2 perfmon box wide filter1.
D7AH		MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-box 3 perfmon box wide filter1.
D9AH		MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-box 4 perfmon box wide filter1.
DBAH		MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-box 5 perfmon box wide filter1.
DDAH		MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-box 6 perfmon box wide filter1.
DFAH		MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-box 7 perfmon box wide filter1.
E04H		MSR_C8_PMON_BOX_CTL	Package	Uncore C-box 8 perfmon local box wide control.
E10H		MSR_C8_PMON_EVNTSEL0	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 0.
E11H		MSR_C8_PMON_EVNTSEL1	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 1.
E12H		MSR_C8_PMON_EVNTSEL2	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 2.
E13H		MSR_C8_PMON_EVNTSEL3	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 3.
E14H		MSR_C8_PMON_BOX_FILTER	Package	Uncore C-box 8 perfmon box wide filter.
E16H		MSR_C8_PMON_CTR0	Package	Uncore C-box 8 perfmon counter 0.
E17H		MSR_C8_PMON_CTR1	Package	Uncore C-box 8 perfmon counter 1.
E18H		MSR_C8_PMON_CTR2	Package	Uncore C-box 8 perfmon counter 2.
E19H		MSR_C8_PMON_CTR3	Package	Uncore C-box 8 perfmon counter 3.
E1AH		MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-box 8 perfmon box wide filter1.
E24H		MSR_C9_PMON_BOX_CTL	Package	Uncore C-box 9 perfmon local box wide control.
E30H		MSR_C9_PMON_EVNTSEL0	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 0.
E31H		MSR_C9_PMON_EVNTSEL1	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 1.
E32H		MSR_C9_PMON_EVNTSEL2	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 2.
E33H		MSR_C9_PMON_EVNTSEL3	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 3.
E34H		MSR_C9_PMON_BOX_FILTER	Package	Uncore C-box 9 perfmon box wide filter.
E36H		MSR_C9_PMON_CTR0	Package	Uncore C-box 9 perfmon counter 0.
E37H		MSR_C9_PMON_CTR1	Package	Uncore C-box 9 perfmon counter 1.

**Table 2-27. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E38H		MSR_C9_PMON_CTR2	Package	Uncore C-box 9 perfmon counter 2.
E39H		MSR_C9_PMON_CTR3	Package	Uncore C-box 9 perfmon counter 3.
E3AH		MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-box 9 perfmon box wide filter1.
E44H		MSR_C10_PMON_BOX_CTL	Package	Uncore C-box 10 perfmon local box wide control.
E50H		MSR_C10_PMON_EVNTSELO	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 0.
E51H		MSR_C10_PMON_EVNTSEL1	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 1.
E52H		MSR_C10_PMON_EVNTSEL2	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 2.
E53H		MSR_C10_PMON_EVNTSEL3	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 3.
E54H		MSR_C10_PMON_BOX_FILTER	Package	Uncore C-box 10 perfmon box wide filter.
E56H		MSR_C10_PMON_CTR0	Package	Uncore C-box 10 perfmon counter 0.
E57H		MSR_C10_PMON_CTR1	Package	Uncore C-box 10 perfmon counter 1.
E58H		MSR_C10_PMON_CTR2	Package	Uncore C-box 10 perfmon counter 2.
E59H		MSR_C10_PMON_CTR3	Package	Uncore C-box 10 perfmon counter 3.
E5AH		MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-box 10 perfmon box wide filter1.
E64H		MSR_C11_PMON_BOX_CTL	Package	Uncore C-box 11 perfmon local box wide control.
E70H		MSR_C11_PMON_EVNTSELO	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 0.
E71H		MSR_C11_PMON_EVNTSEL1	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 1.
E72H		MSR_C11_PMON_EVNTSEL2	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 2.
E73H		MSR_C11_PMON_EVNTSEL3	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 3.
E74H		MSR_C11_PMON_BOX_FILTER	Package	Uncore C-box 11 perfmon box wide filter.
E76H		MSR_C11_PMON_CTR0	Package	Uncore C-box 11 perfmon counter 0.
E77H		MSR_C11_PMON_CTR1	Package	Uncore C-box 11 perfmon counter 1.
E78H		MSR_C11_PMON_CTR2	Package	Uncore C-box 11 perfmon counter 2.
E79H		MSR_C11_PMON_CTR3	Package	Uncore C-box 11 perfmon counter 3.
E7AH		MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-box 11 perfmon box wide filter1.
E84H		MSR_C12_PMON_BOX_CTL	Package	Uncore C-box 12 perfmon local box wide control.
E90H		MSR_C12_PMON_EVNTSELO	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 0.
E91H		MSR_C12_PMON_EVNTSEL1	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 1.
E92H		MSR_C12_PMON_EVNTSEL2	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 2.
E93H		MSR_C12_PMON_EVNTSEL3	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 3.
E94H		MSR_C12_PMON_BOX_FILTER	Package	Uncore C-box 12 perfmon box wide filter.
E96H		MSR_C12_PMON_CTR0	Package	Uncore C-box 12 perfmon counter 0.
E97H		MSR_C12_PMON_CTR1	Package	Uncore C-box 12 perfmon counter 1.
E98H		MSR_C12_PMON_CTR2	Package	Uncore C-box 12 perfmon counter 2.
E99H		MSR_C12_PMON_CTR3	Package	Uncore C-box 12 perfmon counter 3.
E9AH		MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-box 12 perfmon box wide filter1.
EA4H		MSR_C13_PMON_BOX_CTL	Package	Uncore C-box 13 perfmon local box wide control.

Table 2-27. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
EB0H		MSR_C13_PMON_EVNTSEL0	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 0.
EB1H		MSR_C13_PMON_EVNTSEL1	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 1.
EB2H		MSR_C13_PMON_EVNTSEL2	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 2.
EB3H		MSR_C13_PMON_EVNTSEL3	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 3.
EB4H		MSR_C13_PMON_BOX_FILTER	Package	Uncore C-box 13 perfmon box wide filter.
EB6H		MSR_C13_PMON_CTR0	Package	Uncore C-box 13 perfmon counter 0.
EB7H		MSR_C13_PMON_CTR1	Package	Uncore C-box 13 perfmon counter 1.
EB8H		MSR_C13_PMON_CTR2	Package	Uncore C-box 13 perfmon counter 2.
EB9H		MSR_C13_PMON_CTR3	Package	Uncore C-box 13 perfmon counter 3.
EBAH		MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-box 13 perfmon box wide filter1.
EC4H		MSR_C14_PMON_BOX_CTL	Package	Uncore C-box 14 perfmon local box wide control.
ED0H		MSR_C14_PMON_EVNTSEL0	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 0.
ED1H		MSR_C14_PMON_EVNTSEL1	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 1.
ED2H		MSR_C14_PMON_EVNTSEL2	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 2.
ED3H		MSR_C14_PMON_EVNTSEL3	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 3.
ED4H		MSR_C14_PMON_BOX_FILTER	Package	Uncore C-box 14 perfmon box wide filter.
ED6H		MSR_C14_PMON_CTR0	Package	Uncore C-box 14 perfmon counter 0.
ED7H		MSR_C14_PMON_CTR1	Package	Uncore C-box 14 perfmon counter 1.
ED8H		MSR_C14_PMON_CTR2	Package	Uncore C-box 14 perfmon counter 2.
ED9H		MSR_C14_PMON_CTR3	Package	Uncore C-box 14 perfmon counter 3.
EDAH		MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-box 14 perfmon box wide filter1.

## 2.12 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS (BASED ON HASWELL MICROARCHITECTURE)

The 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v3 product family (based on Haswell microarchitecture), with CPUID DisplayFamily\_DisplayModel signature 06\_3CH/06\_45H/06\_46H, support the MSR interfaces listed in Table 2-19, Table 2-20, Table 2-21, and Table 2-28. For an MSR listed in Table 2-19 that also appears in Table 2-28, Table 2-28 supercede Table 2-19.

The MSRs listed in Table 2-28 also apply to processors based on Haswell-E microarchitecture (see Section 2.13).

Table 2-28. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3BH	59	IA32_TSC_ADJUST	THREAD	<b>Per-Logical-Processor TSC ADJUST (R/W)</b> See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .

**Table 2-28. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		7:0		Reserved.
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> 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	<b>Programmable TDP Limit for Turbo Mode (R/O)</b> 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.
		31:30		Reserved.
		32	Package	<b>Low Power Mode Support (LPM) (R/O)</b> When set to 1, indicates that LPM is supported, and when set to 0, indicates LPM is not supported.
		34:33	Package	<b>Number of ConfigTDP Levels (R/O)</b> 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 11: Reserved
		39:35		Reserved.
		47:40	Package	<b>Maximum Efficiency Ratio (R/O)</b> The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		55:48	Package	<b>Minimum Operating Ratio (R/O)</b> Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved.
186H	390	IA32_PERFEVTSELO	THREAD	<b>Performance Event Select for Counter 0 (R/W)</b> Supports all fields described inTable 2-2 and the fields below.
		32		IN_TX: see Section 18.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
187H	391	IA32_PERFEVTSEL1	THREAD	<b>Performance Event Select for Counter 1 (R/W)</b> Supports all fields described inTable 2-2 and the fields below.
		32		IN_TX: see Section 18.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
188H	392	IA32_PERFEVTSEL2	THREAD	<b>Performance Event Select for Counter 2 (R/W)</b> Supports all fields described inTable 2-2 and the fields below.

**Table 2-28. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		32		IN_TX: see Section 18.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
		33		IN_TXCP: see Section 18.3.6.5.1 When IN_TXCP=1 & IN_TX=1 and in sampling, spurious PMI may occur and transactions may continuously abort near overflow conditions. Software should favor using IN_TXCP for counting over sampling. If sampling, software should use large “sample-after” value after clearing the counter configured to use IN_TXCP and also always reset the counter even when no overflow condition was reported.
189H	393	IA32_PERFEVTSEL3	THREAD	<b>Performance Event Select for Counter 3 (R/W)</b> Supports all fields described in Table 2-2 and the fields below.
		32		IN_TX: see Section 18.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
1C8H	456	MSR_LBR_SELECT	Thread	<b>Last Branch Record Filtering Select Register (R/W)</b>
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
		9		EN_CALL_STACK
		63:9		Reserved.
1D9H	473	IA32_DEBUGCTL	Thread	<b>Debug Control (R/W)</b> See Table 2-2.
		0		LBR: Last Branch Record
		1		BTF
		5:2		Reserved.
		6		TR: Branch Trace
		7		BTS: Log Branch Trace Message to BTS buffer
		8		BTINT
		9		BTS_OFF_OS
		10		BTS_OFF_USER
		11		FREEZE_LBR_ON_PMI
		12		FREEZE_PERFMON_ON_PMI

**Table 2-28. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures**

Register Address		Register Name	Scope	Bit Description	
Hex	Dec				
		13		<b>ENABLE_UNCORE_PMI</b>	
		14		<b>FREEZE_WHILE_SMM</b>	
		15		<b>RTM_DEBUG</b>	
		63:15		Reserved.	
491H	1169	IA32_VMX_VMFUNC	THREAD	<b>Capability Reporting Register of VM-function Controls (R/O)</b> See Table 2-2	
60BH	1548	MSR_PKGC_IRTL1	Package	<b>Package C6/C7 Interrupt Response Limit 1 (R/W)</b> This MSR defines the interrupt response time limit used by the processor to manage transition to package C6 or C7 state. The latency programmed in this register is for the shorter-latency sub C-states used by an MWAIT hint to C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.	
				9:0	<b>Interrupt response time limit (R/W)</b> Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.
				12:10	<b>Time Unit (R/W)</b> Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-19 for supported time unit encodings.
				14:13	Reserved.
				15	<b>Valid (R/W)</b> Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
				63:16	Reserved.
60CH	1548	MSR_PKGC_IRTL2	Package	<b>Package C6/C7 Interrupt Response Limit 2 (R/W)</b> This MSR defines the interrupt response time limit used by the processor to manage transition to package C6 or C7 state. The latency programmed in this register is for the longer-latency sub C-states used by an MWAIT hint to C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.	
				9:0	<b>Interrupt response time limit (R/W)</b> Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.
				12:10	<b>Time Unit (R/W)</b> Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-19 for supported time unit encodings.
				14:13	Reserved.
				15	<b>Valid (R/W)</b> Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
				63:16	Reserved.



**Table 2-28. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
613H	1555	MSR_PKG_PERF_STATUS	Package	<b>PKG Perf Status (R/O)</b> See Section 14.9.3, "Package RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	<b>DRAM Energy Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	<b>DRAM Performance Throttling Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	<b>Base TDP Ratio (R/O)</b>
		7:0		<b>Config_TDP_Base</b> Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved.
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O)
		14:0		PKG_TDP_LVL1. Power setting for ConfigTDP Level 1.
		15		Reserved
		23:16		Config_TDP_LVL1_Ratio. ConfigTDP level 1 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL1. Max Power setting allowed for ConfigTDP Level 1.
		62:47		PKG_MIN_PWR_LVL1. MIN Power setting allowed for ConfigTDP Level 1.
		63		Reserved.
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O)
		14:0		PKG_TDP_LVL2. Power setting for ConfigTDP Level 2.
		15		Reserved
		23:16		Config_TDP_LVL2_Ratio. ConfigTDP level 2 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL2. Max Power setting allowed for ConfigTDP Level 2.
		62:47		PKG_MIN_PWR_LVL2. MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved.
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	<b>ConfigTDP Control (R/W)</b>
		1:0		<b>TDP_LEVEL (RW/L)</b> System BIOS can program this field.
		30:2		Reserved.

**Table 2-28. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31		<b>Config_TDP_Lock (R/W/L)</b> When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	<b>ConfigTDP Control (R/W)</b>
		7:0		<b>MAX_NON_TURBO_RATIO (R/W/L)</b> System BIOS can program this field.
		30:8		Reserved.
		31		<b>TURBO_ACTIVATION_RATIO_Lock (R/W/L)</b> When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
C80H	3200	IA32_DEBUG_INTERFACE	Package	<b>Silicon Debug Feature Control (R/W)</b> See Table 2-2.

### 2.12.1 MSRs in 4th Generation Intel® Core™ Processor Family (based on Haswell Microarchitecture)

Table 2-29 lists model-specific registers (MSRs) that are specific to 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_3CH/06\_45H/06\_46H, see Table 2-1.

**Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See <a href="http://biosbits.org">http://biosbits.org</a> .

Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		3:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s Package C states C7 are not available to processor with signature 06_3CH
		9:4		Reserved
		10		<b>I/O MWAIT Redirection Enable (R/W)</b>
		14:11		Reserved
		15		<b>CFG Lock (R/WO)</b>
		24:16		Reserved
		25		<b>C3 State Auto Demotion Enable (R/W)</b>
		26		<b>C1 State Auto Demotion Enable (R/W)</b>
		27		<b>Enable C3 Undemotion (R/W)</b>
		28		<b>Enable C1 Undemotion (R/W)</b>
		63:29		Reserved
17DH	390	MSR_SMM_MCA_CAP	THREAD	<b>Enhanced SMM Capabilities (SMM-RO)</b> Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		<b>SMM_Code_Access_Chk (SMM-RO)</b> If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.
		59		<b>Long_Flow_Indication (SMM-RO)</b> If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.
		63:60		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.

**Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Core 0 select
		1		Core 1 select
		2		Core 2 select
		3		Core 3 select
		18:4		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
63:32		Reserved.		
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU main status
		0		Fixed counter overflowed
		1		An ARB counter overflowed
		2		Reserved
		3		A CBox counter overflowed (on any slice)
		63:4		Reserved.
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore fixed counter control (R/W)
		19:0		Reserved
		20		Enable overflow propagation
		21		Reserved
		22		Enable counting
		63:23		Reserved.
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box configuration information (R/O)
		3:0		Encoded number of C-Box, derive value by "-1"
		63:4		Reserved.
3B0H	946	MSR_UNC_ARB_PERFCTRO	Package	Uncore Arb unit, performance counter 0

Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb unit, performance counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb unit, counter 0 event select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, counter 1 event select MSR
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Core 0 select
		1		Core 1 select
		2		Core 2 select
		3		Core 3 select
		18:4		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
63:32		Reserved.		
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, counter 1 event select MSR
4E0H	1248	MSR_SMM_FEATURE_CONTR_OL	Package	<b>Enhanced SMM Feature Control (SMM-RW)</b> Reports SMM capability Enhancement. Accessible only while in SMM.
		0		<b>Lock (SMM-RW0)</b> When set to '1' locks this register from further changes
		1		Reserved
		2		<b>SMM_Code_Chk_En (SMM-RW)</b> This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	<b>SMM Delayed (SMM-RO)</b> Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.

**Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		N-1:0		<p><b>LOG_PROC_STATE (SMM-RO)</b></p> <p>Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle.</p> <p>The bit is automatically cleared at the end of each long event. The reset value of this field is 0.</p> <p>Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.</p>
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	<p><b>SMM Blocked (SMM-RO)</b></p> <p>Reports the blocked state of all logical processors in the package. Available only while in SMM.</p>
		N-1:0		<p><b>LOG_PROC_STATE (SMM-RO)</b></p> <p>Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPL or SENTER Sleep.</p> <p>The reset value of this field is 0FFFH.</p> <p>Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.</p>
		63:N		Reserved
606H	1542	MSR_RAPL_POWER_UNIT	Package	<p><b>Unit Multipliers used in RAPL Interfaces (R/O)</b></p>
		3:0	Package	<p><b>Power Units</b></p> <p>See Section 14.9.1, "RAPL Interfaces."</p>
		7:4	Package	Reserved
		12:8	Package	<p><b>Energy Status Units</b></p> <p>Energy related information (in Joules) is based on the multiplier, <math>1/2^{\wedge}ESU</math>; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)</p>
		15:13	Package	Reserved
		19:16	Package	<p><b>Time Units</b></p> <p>See Section 14.9.1, "RAPL Interfaces."</p>
		63:20		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	<p><b>PP0 Energy Status (R/O)</b></p> <p>See Section 14.9.4, "PP0/PP1 RAPL Domains."</p>
640H	1600	MSR_PP1_POWER_LIMIT	Package	<p><b>PP1 RAPL Power Limit Control (R/W)</b></p> <p>See Section 14.9.4, "PP0/PP1 RAPL Domains."</p>
641H	1601	MSR_PP1_ENERGY_STATUS	Package	<p><b>PP1 Energy Status (R/O)</b></p> <p>See Section 14.9.4, "PP0/PP1 RAPL Domains."</p>
642H	1602	MSR_PP1_POLICY	Package	<p><b>PP1 Balance Policy (R/W)</b></p> <p>See Section 14.9.4, "PP0/PP1 RAPL Domains."</p>

Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in Processor Cores (R/W)</b> (frequency refers to processor core frequency)
		0		<b>PROCHOT Status (R0)</b> When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved.
		4		<b>Graphics Driver Status (R0)</b> When set, frequency is reduced below the operating system request due to Processor Graphics driver override.
		5		<b>Autonomous Utilization-Based Frequency Control Status (R0)</b> When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		6		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		<b>Electrical Design Point Status (R0)</b> When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		<b>Core Power Limiting Status (R0)</b> When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		<b>Package-Level Power Limiting PL1 Status (R0)</b> When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		<b>Package-Level PL2 Power Limiting Status (R0)</b> When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		12		<b>Max Turbo Limit Status (R0)</b> When set, frequency is reduced below the operating system request due to multi-core turbo limits.
		13		<b>Turbo Transition Attenuation Status (R0)</b> When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
15:14		Reserved		

**Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.
		20		<b>Graphics Driver Log</b> When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		<b>Autonomous Utilization-Based Frequency Control Log</b> When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		<b>Electrical Design Point Log</b> When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		<b>Core Power Limiting Log</b> When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		<b>Package-Level PL1 Power Limiting Log</b> When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		<b>Package-Level PL2 Power Limiting Log</b> When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		<b>Max Turbo Limit Log</b> When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.



Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		29		<b>Turbo Transition Attenuation Log</b> When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved.
6B0H	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in the Processor Graphics (R/W)</b> (frequency refers to processor graphics frequency)
		0		<b>PROCHOT Status (R0)</b> When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved.
		4		<b>Graphics Driver Status (R0)</b> When set, frequency is reduced below the operating system request due to Processor Graphics driver override.
		5		<b>Autonomous Utilization-Based Frequency Control Status (R0)</b> When set, frequency is reduced below the operating system request because the processor has detected that utilization is low
		6		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		<b>Electrical Design Point Status (R0)</b> When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		<b>Graphics Power Limiting Status (R0)</b> When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		<b>Package-Level Power Limiting PL1 Status (R0)</b> When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		<b>Package-Level PL2 Power Limiting Status (R0)</b> When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		15:12		Reserved
16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.		

**Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.
		20		<b>Graphics Driver Log</b> When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		<b>Autonomous Utilization-Based Frequency Control Log</b> When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		<b>Electrical Design Point Log</b> When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		<b>Core Power Limiting Log</b> When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		<b>Package-Level PL1 Power Limiting Log</b> When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		<b>Package-Level PL2 Power Limiting Log</b> When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		<b>Max Turbo Limit Log</b> When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		<b>Turbo Transition Attenuation Log</b> When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:30		Reserved.
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in the Ring Interconnect (R/W) (frequency refers to ring interconnect in the uncore)</b>
		0		<b>PROCHOT Status (R0)</b> When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal event.
		5:2		Reserved.
		6		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		<b>Electrical Design Point Status (R0)</b> When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		Reserved.
		10		<b>Package-Level Power Limiting PL1 Status (R0)</b> When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		<b>Package-Level PL2 Power Limiting Status (R0)</b> When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		15:12		<b>Reserved</b>
		16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
19:18		Reserved.		
20		<b>Graphics Driver Log</b> When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.		

**Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		21		<b>Autonomous Utilization-Based Frequency Control Log</b> When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		<b>Electrical Design Point Log</b> When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		<b>Core Power Limiting Log</b> When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		<b>Package-Level PL1 Power Limiting Log</b> When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		<b>Package-Level PL2 Power Limiting Log</b> When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		<b>Max Turbo Limit Log</b> When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		<b>Turbo Transition Attenuation Log</b> When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved.
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, counter 0 event select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, counter 1 event select MSR
706H	1798	MSR_UNC_CBO_0_PERFCTRO	Package	Uncore C-Box 0, performance counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, performance counter 1

**Table 2-29. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
710H	1808	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, counter 0 event select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, counter 1 event select MSR
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, performance counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, performance counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, counter 0 event select MSR
721H	1824	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, counter 1 event select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, performance counter 1
730H	1840	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, counter 0 event select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, counter 1 event select MSR.
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, performance counter 0.
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, performance counter 1.
See Table 2-19, Table 2-20, Table 2-21, Table 2-24, Table 2-28 for other MSR definitions applicable to processors with CPUID signatures 063CH, 06_46H.				

## 2.12.2 Additional Residency MSRs Supported in 4th Generation Intel® Core™ Processors

The 4th generation Intel® Core™ processor family (based on Haswell microarchitecture) with CPUID DisplayFamily\_DisplayModel signature 06\_45H supports the MSR interfaces listed in Table 2-19, Table 2-20, Table 2-28, Table 2-29, and Table 2-30.

**Table 2-30. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily\_DisplayModel Signature 06\_45H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See <a href="http://biosbits.org">http://biosbits.org</a> .

**Table 2-30. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily\_DisplayModel Signature 06\_45H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		3:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10
		9:4		Reserved
		10		<b>I/O MWAIT Redirection Enable (R/W)</b>
		14:11		Reserved
		15		<b>CFG Lock (R/WO)</b>
		24:16		Reserved
		25		<b>C3 State Auto Demotion Enable (R/W)</b>
		26		<b>C1 State Auto Demotion Enable (R/W)</b>
		27		<b>Enable C3 Undemotion (R/W)</b>
		28		<b>Enable C1 Undemotion (R/W)</b>
		63:29		Reserved
630H	1584	MSR_PKG_C8_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C8 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.
		63:60		Reserved
631H	1585	MSR_PKG_C9_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C9 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.
		63:60		Reserved
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

**Table 2-30. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily\_DisplayModel Signature 06\_45H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		59:0		Package C10 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.
		63:60		Reserved

See Table 2-19, Table 2-20, Table 2-21, Table 2-28, Table 2-29 for other MSR definitions applicable to processors with CPUID signature 06\_45H.

## 2.13 MSRS IN INTEL® XEON® PROCESSOR E5 V3 AND E7 V3 PRODUCT FAMILY

Intel® Xeon® processor E5 v3 family and Intel® Xeon® processor E7 v3 family are based on Haswell-E microarchitecture (CPUID DisplayFamily\_DisplayModel = 06\_3F). These processors supports the MSR interfaces listed in Table 2-19, Table 2-28, and Table 2-31.

**Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
35H	53	MSR_CORE_THREAD_COUNT	Package	<b>Configured State of Enabled Processor Core Count and Logical Processor Count (RO)</b> <ul style="list-style-type: none"> <li>After a Power-On RESET, enumerates factory configuration of the number of processor cores and logical processors in the physical package.</li> <li>Following the sequence of (i) BIOS modified a Configuration Mask which selects a subset of processor cores to be active post RESET and (ii) a RESET event after the modification, enumerates the current configuration of enabled processor core count and logical processor count in the physical package.</li> </ul>
		15:0		<b>Core_COUNT (RO)</b> The number of processor cores that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.
		31:16		<b>THREAD_COUNT (RO)</b> The number of logical processors that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.
		63:32		Reserved
53H	83	MSR_THREAD_ID_INFO	Thread	<b>A Hardware Assigned ID for the Logical Processor (RO)</b>
		7:0		<b>Logical_Processor_ID (RO)</b> An implementation-specific numerical. value physically assigned to each logical processor. This ID is not related to Initial APIC ID or x2APIC ID, it is unique within a physical package.
		63:8		Reserved

**Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See <a href="http://biosbits.org">http://biosbits.org</a> .
		2:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		<b>I/O MWAIT Redirection Enable (R/W)</b>
		14:11		Reserved
		15		<b>CFG Lock (R/WO)</b>
		24:16		Reserved
		25		<b>C3 State Auto Demotion Enable (R/W)</b>
		26		<b>C1 State Auto Demotion Enable (R/W)</b>
		27		<b>Enable C3 Undemotion (R/W)</b>
		28		<b>Enable C1 Undemotion (R/W)</b>
		29		<b>Package C State Demotion Enable (R/W)</b>
		30		<b>Package C State UnDemotion Enable (R/W)</b>
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	<b>Global Machine Check Capability (R/O)</b>
		7:0		<b>Count</b>
		8		<b>MCG_CTL_P</b>
		9		<b>MCG_EXT_P</b>
		10		<b>MCP_CMCI_P</b>
		11		<b>MCG_TES_P</b>
		15:12		Reserved.
		23:16		<b>MCG_EXT_CNT</b>
		24		<b>MCG_SER_P</b>
		25		<b>MCG_EM_P</b>
		26		<b>MCG_ELOG_P</b>
		63:27		Reserved.



Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17DH	390	MSR_SMM_MCA_CAP	THREAD	<b>Enhanced SMM Capabilities (SMM-RO)</b> Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		<b>Reserved</b>
		58		<b>SMM_Code_Access_Chk (SMM-RO)</b> If set to 1 indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		<b>Long_Flow_Indication (SMM-RO)</b> If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
17FH	383	MSR_ERROR_CONTROL	Package	<b>MC Bank Error Configuration (R/W)</b>
		0		Reserved
		1		<b>MemError Log Enable (R/W)</b> When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		39:32	Package	<b>Maximum Ratio Limit for 5C</b> Maximum turbo ratio limit of 5 core active.
		47:40	Package	<b>Maximum Ratio Limit for 6C</b> Maximum turbo ratio limit of 6 core active.
		55:48	Package	<b>Maximum Ratio Limit for 7C</b> Maximum turbo ratio limit of 7 core active.
		63:56	Package	<b>Maximum Ratio Limit for 8C</b> Maximum turbo ratio limit of 8 core active.
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1

**Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		7:0	Package	<b>Maximum Ratio Limit for 9C</b> Maximum turbo ratio limit of 9 core active.
		15:8	Package	<b>Maximum Ratio Limit for 10C</b> Maximum turbo ratio limit of 10 core active.
		23:16	Package	<b>Maximum Ratio Limit for 11C</b> Maximum turbo ratio limit of 11 core active.
		31:24	Package	<b>Maximum Ratio Limit for 12C</b> Maximum turbo ratio limit of 12 core active.
		39:32	Package	<b>Maximum Ratio Limit for 13C</b> Maximum turbo ratio limit of 13 core active.
		47:40	Package	<b>Maximum Ratio Limit for 14C</b> Maximum turbo ratio limit of 14 core active.
		55:48	Package	<b>Maximum Ratio Limit for 15C</b> Maximum turbo ratio limit of 15 core active.
		63:56	Package	<b>Maximum Ratio Limit for 16C</b> Maximum turbo ratio limit of 16 core active.
1AFH	431	MSR_TURBO_RATIO_LIMIT2	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 17C</b> Maximum turbo ratio limit of 17 core active.
		15:8	Package	<b>Maximum Ratio Limit for 18C</b> Maximum turbo ratio limit of 18 core active.
		62:16	Package	Reserved
		63	Package	<b>Semaphore for Turbo Ratio Limit Configuration</b> If 1, the processor uses override configuration <sup>1</sup> specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1 and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC error from the Intel QPI 0 module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC error from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	

**Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC error from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC error from the home agent HA 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	

**Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC error from the Intel QPI 1 module.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC error from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	<b>Unit Multipliers used in RAPL Interfaces (R/O)</b>
		3:0	Package	<b>Power Units</b> See Section 14.9.1, "RAPL Interfaces."
		7:4	Package	<b>Reserved</b>
		12:8	Package	<b>Energy Status Units</b> Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$ ; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)
		15:13	Package	<b>Reserved</b>
		19:16	Package	<b>Time Units</b> See Section 14.9.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	<b>DRAM RAPL Power Limit Control (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."

Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	<b>DRAM Energy Status (R/O)</b> Energy Consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	<b>DRAM Performance Throttling Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	<b>DRAM RAPL Parameters (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
61EH	1566	MSR_PCIE_PLL_RATIO	Package	<b>Configuration of PCIE PLL Relative to BCLK(R/W)</b>
		1:0	Package	<b>PCIE Ratio (R/W)</b> 00b: Use 5:5 mapping for 100MHz operation (default) 01b: Use 5:4 mapping for 125MHz operation 10b: Use 5:3 mapping for 166MHz operation 11b: Use 5:2 mapping for 250MHz operation
		2	Package	<b>LPLL Select (R/W)</b> if 1, use configured setting of PCIE Ratio
		3	Package	<b>LONG RESET (R/W)</b> if 1, wait additional time-out before re-locking Gen2/Gen3 PLLs.
		63:4		Reserved
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	<b>Uncore Ratio Limit (R/W)</b> Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved.
		14:8		<b>MIN_RATIO</b> Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved.
		6:0		<b>MAX_RATIO</b> This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>Reserved (R/O)</b> Reads return 0
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in Processor Cores (R/W)</b> <b>(frequency refers to processor core frequency)</b>
		0		<b>PROCHOT Status (R0)</b> When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.

**Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal event.
		2		<b>Power Budget Management Status (R0)</b> When set, frequency is reduced below the operating system request due to PBM limit
		3		<b>Platform Configuration Services Status (R0)</b> When set, frequency is reduced below the operating system request due to PCS limit
		4		Reserved.
		5		<b>Autonomous Utilization-Based Frequency Control Status (R0)</b> When set, frequency is reduced below the operating system request because the processor has detected that utilization is low
		6		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		<b>Electrical Design Point Status (R0)</b> When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		Reserved.
		10		<b>Multi-Core Turbo Status (R0)</b> When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits
		12:11		Reserved.
		13		<b>Core Frequency P1 Status (R0)</b> When set, frequency is reduced below max non-turbo P1
		14		<b>Core Max n-core Turbo Frequency Limiting Status (R0)</b> When set, frequency is reduced below max n-core turbo frequency
		15		<b>Core Frequency Limiting Status (R0)</b> When set, frequency is reduced below the operating system request.
		16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		18		<b>Power Budget Management Log</b> When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		<b>Platform Configuration Services Log</b> When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20		Reserved.
		21		<b>Autonomous Utilization-Based Frequency Control Log</b> When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		<b>Electrical Design Point Log</b> When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved.
		26		<b>Multi-Core Turbo Log</b> When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28:27		Reserved.
		29		<b>Core Frequency P1 Log</b> When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		<b>Core Max n-core Turbo Frequency Limiting Log</b> When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		31		<b>Core Frequency Limiting Log</b> When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:32		Reserved.

**Table 2-31. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C8DH	3213	IA32_QM_EVTSEL	THREAD	<b>Monitoring Event Select Register (R/W)</b> if CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1
		7:0		<b>EventID (Rw)</b> <b>Event encoding:</b> <b>0x0: no monitoring</b> <b>0x1: L3 occupancy monitoring</b> <b>all other encoding reserved.</b>
		31:8		Reserved.
		41:32		<b>RMID (Rw)</b>
		63:42		Reserved.
C8EH	3214	IA32_QM_CTR	THREAD	<b>Monitoring Counter Register (R/O).</b> if CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1
		61:0		<b>Resource Monitored Data</b>
		62		<b>Unavailable:</b> If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.
		63		<b>Error:</b> If 1, indicates and unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.
C8FH	3215	IA32_PQR_ASSOC	THREAD	<b>Resource Association Register (R/W).</b>
		9:0		<b>RMID</b>
		63: 10		<b>Reserved</b>

See Table 2-19, Table 2-28 for other MSR definitions applicable to processors with CPUID signature 06\_3FH.

**NOTES:**

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

**2.13.1 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family**

Intel Xeon Processor E5 v3 and E7 v3 family are based on the Haswell-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-32. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 v3 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily\_DisplayModel of 06\_3FH.

**Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
700H		MSR_PMON_GLOBAL_CTL	Package	Uncore perfmon per-socket global control.
701H		MSR_PMON_GLOBAL_STATUS	Package	Uncore perfmon per-socket global status.
702H		MSR_PMON_GLOBAL_CONFIG	Package	Uncore perfmon per-socket global configuration.
703H		MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-box UCLK fixed counter control
704H		MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-box UCLK fixed counter



Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
705H		MSR_U_PMON_EVNTSEL0	Package	Uncore U-box perfmon event select for U-box counter 0.
706H		MSR_U_PMON_EVNTSEL1	Package	Uncore U-box perfmon event select for U-box counter 1.
708H		MSR_U_PMON_BOX_STATUS	Package	Uncore U-box perfmon U-box wide status.
709H		MSR_U_PMON_CTR0	Package	Uncore U-box perfmon counter 0
70AH		MSR_U_PMON_CTR1	Package	Uncore U-box perfmon counter 1
710H		MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU perfmon for PCU-box-wide control
711H		MSR_PCU_PMON_EVNTSEL0	Package	Uncore PCU perfmon event select for PCU counter 0.
712H		MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU perfmon event select for PCU counter 1.
713H		MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU perfmon event select for PCU counter 2.
714H		MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU perfmon event select for PCU counter 3.
715H		MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU perfmon box-wide filter.
716H		MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU perfmon box wide status.
717H		MSR_PCU_PMON_CTR0	Package	Uncore PCU perfmon counter 0.
718H		MSR_PCU_PMON_CTR1	Package	Uncore PCU perfmon counter 1.
719H		MSR_PCU_PMON_CTR2	Package	Uncore PCU perfmon counter 2.
71AH		MSR_PCU_PMON_CTR3	Package	Uncore PCU perfmon counter 3.
720H		MSR_S0_PMON_BOX_CTL	Package	Uncore SBo 0 perfmon for SBo 0 box-wide control
721H		MSR_S0_PMON_EVNTSEL0	Package	Uncore SBo 0 perfmon event select for SBo 0 counter 0.
722H		MSR_S0_PMON_EVNTSEL1	Package	Uncore SBo 0 perfmon event select for SBo 0 counter 1.
723H		MSR_S0_PMON_EVNTSEL2	Package	Uncore SBo 0 perfmon event select for SBo 0 counter 2.
724H		MSR_S0_PMON_EVNTSEL3	Package	Uncore SBo 0 perfmon event select for SBo 0 counter 3.
725H		MSR_S0_PMON_BOX_FILTER	Package	Uncore SBo 0 perfmon box-wide filter.
726H		MSR_S0_PMON_CTR0	Package	Uncore SBo 0 perfmon counter 0.
727H		MSR_S0_PMON_CTR1	Package	Uncore SBo 0 perfmon counter 1.
728H		MSR_S0_PMON_CTR2	Package	Uncore SBo 0 perfmon counter 2.
729H		MSR_S0_PMON_CTR3	Package	Uncore SBo 0 perfmon counter 3.
72AH		MSR_S1_PMON_BOX_CTL	Package	Uncore SBo 1 perfmon for SBo 1 box-wide control
72BH		MSR_S1_PMON_EVNTSEL0	Package	Uncore SBo 1 perfmon event select for SBo 1 counter 0.
72CH		MSR_S1_PMON_EVNTSEL1	Package	Uncore SBo 1 perfmon event select for SBo 1 counter 1.
72DH		MSR_S1_PMON_EVNTSEL2	Package	Uncore SBo 1 perfmon event select for SBo 1 counter 2.
72EH		MSR_S1_PMON_EVNTSEL3	Package	Uncore SBo 1 perfmon event select for SBo 1 counter 3.
72FH		MSR_S1_PMON_BOX_FILTER	Package	Uncore SBo 1 perfmon box-wide filter.
730H		MSR_S1_PMON_CTR0	Package	Uncore SBo 1 perfmon counter 0.
731H		MSR_S1_PMON_CTR1	Package	Uncore SBo 1 perfmon counter 1.
732H		MSR_S1_PMON_CTR2	Package	Uncore SBo 1 perfmon counter 2.
733H		MSR_S1_PMON_CTR3	Package	Uncore SBo 1 perfmon counter 3.
734H		MSR_S2_PMON_BOX_CTL	Package	Uncore SBo 2 perfmon for SBo 2 box-wide control

**Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
735H		MSR_S2_PMON_EVNTSEL0	Package	Uncore SBo 2 perfmon event select for SBo 2 counter 0.
736H		MSR_S2_PMON_EVNTSEL1	Package	Uncore SBo 2 perfmon event select for SBo 2 counter 1.
737H		MSR_S2_PMON_EVNTSEL2	Package	Uncore SBo 2 perfmon event select for SBo 2 counter 2.
738H		MSR_S2_PMON_EVNTSEL3	Package	Uncore SBo 2 perfmon event select for SBo 2 counter 3.
739H		MSR_S2_PMON_BOX_FILTER	Package	Uncore SBo 2 perfmon box-wide filter.
73AH		MSR_S2_PMON_CTR0	Package	Uncore SBo 2 perfmon counter 0.
73BH		MSR_S2_PMON_CTR1	Package	Uncore SBo 2 perfmon counter 1.
73CH		MSR_S2_PMON_CTR2	Package	Uncore SBo 2 perfmon counter 2.
73DH		MSR_S2_PMON_CTR3	Package	Uncore SBo 2 perfmon counter 3.
73EH		MSR_S3_PMON_BOX_CTL	Package	Uncore SBo 3 perfmon for SBo 3 box-wide control
73FH		MSR_S3_PMON_EVNTSEL0	Package	Uncore SBo 3 perfmon event select for SBo 3 counter 0.
740H		MSR_S3_PMON_EVNTSEL1	Package	Uncore SBo 3 perfmon event select for SBo 3 counter 1.
741H		MSR_S3_PMON_EVNTSEL2	Package	Uncore SBo 3 perfmon event select for SBo 3 counter 2.
742H		MSR_S3_PMON_EVNTSEL3	Package	Uncore SBo 3 perfmon event select for SBo 3 counter 3.
743H		MSR_S3_PMON_BOX_FILTER	Package	Uncore SBo 3 perfmon box-wide filter.
744H		MSR_S3_PMON_CTR0	Package	Uncore SBo 3 perfmon counter 0.
745H		MSR_S3_PMON_CTR1	Package	Uncore SBo 3 perfmon counter 1.
746H		MSR_S3_PMON_CTR2	Package	Uncore SBo 3 perfmon counter 2.
747H		MSR_S3_PMON_CTR3	Package	Uncore SBo 3 perfmon counter 3.
E00H		MSR_CO_PMON_BOX_CTL	Package	Uncore C-box 0 perfmon for box-wide control
E01H		MSR_CO_PMON_EVNTSEL0	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 0.
E02H		MSR_CO_PMON_EVNTSEL1	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 1.
E03H		MSR_CO_PMON_EVNTSEL2	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 2.
E04H		MSR_CO_PMON_EVNTSEL3	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 3.
E05H		MSR_CO_PMON_BOX_FILTER0	Package	Uncore C-box 0 perfmon box wide filter 0.
E06H		MSR_CO_PMON_BOX_FILTER1	Package	Uncore C-box 0 perfmon box wide filter 1.
E07H		MSR_CO_PMON_BOX_STATUS	Package	Uncore C-box 0 perfmon box wide status.
E08H		MSR_CO_PMON_CTR0	Package	Uncore C-box 0 perfmon counter 0.
E09H		MSR_CO_PMON_CTR1	Package	Uncore C-box 0 perfmon counter 1.
E0AH		MSR_CO_PMON_CTR2	Package	Uncore C-box 0 perfmon counter 2.
E0BH		MSR_CO_PMON_CTR3	Package	Uncore C-box 0 perfmon counter 3.
E10H		MSR_C1_PMON_BOX_CTL	Package	Uncore C-box 1 perfmon for box-wide control
E11H		MSR_C1_PMON_EVNTSEL0	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 0.
E12H		MSR_C1_PMON_EVNTSEL1	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 1.
E13H		MSR_C1_PMON_EVNTSEL2	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 2.
E14H		MSR_C1_PMON_EVNTSEL3	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 3.
E15H		MSR_C1_PMON_BOX_FILTER0	Package	Uncore C-box 1 perfmon box wide filter 0.

Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E16H		MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-box 1 perfmon box wide filter1.
E17H		MSR_C1_PMON_BOX_STATUS	Package	Uncore C-box 1 perfmon box wide status.
E18H		MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter 0.
E19H		MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter 1.
E1AH		MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter 2.
E1BH		MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter 3.
E20H		MSR_C2_PMON_BOX_CTL	Package	Uncore C-box 2 perfmon for box-wide control
E21H		MSR_C2_PMON_EVNTSELO	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 0.
E22H		MSR_C2_PMON_EVNTSEL1	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 1.
E23H		MSR_C2_PMON_EVNTSEL2	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 2.
E24H		MSR_C2_PMON_EVNTSEL3	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 3.
E25H		MSR_C2_PMON_BOX_FILTER0	Package	Uncore C-box 2 perfmon box wide filter 0.
E26H		MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-box 2 perfmon box wide filter1.
E27H		MSR_C2_PMON_BOX_STATUS	Package	Uncore C-box 2 perfmon box wide status.
E28H		MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter 0.
E29H		MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter 1.
E2AH		MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter 2.
E2BH		MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter 3.
E30H		MSR_C3_PMON_BOX_CTL	Package	Uncore C-box 3 perfmon for box-wide control
E31H		MSR_C3_PMON_EVNTSELO	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 0.
E32H		MSR_C3_PMON_EVNTSEL1	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 1.
E33H		MSR_C3_PMON_EVNTSEL2	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 2.
E34H		MSR_C3_PMON_EVNTSEL3	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 3.
E35H		MSR_C3_PMON_BOX_FILTER0	Package	Uncore C-box 3 perfmon box wide filter 0.
E36H		MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-box 3 perfmon box wide filter1.
E37H		MSR_C3_PMON_BOX_STATUS	Package	Uncore C-box 3 perfmon box wide status.
E38H		MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter 0.
E39H		MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter 1.
E3AH		MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter 2.
E3BH		MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter 3.
E40H		MSR_C4_PMON_BOX_CTL	Package	Uncore C-box 4 perfmon for box-wide control
E41H		MSR_C4_PMON_EVNTSELO	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 0.
E42H		MSR_C4_PMON_EVNTSEL1	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 1.
E43H		MSR_C4_PMON_EVNTSEL2	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 2.
E44H		MSR_C4_PMON_EVNTSEL3	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 3.
E45H		MSR_C4_PMON_BOX_FILTER0	Package	Uncore C-box 4 perfmon box wide filter 0.
E46H		MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-box 4 perfmon box wide filter1.

Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E47H		MSR_C4_PMON_BOX_STATUS	Package	Uncore C-box 4 perfmon box wide status.
E48H		MSR_C4_PMON_CTRL0	Package	Uncore C-box 4 perfmon counter 0.
E49H		MSR_C4_PMON_CTRL1	Package	Uncore C-box 4 perfmon counter 1.
E4AH		MSR_C4_PMON_CTRL2	Package	Uncore C-box 4 perfmon counter 2.
E4BH		MSR_C4_PMON_CTRL3	Package	Uncore C-box 4 perfmon counter 3.
E50H		MSR_C5_PMON_BOX_CTL	Package	Uncore C-box 5 perfmon for box-wide control
E51H		MSR_C5_PMON_EVTSEL0	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 0.
E52H		MSR_C5_PMON_EVTSEL1	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 1.
E53H		MSR_C5_PMON_EVTSEL2	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 2.
E54H		MSR_C5_PMON_EVTSEL3	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 3.
E55H		MSR_C5_PMON_BOX_FILTER0	Package	Uncore C-box 5 perfmon box wide filter 0.
E56H		MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-box 5 perfmon box wide filter 1.
E57H		MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon box wide status.
E58H		MSR_C5_PMON_CTRL0	Package	Uncore C-box 5 perfmon counter 0.
E59H		MSR_C5_PMON_CTRL1	Package	Uncore C-box 5 perfmon counter 1.
E5AH		MSR_C5_PMON_CTRL2	Package	Uncore C-box 5 perfmon counter 2.
E5BH		MSR_C5_PMON_CTRL3	Package	Uncore C-box 5 perfmon counter 3.
E60H		MSR_C6_PMON_BOX_CTL	Package	Uncore C-box 6 perfmon for box-wide control
E61H		MSR_C6_PMON_EVTSEL0	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 0.
E62H		MSR_C6_PMON_EVTSEL1	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 1.
E63H		MSR_C6_PMON_EVTSEL2	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 2.
E64H		MSR_C6_PMON_EVTSEL3	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 3.
E65H		MSR_C6_PMON_BOX_FILTER0	Package	Uncore C-box 6 perfmon box wide filter 0.
E66H		MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-box 6 perfmon box wide filter 1.
E67H		MSR_C6_PMON_BOX_STATUS	Package	Uncore C-box 6 perfmon box wide status.
E68H		MSR_C6_PMON_CTRL0	Package	Uncore C-box 6 perfmon counter 0.
E69H		MSR_C6_PMON_CTRL1	Package	Uncore C-box 6 perfmon counter 1.
E6AH		MSR_C6_PMON_CTRL2	Package	Uncore C-box 6 perfmon counter 2.
E6BH		MSR_C6_PMON_CTRL3	Package	Uncore C-box 6 perfmon counter 3.
E70H		MSR_C7_PMON_BOX_CTL	Package	Uncore C-box 7 perfmon for box-wide control.
E71H		MSR_C7_PMON_EVTSEL0	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 0.
E72H		MSR_C7_PMON_EVTSEL1	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 1.
E73H		MSR_C7_PMON_EVTSEL2	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 2.
E74H		MSR_C7_PMON_EVTSEL3	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 3.
E75H		MSR_C7_PMON_BOX_FILTER0	Package	Uncore C-box 7 perfmon box wide filter 0.
E76H		MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-box 7 perfmon box wide filter 1.
E77H		MSR_C7_PMON_BOX_STATUS	Package	Uncore C-box 7 perfmon box wide status.

Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E78H		MSR_C7_PMON_CTRL0	Package	Uncore C-box 7 perfmon counter 0.
E79H		MSR_C7_PMON_CTRL1	Package	Uncore C-box 7 perfmon counter 1.
E7AH		MSR_C7_PMON_CTRL2	Package	Uncore C-box 7 perfmon counter 2.
E7BH		MSR_C7_PMON_CTRL3	Package	Uncore C-box 7 perfmon counter 3.
E80H		MSR_C8_PMON_BOX_CTL	Package	Uncore C-box 8 perfmon local box wide control.
E81H		MSR_C8_PMON_EVTSEL0	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 0.
E82H		MSR_C8_PMON_EVTSEL1	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 1.
E83H		MSR_C8_PMON_EVTSEL2	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 2.
E84H		MSR_C8_PMON_EVTSEL3	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 3.
E85H		MSR_C8_PMON_BOX_FILTER0	Package	Uncore C-box 8 perfmon box wide filter0.
E86H		MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-box 8 perfmon box wide filter1.
E87H		MSR_C8_PMON_BOX_STATUS	Package	Uncore C-box 8 perfmon box wide status.
E88H		MSR_C8_PMON_CTRL0	Package	Uncore C-box 8 perfmon counter 0.
E89H		MSR_C8_PMON_CTRL1	Package	Uncore C-box 8 perfmon counter 1.
E8AH		MSR_C8_PMON_CTRL2	Package	Uncore C-box 8 perfmon counter 2.
E8BH		MSR_C8_PMON_CTRL3	Package	Uncore C-box 8 perfmon counter 3.
E90H		MSR_C9_PMON_BOX_CTL	Package	Uncore C-box 9 perfmon local box wide control.
E91H		MSR_C9_PMON_EVTSEL0	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 0.
E92H		MSR_C9_PMON_EVTSEL1	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 1.
E93H		MSR_C9_PMON_EVTSEL2	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 2.
E94H		MSR_C9_PMON_EVTSEL3	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 3.
E95H		MSR_C9_PMON_BOX_FILTER0	Package	Uncore C-box 9 perfmon box wide filter0.
E96H		MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-box 9 perfmon box wide filter1.
E97H		MSR_C9_PMON_BOX_STATUS	Package	Uncore C-box 9 perfmon box wide status.
E98H		MSR_C9_PMON_CTRL0	Package	Uncore C-box 9 perfmon counter 0.
E99H		MSR_C9_PMON_CTRL1	Package	Uncore C-box 9 perfmon counter 1.
E9AH		MSR_C9_PMON_CTRL2	Package	Uncore C-box 9 perfmon counter 2.
E9BH		MSR_C9_PMON_CTRL3	Package	Uncore C-box 9 perfmon counter 3.
EA0H		MSR_C10_PMON_BOX_CTL	Package	Uncore C-box 10 perfmon local box wide control.
EA1H		MSR_C10_PMON_EVTSEL0	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 0.
EA2H		MSR_C10_PMON_EVTSEL1	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 1.
EA3H		MSR_C10_PMON_EVTSEL2	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 2.
EA4H		MSR_C10_PMON_EVTSEL3	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 3.
EA5H		MSR_C10_PMON_BOX_FILTER0	Package	Uncore C-box 10 perfmon box wide filter0.
EA6H		MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-box 10 perfmon box wide filter1.
EA7H		MSR_C10_PMON_BOX_STATUS	Package	Uncore C-box 10 perfmon box wide status.
EA8H		MSR_C10_PMON_CTRL0	Package	Uncore C-box 10 perfmon counter 0.

Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
EA9H		MSR_C10_PMON_CTR1	Package	Uncore C-box 10 perfmon counter 1.
EAAH		MSR_C10_PMON_CTR2	Package	Uncore C-box 10 perfmon counter 2.
EABH		MSR_C10_PMON_CTR3	Package	Uncore C-box 10 perfmon counter 3.
EB0H		MSR_C11_PMON_BOX_CTL	Package	Uncore C-box 11 perfmon local box wide control.
EB1H		MSR_C11_PMON_EVNTSELO	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 0.
EB2H		MSR_C11_PMON_EVNTSEL1	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 1.
EB3H		MSR_C11_PMON_EVNTSEL2	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 2.
EB4H		MSR_C11_PMON_EVNTSEL3	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 3.
EB5H		MSR_C11_PMON_BOX_FILTER0	Package	Uncore C-box 11 perfmon box wide filter0.
EB6H		MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-box 11 perfmon box wide filter1.
EB7H		MSR_C11_PMON_BOX_STATUS	Package	Uncore C-box 11 perfmon box wide status.
EB8H		MSR_C11_PMON_CTR0	Package	Uncore C-box 11 perfmon counter 0.
EB9H		MSR_C11_PMON_CTR1	Package	Uncore C-box 11 perfmon counter 1.
EBAH		MSR_C11_PMON_CTR2	Package	Uncore C-box 11 perfmon counter 2.
EBBH		MSR_C11_PMON_CTR3	Package	Uncore C-box 11 perfmon counter 3.
EC0H		MSR_C12_PMON_BOX_CTL	Package	Uncore C-box 12 perfmon local box wide control.
EC1H		MSR_C12_PMON_EVNTSELO	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 0.
EC2H		MSR_C12_PMON_EVNTSEL1	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 1.
EC3H		MSR_C12_PMON_EVNTSEL2	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 2.
EC4H		MSR_C12_PMON_EVNTSEL3	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 3.
EC5H		MSR_C12_PMON_BOX_FILTER0	Package	Uncore C-box 12 perfmon box wide filter0.
EC6H		MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-box 12 perfmon box wide filter1.
EC7H		MSR_C12_PMON_BOX_STATUS	Package	Uncore C-box 12 perfmon box wide status.
EC8H		MSR_C12_PMON_CTR0	Package	Uncore C-box 12 perfmon counter 0.
EC9H		MSR_C12_PMON_CTR1	Package	Uncore C-box 12 perfmon counter 1.
ECAH		MSR_C12_PMON_CTR2	Package	Uncore C-box 12 perfmon counter 2.
ECBH		MSR_C12_PMON_CTR3	Package	Uncore C-box 12 perfmon counter 3.
ED0H		MSR_C13_PMON_BOX_CTL	Package	Uncore C-box 13 perfmon local box wide control.
ED1H		MSR_C13_PMON_EVNTSELO	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 0.
ED2H		MSR_C13_PMON_EVNTSEL1	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 1.
ED3H		MSR_C13_PMON_EVNTSEL2	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 2.
ED4H		MSR_C13_PMON_EVNTSEL3	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 3.
ED5H		MSR_C13_PMON_BOX_FILTER0	Package	Uncore C-box 13 perfmon box wide filter0.
ED6H		MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-box 13 perfmon box wide filter1.
ED7H		MSR_C13_PMON_BOX_STATUS	Package	Uncore C-box 13 perfmon box wide status.
ED8H		MSR_C13_PMON_CTR0	Package	Uncore C-box 13 perfmon counter 0.
ED9H		MSR_C13_PMON_CTR1	Package	Uncore C-box 13 perfmon counter 1.



Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
EDA	H	MSR_C13_PMON_CTR2	Package	Uncore C-box 13 perfmon counter 2.
EDB	H	MSR_C13_PMON_CTR3	Package	Uncore C-box 13 perfmon counter 3.
EE0	H	MSR_C14_PMON_BOX_CTL	Package	Uncore C-box 14 perfmon local box wide control.
EE1	H	MSR_C14_PMON_EVNTSELO	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 0.
EE2	H	MSR_C14_PMON_EVNTSEL1	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 1.
EE3	H	MSR_C14_PMON_EVNTSEL2	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 2.
EE4	H	MSR_C14_PMON_EVNTSEL3	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 3.
EE5	H	MSR_C14_PMON_BOX_FILTER	Package	Uncore C-box 14 perfmon box wide filter0.
EE6	H	MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-box 14 perfmon box wide filter1.
EE7	H	MSR_C14_PMON_BOX_STATUS	Package	Uncore C-box 14 perfmon box wide status.
EE8	H	MSR_C14_PMON_CTR0	Package	Uncore C-box 14 perfmon counter 0.
EE9	H	MSR_C14_PMON_CTR1	Package	Uncore C-box 14 perfmon counter 1.
EEA	H	MSR_C14_PMON_CTR2	Package	Uncore C-box 14 perfmon counter 2.
EEB	H	MSR_C14_PMON_CTR3	Package	Uncore C-box 14 perfmon counter 3.
EF0	H	MSR_C15_PMON_BOX_CTL	Package	Uncore C-box 15 perfmon local box wide control.
EF1	H	MSR_C15_PMON_EVNTSELO	Package	Uncore C-box 15 perfmon event select for C-box 15 counter 0.
EF2	H	MSR_C15_PMON_EVNTSEL1	Package	Uncore C-box 15 perfmon event select for C-box 15 counter 1.
EF3	H	MSR_C15_PMON_EVNTSEL2	Package	Uncore C-box 15 perfmon event select for C-box 15 counter 2.
EF4	H	MSR_C15_PMON_EVNTSEL3	Package	Uncore C-box 15 perfmon event select for C-box 15 counter 3.
EF5	H	MSR_C15_PMON_BOX_FILTER0	Package	Uncore C-box 15 perfmon box wide filter0.
EF6	H	MSR_C15_PMON_BOX_FILTER1	Package	Uncore C-box 15 perfmon box wide filter1.
EF7	H	MSR_C15_PMON_BOX_STATUS	Package	Uncore C-box 15 perfmon box wide status.
EF8	H	MSR_C15_PMON_CTR0	Package	Uncore C-box 15 perfmon counter 0.
EF9	H	MSR_C15_PMON_CTR1	Package	Uncore C-box 15 perfmon counter 1.
EFA	H	MSR_C15_PMON_CTR2	Package	Uncore C-box 15 perfmon counter 2.
EFB	H	MSR_C15_PMON_CTR3	Package	Uncore C-box 15 perfmon counter 3.
F00	H	MSR_C16_PMON_BOX_CTL	Package	Uncore C-box 16 perfmon for box-wide control
F01	H	MSR_C16_PMON_EVNTSELO	Package	Uncore C-box 16 perfmon event select for C-box 16 counter 0.
F02	H	MSR_C16_PMON_EVNTSEL1	Package	Uncore C-box 16 perfmon event select for C-box 16 counter 1.
F03	H	MSR_C16_PMON_EVNTSEL2	Package	Uncore C-box 16 perfmon event select for C-box 16 counter 2.
F04	H	MSR_C16_PMON_EVNTSEL3	Package	Uncore C-box 16 perfmon event select for C-box 16 counter 3.
F05	H	MSR_C16_PMON_BOX_FILTER0	Package	Uncore C-box 16 perfmon box wide filter 0.
F06	H	MSR_C16_PMON_BOX_FILTER1	Package	Uncore C-box 16 perfmon box wide filter 1.
F07	H	MSR_C16_PMON_BOX_STATUS	Package	Uncore C-box 16 perfmon box wide status.
F08	H	MSR_C16_PMON_CTR0	Package	Uncore C-box 16 perfmon counter 0.
F09	H	MSR_C16_PMON_CTR1	Package	Uncore C-box 16 perfmon counter 1.
F0A	H	MSR_C16_PMON_CTR2	Package	Uncore C-box 16 perfmon counter 2.

**Table 2-32. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E0BH		MSR_C16_PMON_CTR3	Package	Uncore C-box 16 perfmon counter 3.
F10H		MSR_C17_PMON_BOX_CTL	Package	Uncore C-box 17 perfmon for box-wide control
F11H		MSR_C17_PMON_EVTNSELO	Package	Uncore C-box 17 perfmon event select for C-box 17 counter 0.
F12H		MSR_C17_PMON_EVTNSEL1	Package	Uncore C-box 17 perfmon event select for C-box 17 counter 1.
F13H		MSR_C17_PMON_EVTNSEL2	Package	Uncore C-box 17 perfmon event select for C-box 17 counter 2.
F14H		MSR_C17_PMON_EVTNSEL3	Package	Uncore C-box 17 perfmon event select for C-box 17 counter 3.
F15H		MSR_C17_PMON_BOX_FILTER0	Package	Uncore C-box 17 perfmon box wide filter 0.
F16H		MSR_C17_PMON_BOX_FILTER1	Package	Uncore C-box 17 perfmon box wide filter 1.
F17H		MSR_C17_PMON_BOX_STATUS	Package	Uncore C-box 17 perfmon box wide status.
F18H		MSR_C17_PMON_CTR0	Package	Uncore C-box 17 perfmon counter 0.
F19H		MSR_C17_PMON_CTR1	Package	Uncore C-box 17 perfmon counter 1.
F1AH		MSR_C17_PMON_CTR2	Package	Uncore C-box 17 perfmon counter 2.
F1BH		MSR_C17_PMON_CTR3	Package	Uncore C-box 17 perfmon counter 3.

## 2.14 MSRS IN INTEL® CORE™ M PROCESSORS AND 5TH GENERATION INTEL CORE PROCESSORS

The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors, and Intel® Xeon® Processor E3-1200 v4 family are based on the Broadwell microarchitecture. The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors have CPUID DisplayFamily\_DisplayModel signature 06\_3DH. Intel® Xeon® Processor E3-1200 v4 family and the 5th generation Intel® Core™ Processors have CPUID DisplayFamily\_DisplayModel signature 06\_47H. Processors with signatures 06\_3DH and 06\_47H support the MSR interfaces listed in Table 2-19, Table 2-20, Table 2-21, Table 2-24, Table 2-28, Table 2-29, Table 2-33, and Table 2-34. For an MSR listed in Table 2-34 that also appears in the model-specific tables of prior generations, Table 2-34 supercede prior generation tables.

Table 2-33 lists MSRs that are common to processors based on the Broadwell microarchitectures (including CPUID signatures 06\_3DH, 06\_47H, 06\_4FH, and 06\_56H).

**Table 2-33. Additional MSRs Common to Processors Based the Broadwell Microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
		0		Ovf_PMC0
		1		Ovf_PMC1
		2		Ovf_PMC2
		3		Ovf_PMC3
		31:4		Reserved.
		32		Ovf_FixedCtr0



Table 2-33. Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		33		Ovf_FixedCtr1
		34		Ovf_FixedCtr2
		54:35		Reserved.
		55		Trace_ToPA_PMI. See Section 35.2.6.2, "Table of Physical Addresses (ToPA)."
		60:56		Reserved.
		61		Ovf_Uncore
		62		Ovf_BufDSSAVE
		63		CondChgd
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2. See Section 18.6.2.2, "Global Counter Control Facilities."
		0		Set 1 to clear Ovf_PMC0
		1		Set 1 to clear Ovf_PMC1
		2		Set 1 to clear Ovf_PMC2
		3		Set 1 to clear Ovf_PMC3
		31:4		Reserved.
		32		Set 1 to clear Ovf_FixedCtr0
		33		Set 1 to clear Ovf_FixedCtr1
		34		Set 1 to clear Ovf_FixedCtr2
		54:35		Reserved.
		55		Set 1 to clear Trace_ToPA_PMI. See Section 35.2.6.2, "Table of Physical Addresses (ToPA)."
		60:56		Reserved.
		61		Set 1 to clear Ovf_Uncore
		62		Set 1 to clear Ovf_BufDSSAVE
		63		Set 1 to clear CondChgd
		560H	1376	IA32_RTIT_OUTPUT_BASE
6:0				Reserved.
MAXPHYADDR <sup>1</sup> -1:7				Base physical address.
63:MAXPHYADDR				Reserved.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	THREAD	Trace Output Mask Pointers Register (R/W)
		6:0		Reserved.
		31:7		MaskOffsetTableOffset
		63:32		Output Offset.
570H	1392	IA32_RTIT_CTL	Thread	Trace Control Register (R/W)
		0		TraceEn
		1		Reserved, MBZ.

**Table 2-33. Additional MSRs Common to Processors Based the Broadwell Microarchitectures**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		<b>OS</b>
		3		<b>User</b>
		6:4		Reserved, MBZ
		7		<b>CR3 filter</b>
		8		<b>ToPA; writing 0 will #GP if also setting TraceEn</b>
		9		Reserved, MBZ
		10		<b>TSCEn</b>
		11		<b>DisRETc</b>
		12		Reserved, MBZ
		13		<b>Reserved; writing 0 will #GP if also setting TraceEn</b>
		63:14		Reserved, MBZ.
571H	1393	IA32_RTIT_STATUS	Thread	<b>Tracing Status Register (R/W)</b>
		0		Reserved, writes ignored.
		1		<b>ContexEn</b> , writes ignored.
		2		<b>TriggerEn</b> , writes ignored.
		3		Reserved
		4		<b>Error (R/W)</b>
		5		<b>Stopped</b>
		63:6		Reserved, MBZ.
572H	1394	IA32_RTIT_CR3_MATCH	THREAD	<b>Trace Filter CR3 Match Register (R/W)</b>
		4:0		Reserved
		63:5		CR3[63:5] value to match
620H		MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved.
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved.
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.

**NOTES:**

1. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].

Table 2-34 lists MSRs that are specific to Intel Core M processors and 5th Generation Intel Core Processors.

Table 2-34. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See <a href="http://biosbits.org">http://biosbits.org</a> .
		3:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10
		9:4		Reserved
		10		<b>I/O MWAIT Redirection Enable (R/W)</b>
		14:11		Reserved
		15		<b>CFG Lock (R/WO)</b>
		24:16		Reserved
		25		<b>C3 State Auto Demotion Enable (R/W)</b>
		26		<b>C1 State Auto Demotion Enable (R/W)</b>
		27		<b>Enable C3 Undemotion (R/W)</b>
		28		<b>Enable C1 Undemotion (R/W)</b>
		29		<b>Enable Package C-State Auto-demotion (R/W)</b>
		30		<b>Enable Package C-State Undemotion (R/W)</b>
		63:31		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.

**Table 2-34. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		39:32	Package	<b>Maximum Ratio Limit for 5C</b> Maximum turbo ratio limit of 5core active.
		47:40	Package	<b>Maximum Ratio Limit for 6C</b> Maximum turbo ratio limit of 6core active.
		63:48		Reserved.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>PPO Energy Status (R/O)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."

See Table 2-19, Table 2-20, Table 2-21, Table 2-24, Table 2-28, Table 2-29, Table 2-33 for other MSR definitions applicable to processors with CPUID signature 06\_3DH.

## 2.15 MSRS IN INTEL® XEON® PROCESSORS E5 V4 FAMILY

The MSRs listed in Table 2-35 are available and common to Intel® Xeon® Processor D product Family (CPUID DisplayFamily\_DisplayModel = 06\_56H) and to Intel Xeon processors E5 v4, E7 v4 families (CPUID DisplayFamily\_DisplayModel = 06\_4FH). They are based on the Broadwell microarchitecture.

See Section 2.15.1 for lists of tables of MSRs that are supported by Intel® Xeon® Processor D Family.

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
4EH	78	MSR_PPIN_CTL	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/W0) See Table 2-25.
		1		Enable_PPIN (R/W) See Table 2-25.
		63:2		Reserved.
4FH	79	MSR_PPIN	Package	Protected Processor Inventory Number (R/O)
		63:0		<b>Protected Processor Inventory Number (R/O)</b> See Table 2-25.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> See Table 2-25.
		22:16		Reserved.

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23	Package	<b>PPIN_CAP (R/O)</b> See Table 2-25.
		27:24		Reserved.
		28	Package	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> See Table 2-25.
		29	Package	<b>Programmable TDP Limit for Turbo Mode (R/O)</b> See Table 2-25.
		30	Package	<b>Programmable TJ OFFSET (R/O)</b> See Table 2-25.
		39:31		Reserved.
		47:40	Package	<b>Maximum Efficiency Ratio (R/O)</b> See Table 2-25.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See <a href="http://biosbits.org">http://biosbits.org</a> .
		2:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		<b>I/O MWAIT Redirection Enable (R/W)</b>
		14:11		Reserved
		15		<b>CFG Lock (R/WO)</b>
		16		<b>Automatic C-State Conversion Enable (R/W)</b> If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6)
		24:17		Reserved
		25		<b>C3 State Auto Demotion Enable (R/W)</b>
		26		<b>C1 State Auto Demotion Enable (R/W)</b>
		27		<b>Enable C3 Undemotion (R/W)</b>
28		<b>Enable C1 Undemotion (R/W)</b>		

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		29		<b>Package C State Demotion Enable (R/W)</b>
		30		<b>Package C State UnDemotion Enable (R/W)</b>
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	<b>Global Machine Check Capability (R/O)</b>
		7:0		<b>Count</b>
		8		<b>MCG_CTL_P</b>
		9		<b>MCG_EXT_P</b>
		10		<b>MCP_CMCI_P</b>
		11		<b>MCG_TES_P</b>
		15:12		Reserved.
		23:16		<b>MCG_EXT_CNT</b>
		24		<b>MCG_SER_P</b>
		25		<b>MCG_EM_P</b>
		26		<b>MCG_ELOG_P</b>
63:27		Reserved.		
17DH	390	MSR_SMM_MCA_CAP	THREAD	<b>Enhanced SMM Capabilities (SMM-RO)</b> Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		<b>Reserved</b>
		58		<b>SMM_Code_Access_Chk (SMM-RO)</b> If set to 1 indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		<b>Long_Flow_Indication (SMM-RO)</b> If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
19CH	412	IA32_THERM_STATUS	Core	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.
		0		<b>Thermal status (RO)</b> See Table 2-2.
		1		<b>Thermal status log (R/WCO)</b> See Table 2-2.
		2		<b>PROTCHOT # or FORCEPR# status (RO)</b> See Table 2-2.
		3		<b>PROTCHOT # or FORCEPR# log (R/WCO)</b> See Table 2-2.
		4		<b>Critical Temperature status (RO)</b> See Table 2-2.

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		5		<b>Critical Temperature status log (R/WC0)</b> See Table 2-2.
		6		<b>Thermal threshold #1 status (RO)</b> See Table 2-2.
		7		<b>Thermal threshold #1 log (R/WC0)</b> See Table 2-2.
		8		<b>Thermal threshold #2 status (RO)</b> See Table 2-2.
		9		<b>Thermal threshold #2 log (R/WC0)</b> See Table 2-2.
		10		<b>Power Limitation status (RO)</b> See Table 2-2.
		11		<b>Power Limitation log (R/WC0)</b> See Table 2-2.
		12		<b>Current Limit status (RO)</b> See Table 2-2.
		13		<b>Current Limit log (R/WC0)</b> See Table 2-2.
		14		<b>Cross Domain Limit status (RO)</b> See Table 2-2.
		15		<b>Cross Domain Limit log (R/WC0)</b> See Table 2-2.
		22:16		<b>Digital Readout (RO)</b> See Table 2-2.
		26:23		Reserved.
		30:27		<b>Resolution in degrees Celsius (RO)</b> See Table 2-2.
		31		<b>Reading Valid (RO)</b> See Table 2-2.
63:32		Reserved.		
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved.
		23:16		<b>Temperature Target (RO)</b> See Table 2-25.
		27:24		<b>TCC Activation Offset (R/W)</b> See Table 2-25.
		63:28		Reserved.

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b>
		15:8	Package	<b>Maximum Ratio Limit for 2C</b>
		23:16	Package	<b>Maximum Ratio Limit for 3C</b>
		31:24	Package	<b>Maximum Ratio Limit for 4C</b>
		39:32	Package	<b>Maximum Ratio Limit for 5C</b>
		47:40	Package	<b>Maximum Ratio Limit for 6C</b>
		55:48	Package	<b>Maximum Ratio Limit for 7C</b>
		63:56	Package	<b>Maximum Ratio Limit for 8C</b>
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 9C</b>
		15:8	Package	<b>Maximum Ratio Limit for 10C</b>
		23:16	Package	<b>Maximum Ratio Limit for 11C</b>
		31:24	Package	<b>Maximum Ratio Limit for 12C</b>
		39:32	Package	<b>Maximum Ratio Limit for 13C</b>
		47:40	Package	<b>Maximum Ratio Limit for 14C</b>
		55:48	Package	<b>Maximum Ratio Limit for 15C</b>
		63:56	Package	<b>Maximum Ratio Limit for 16C</b>
606H	1542	MSR_RAPL_POWER_UNIT	Package	<b>Unit Multipliers used in RAPL Interfaces (R/O)</b>
		3:0	Package	<b>Power Units</b> See Section 14.9.1, "RAPL Interfaces."
		7:4	Package	<b>Reserved</b>
		12:8	Package	<b>Energy Status Units</b> Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$ ; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)
		15:13	Package	<b>Reserved</b>
		19:16	Package	<b>Time Units</b> See Section 14.9.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	<b>DRAM RAPL Power Limit Control (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	<b>DRAM Energy Status (R/O)</b> Energy consumed by DRAM devices



**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	<b>DRAM Performance Throttling Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	<b>DRAM RAPL Parameters (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved.
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved.
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>Reserved (R/O)</b> Reads return 0
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in Processor Cores (R/W) (frequency refers to processor core frequency)</b>
		0		<b>PROCHOT Status (R0)</b> When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal event.
		2		<b>Power Budget Management Status (R0)</b> When set, frequency is reduced below the operating system request due to PBM limit
		3		<b>Platform Configuration Services Status (R0)</b> When set, frequency is reduced below the operating system request due to PCS limit
		4		Reserved.
		5		<b>Autonomous Utilization-Based Frequency Control Status (R0)</b> When set, frequency is reduced below the operating system request because the processor has detected that utilization is low

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		6		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		<b>Electrical Design Point Status (R0)</b> When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		Reserved.
		10		<b>Multi-Core Turbo Status (R0)</b> When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits
		12:11		Reserved.
		13		<b>Core Frequency P1 Status (R0)</b> When set, frequency is reduced below max non-turbo P1
		14		<b>Core Max n-core Turbo Frequency Limiting Status (R0)</b> When set, frequency is reduced below max n-core turbo frequency
		15		<b>Core Frequency Limiting Status (R0)</b> When set, frequency is reduced below the operating system request.
		16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		<b>Power Budget Management Log</b> When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		<b>Platform Configuration Services Log</b> When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20		Reserved.
		21		<b>Autonomous Utilization-Based Frequency Control Log</b> When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		22		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		<b>Electrical Design Point Log</b> When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved.
		26		<b>Multi-Core Turbo Log</b> When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28:27		Reserved.
		29		<b>Core Frequency P1 Log</b> When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		<b>Core Max n-core Turbo Frequency Limiting Log</b> When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		31		<b>Core Frequency Limiting Log</b> When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:32		Reserved.
770H	1904	IA32_PM_ENABLE	Package	See Section 14.4.2, “Enabling HWP”
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 14.4.3, “HWP Performance Range and Dynamic Capabilities”
774H	1908	IA32_HWP_REQUEST	Thread	See Section 14.4.4, “Managing HWP”
		7:0		<b>Minimum Performance (R/W)</b>
		15:8		<b>Maximum Performance (R/W)</b>
		23:16		<b>Desired Performance (R/W)</b>
		63:24		Reserved.
777H	1911	IA32_HWP_STATUS	Thread	See Section 14.4.5, “HWP Feedback”
		1:0		Reserved.
		2		<b>Excursion to Minimum (RO)</b>
		63:3		Reserved.

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C8DH	3213	IA32_QM_EVTSEL	THREAD	<b>Monitoring Event Select Register (R/W)</b> if CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1
		7:0		<b>EventID (RW)</b> <b>Event encoding:</b> 0x00: no monitoring 0x01: L3 occupancy monitoring 0x02: Total memory bandwidth monitoring 0x03: Local memory bandwidth monitoring All other encoding reserved
		31:8		Reserved.
		41:32		<b>RMID (RW)</b>
		63:42		Reserved.
C8FH	3215	IA32_PQR_ASSOC	THREAD	<b>Resource Association Register (R/W)</b>
		9:0		<b>RMID</b>
		31:10		<b>Reserved</b>
		51:32		<b>COS (R/W).</b>
		63: 52		<b>Reserved</b>
C90H	3216	IA32_L3_QOS_MASK_0	Package	<b>L3 Class Of Service Mask - COS 0 (R/W)</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0
		0:19		<b>CBM: Bit vector of available L3 ways for COS 0 enforcement</b>
		63:20		<b>Reserved</b>
C91H	3217	IA32_L3_QOS_MASK_1	Package	<b>L3 Class Of Service Mask - COS 1 (R/W)</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1
		0:19		<b>CBM: Bit vector of available L3 ways for COS 1 enforcement</b>
		63:20		<b>Reserved</b>
C92H	3218	IA32_L3_QOS_MASK_2	Package	<b>L3 Class Of Service Mask - COS 2 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2
		0:19		<b>CBM: Bit vector of available L3 ways for COS 2 enforcement</b>
		63:20		<b>Reserved</b>
C93H	3219	IA32_L3_QOS_MASK_3	Package	<b>L3 Class Of Service Mask - COS 3 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3
		0:19		<b>CBM: Bit vector of available L3 ways for COS 3 enforcement</b>
		63:20		<b>Reserved</b>
C94H	3220	IA32_L3_QOS_MASK_4	Package	<b>L3 Class Of Service Mask - COS 4 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4
		0:19		<b>CBM: Bit vector of available L3 ways for COS 4 enforcement</b>
		63:20		<b>Reserved</b>

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C95H	3221	IA32_L3_QOS_MASK_5	Package	<b>L3 Class Of Service Mask - COS 5 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5
		0:19		<b>CBM: Bit vector of available L3 ways for COS 5 enforcement</b>
		63:20		<b>Reserved</b>
C96H	3222	IA32_L3_QOS_MASK_6	Package	<b>L3 Class Of Service Mask - COS 6 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6
		0:19		<b>CBM: Bit vector of available L3 ways for COS 6 enforcement</b>
		63:20		<b>Reserved</b>
C97H	3223	IA32_L3_QOS_MASK_7	Package	<b>L3 Class Of Service Mask - COS 7 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7
		0:19		<b>CBM: Bit vector of available L3 ways for COS 7 enforcement</b>
		63:20		<b>Reserved</b>
C98H	3224	IA32_L3_QOS_MASK_8	Package	<b>L3 Class Of Service Mask - COS 8 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8
		0:19		<b>CBM: Bit vector of available L3 ways for COS 8 enforcement</b>
		63:20		<b>Reserved</b>
C99H	3225	IA32_L3_QOS_MASK_9	Package	<b>L3 Class Of Service Mask - COS 9 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9
		0:19		<b>CBM: Bit vector of available L3 ways for COS 9 enforcement</b>
		63:20		<b>Reserved</b>
C9AH	3226	IA32_L3_QOS_MASK_10	Package	<b>L3 Class Of Service Mask - COS 10 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=10
		0:19		<b>CBM: Bit vector of available L3 ways for COS 10 enforcement</b>
		63:20		<b>Reserved</b>
C9BH	3227	IA32_L3_QOS_MASK_11	Package	<b>L3 Class Of Service Mask - COS 11 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11
		0:19		<b>CBM: Bit vector of available L3 ways for COS 11 enforcement</b>
		63:20		<b>Reserved</b>
C9CH	3228	IA32_L3_QOS_MASK_12	Package	<b>L3 Class Of Service Mask - COS 12 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=12
		0:19		<b>CBM: Bit vector of available L3 ways for COS 12 enforcement</b>
		63:20		<b>Reserved</b>
C9DH	3229	IA32_L3_QOS_MASK_13	Package	<b>L3 Class Of Service Mask - COS 13 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=13

**Table 2-35. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0:19		<b>CBM: Bit vector of available L3 ways for COS 13 enforcement</b>
		63:20		<b>Reserved</b>
C9EH	3230	IA32_L3_QOS_MASK_14	Package	<b>L3 Class Of Service Mask - COS 14 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >= 14
		0:19		<b>CBM: Bit vector of available L3 ways for COS 14 enforcement</b>
		63:20		<b>Reserved</b>
C9FH	3231	IA32_L3_QOS_MASK_15	Package	<b>L3 Class Of Service Mask - COS 15 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >= 15
		0:19		<b>CBM: Bit vector of available L3 ways for COS 15 enforcement</b>
		63:20		<b>Reserved</b>

### 2.15.1 Additional MSRs Supported in the Intel® Xeon® Processor D Product Family

The MSRs listed in Table 2-36 are available to Intel® Xeon® Processor D Product Family (CPUID DisplayFamily\_DisplayModel = 06\_56H). The Intel® Xeon® processor D product family is based on the Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-19, Table 2-28, Table 2-33, Table 2-35, and Table 2-36.

**Table 2-36. Additional MSRs Supported by Intel® Xeon® Processor D with DisplayFamily\_DisplayModel 06\_56H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	<b>Config Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		62:0	Package	Reserved
		63	Package	<b>Semaphore for Turbo Ratio Limit Configuration</b> If 1, the processor uses override configuration <sup>1</sup> specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.

**Table 2-36. Additional MSRs Supported by Intel® Xeon® Processor D with DisplayFamily\_DisplayModel 06\_56H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC error from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC error from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC error from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC error from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
See Table 2-19, Table 2-28, Table 2-33, and Table 2-35 for other MSR definitions applicable to processors with CPUID signature 06_56H.				

**NOTES:**

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

**2.15.2 Additional MSRs Supported in Intel® Xeon® Processors E5 v4 and E7 v4 Families**

The MSRs listed in Table 2-36 are available to Intel® Xeon® Processor E5 v4 and E7 v4 Families (CPUID DisplayFamily\_DisplayModel = 06\_4FH). The Intel® Xeon® processor E5 v4 family is based on the Broadwell

microarchitecture and supports the MSR interfaces listed in Table 2-19, Table 2-20, Table 2-28, Table 2-33, Table 2-35, and Table 2-37.

**Table 2-37. Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily\_DisplayModel 06\_4FH**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	<b>Config Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		62:0	Package	Reserved
		63	Package	<b>Semaphore for Turbo Ratio Limit Configuration</b> If 1, the processor uses override configuration <sup>1</sup> specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1 and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC error from the Intel QPI 0 module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC error from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	



**Table 2-37. Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily\_DisplayModel 06\_4FH**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC error from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC error from the home agent HA 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	

**Table 2-37. Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily\_DisplayModel 06\_4FH**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC error from the Intel QPI 1 module.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC error from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
C81H	3201	IA32_L3_QOS_CFG	Package	Cache Allocation Technology Configuration (R/W)
		0		CAT Enable. Set 1 to enable Cache Allocation Technology
		63:1		Reserved.

See Table 2-19, Table 2-20, Table 2-28, and Table 2-29 for other MSR definitions applicable to processors with CPUID signature 06\_45H.

**NOTES:**

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

## 2.16 MSRS IN THE 6TH GENERATION INTEL® CORE™ PROCESSORS, INTEL® XEON® PROCESSOR SCALABLE FAMILY, 7TH GENERATION INTEL® CORE™ PROCESSORS, AND FUTURE INTEL® CORE™ PROCESSORS

6th generation Intel® Core™ processors and the Intel® Xeon® Processor Scalable Family are based on the Skylake microarchitecture and have CPUID DisplayFamily\_DisplayModel signatures of 06\_4EH, 06\_5EH, and 06\_55H. 7th Generation Intel® Core™ processors are based on the Kaby Lake microarchitecture and have CPUID DisplayFamily\_DisplayModel signatures of 06\_8EH and 06\_9EH. Future Intel® Core™ processors are based on Cannon Lake microarchitecture and have a CPUID DisplayFamily\_DisplayModel signature of 06\_66H. These processors support the MSR interfaces listed in Table 2-19, Table 2-20, Table 2-24, Table 2-28, Table 2-34, Table 2-38, and Table 2-39. For an MSR listed in Table 2-38 that also appears in the model-specific tables of prior generations, Table 2-38 supercede prior generation tables.

The notation of “Platform” in the Scope column (with respect to MSR\_PLATFORM\_ENERGY\_COUNTER and MSR\_PLATFORM\_POWER\_LIMIT) is limited to the power-delivery domain and the specifics of the power delivery integration may vary by platform vendor’s implementation.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description	
Hex	Dec				
3AH	58	IA32_FEATURE_CONTROL	Thread	<b>Control Features in Intel 64 Processor (R/W)</b> See Table 2-2.	
FEH	254	IA32_MTRRCAP	Thread	<b>MTRR Capality (RO, Architectural).</b> See Table 2-2	
19CH	412	IA32_THERM_STATUS	Core	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.	
				0	<b>Thermal status (RO)</b> See Table 2-2.
				1	<b>Thermal status log (R/WCO)</b> See Table 2-2.
				2	<b>PROTCHOT # or FORCEPR# status (RO)</b> See Table 2-2.
				3	<b>PROTCHOT # or FORCEPR# log (R/WCO)</b> See Table 2-2.
				4	<b>Critical Temperature status (RO)</b> See Table 2-2.
				5	<b>Critical Temperature status log (R/WCO)</b> See Table 2-2.
				6	<b>Thermal threshold #1 status (RO)</b> See Table 2-2.
				7	<b>Thermal threshold #1 log (R/WCO)</b> See Table 2-2.
8	<b>Thermal threshold #2 status (RO)</b> See Table 2-2.				

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		9		<b>Thermal threshold #2 log (R/WCO)</b> See Table 2-2.
		10		<b>Power Limitation status (RO)</b> See Table 2-2.
		11		<b>Power Limitation log (R/WCO)</b> See Table 2-2.
		12		<b>Current Limit status (RO)</b> See Table 2-2.
		13		<b>Current Limit log (R/WCO)</b> See Table 2-2.
		14		<b>Cross Domain Limit status (RO)</b> See Table 2-2.
		15		<b>Cross Domain Limit log (R/WCO)</b> See Table 2-2.
		22:16		<b>Digital Readout (RO)</b> See Table 2-2.
		26:23		Reserved.
		30:27		<b>Resolution in degrees Celsius (RO)</b> See Table 2-2.
		31		<b>Reading Valid (RO)</b> See Table 2-2.
		63:32		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode</b> RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	<b>Maximum Ratio Limit for 1C</b> Maximum turbo ratio limit of 1 core active.
		15:8	Package	<b>Maximum Ratio Limit for 2C</b> Maximum turbo ratio limit of 2 core active.
		23:16	Package	<b>Maximum Ratio Limit for 3C</b> Maximum turbo ratio limit of 3 core active.
		31:24	Package	<b>Maximum Ratio Limit for 4C</b> Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Thread	<b>Last Branch Record Stack TOS (R/W)</b> Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See <a href="http://biosbits.org">http://biosbits.org</a> .

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0		Reserved.
		1	Package	<b>C1E Enable (R/W)</b> When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		18:2		Reserved.
		19		<b>Disable Race to Halt Optimization (R/W)</b> Setting this bit disables the Race to Halt optimization and avoid this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization. Default value is 1 for processors that do not support Race to Halt optimization.
		20		<b>Disable Energy Efficiency Optimization (R/W)</b> Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.
		63:21		Reserved.
300H	768	MSR_SGXOWNEREPOCH0	Package	<b>Lower 64 Bit CR_SGXOWNEREPOCH.</b> Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.
301H	768	MSR_SGXOWNEREPOCH1	Package	<b>Upper 64 Bit CR_SGXOWNEREPOCH.</b> Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 2-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	<b>Ovf_PMC0</b>
		1	Thread	<b>Ovf_PMC1</b>
		2	Thread	<b>Ovf_PMC2</b>
		3	Thread	<b>Ovf_PMC3</b>
		4	Thread	<b>Ovf_PMC4 (if CPUID.0AH:EAX[15:8] &gt; 4)</b>
		5	Thread	<b>Ovf_PMC5 (if CPUID.0AH:EAX[15:8] &gt; 5)</b>
		6	Thread	<b>Ovf_PMC6 (if CPUID.0AH:EAX[15:8] &gt; 6)</b>
7	Thread	<b>Ovf_PMC7 (if CPUID.0AH:EAX[15:8] &gt; 7)</b>		

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:8		Reserved.
		32	Thread	<b>Ovf_FixedCtr0</b>
		33	Thread	<b>Ovf_FixedCtr1</b>
		34	Thread	<b>Ovf_FixedCtr2</b>
		54:35		Reserved.
		55	Thread	<b>Trace_ToPA_PMI.</b>
		57:56		Reserved.
		58	Thread	<b>LBR_Frz.</b>
		59	Thread	<b>CTR_Frz.</b>
		60	Thread	<b>ASCI.</b>
		61	Thread	<b>Ovf_Uncore</b>
		62	Thread	<b>Ovf_BufDSSAVE</b>
		63	Thread	<b>CondChgd</b>
		390H	912	IA32_PERF_GLOBAL_STAT US_RESET
0	Thread			<b>Set 1 to clear Ovf_PMC0</b>
1	Thread			<b>Set 1 to clear Ovf_PMC1</b>
2	Thread			<b>Set 1 to clear Ovf_PMC2</b>
3	Thread			<b>Set 1 to clear Ovf_PMC3</b>
4	Thread			<b>Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] &gt; 4)</b>
5	Thread			<b>Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] &gt; 5)</b>
6	Thread			<b>Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] &gt; 6)</b>
7	Thread			<b>Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] &gt; 7)</b>
31:8				Reserved.
32	Thread			<b>Set 1 to clear Ovf_FixedCtr0</b>
33	Thread			<b>Set 1 to clear Ovf_FixedCtr1</b>
34	Thread			<b>Set 1 to clear Ovf_FixedCtr2</b>
54:35				Reserved.
55	Thread			<b>Set 1 to clear Trace_ToPA_PMI.</b>
57:56				Reserved.
58	Thread			<b>Set 1 to clear LBR_Frz.</b>
59	Thread			<b>Set 1 to clear CTR_Frz.</b>
60	Thread			<b>Set 1 to clear ASCI.</b>
61	Thread			<b>Set 1 to clear Ovf_Uncore</b>
62	Thread			<b>Set 1 to clear Ovf_BufDSSAVE</b>
63	Thread	<b>Set 1 to clear CondChgd</b>		

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
391H	913	IA32_PERF_GLOBAL_STAT_US_SET		See Table 2-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	<b>Set 1 to cause Ovf_PMC0 = 1</b>
		1	Thread	<b>Set 1 to cause Ovf_PMC1 = 1</b>
		2	Thread	<b>Set 1 to cause Ovf_PMC2 = 1</b>
		3	Thread	<b>Set 1 to cause Ovf_PMC3 = 1</b>
		4	Thread	<b>Set 1 to cause Ovf_PMC4=1 (if CPUID.0AH:EAX[15:8] &gt; 4)</b>
		5	Thread	<b>Set 1 to cause Ovf_PMC5=1 (if CPUID.0AH:EAX[15:8] &gt; 5)</b>
		6	Thread	<b>Set 1 to cause Ovf_PMC6=1 (if CPUID.0AH:EAX[15:8] &gt; 6)</b>
		7	Thread	<b>Set 1 to cause Ovf_PMC7=1 (if CPUID.0AH:EAX[15:8] &gt; 7)</b>
		31:8		Reserved.
		32	Thread	<b>Set 1 to cause Ovf_FixedCtr0 = 1</b>
		33	Thread	<b>Set 1 to cause Ovf_FixedCtr1 = 1</b>
		34	Thread	<b>Set 1 to cause Ovf_FixedCtr2 = 1</b>
		54:35		Reserved.
		55	Thread	<b>Set 1 to cause Trace_ToPA_PMI = 1</b>
		57:56		Reserved.
		58	Thread	<b>Set 1 to cause LBR_Frz = 1</b>
		59	Thread	<b>Set 1 to cause CTR_Frz = 1</b>
		60	Thread	<b>Set 1 to cause ASCII = 1</b>
		61	Thread	<b>Set 1 to cause Ovf_Uncore</b>
62	Thread	<b>Set 1 to cause Ovf_BufDSSAVE</b>		
63		Reserved.		
392H	913	IA32_PERF_GLOBAL_INUSE		See Table 2-2.
3F7H	1015	MSR_PEBS_FRONTEND	Thread	<b>FrontEnd Precise Event Condition Select (R/W)</b>
		2:0		<b>Event Code Select</b>
		3		Reserved.
		4		<b>Event Code Select High</b>
		7:5		Reserved.
		19:8		<b>IDQ_Bubble_Length Specifier</b>
		22:20		<b>IDQ_Bubble_Width Specifier</b>
		63:23		<b>Reserved</b>
500H	1280	IA32_SGX_SVN_STATUS	Thread	<b>Status and SVN Threshold of SGX Support for ACM (RO).</b>
		0		<b>Lock.</b> See Section 41.11.3, "Interactions with Authenticated Code Modules (ACMs)"
		15:1		Reserved.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23:16		<b>SGX_SVN_SINIT.</b> See Section 41.11.3, “Interactions with Authenticated Code Modules (ACMs)”
		63:24		Reserved.
560H	1376	IA32_RTIT_OUTPUT_BASE	Thread	<b>Trace Output Base Register (R/W).</b> See Table 2-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Thread	<b>Trace Output Mask Pointers Register (R/W).</b> See Table 2-2.
570H	1392	IA32_RTIT_CTL	Thread	<b>Trace Control Register (R/W)</b>
		0		<b>TraceEn</b>
		1		<b>CYCEn</b>
		2		<b>OS</b>
		3		<b>User</b>
		6:4		Reserved, MBZ
		7		<b>CR3 filter</b>
		8		<b>ToPA; writing 0 will #GP if also setting TraceEn</b>
		9		<b>MTCEn</b>
		10		<b>TSCEn</b>
		11		<b>DisRETc</b>
		12		Reserved, MBZ
		13		<b>BranchEn</b>
		17:14		<b>MTCFreq</b>
		18		Reserved, MBZ
		22:19		<b>CYCThresh</b>
		23		Reserved, MBZ
		27:24		<b>PSBFreq</b>
		31:28		Reserved, MBZ
		35:32		<b>ADDR0_CFG</b>
39:36		<b>ADDR1_CFG</b>		
63:40		Reserved, MBZ.		
571H	1393	IA32_RTIT_STATUS	Thread	<b>Tracing Status Register (R/W)</b>
		0		<b>FilterEn</b> , writes ignored.
		1		<b>ContexEn</b> , writes ignored.
		2		<b>TriggerEn</b> , writes ignored.
		3		Reserved
		4		<b>Error (R/W)</b>
		5		<b>Stopped</b>
		31:6		Reserved. MBZ



**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		48:32		<b>PacketByteCnt</b>
		63:49		Reserved, MBZ.
572H	1394	IA32_RTIT_CR3_MATCH	Thread	<b>Trace Filter CR3 Match Register (R/W)</b>
		4:0		Reserved
		63:5		CR3[63:5] value to match
580H	1408	IA32_RTIT_ADDR0_A	Thread	<b>Region 0 Start Address (R/W)</b>
		63:0		See Table 2-2.
581H	1409	IA32_RTIT_ADDR0_B	Thread	<b>Region 0 End Address (R/W)</b>
		63:0		See Table 2-2.
582H	1410	IA32_RTIT_ADDR1_A	Thread	<b>Region 1 Start Address (R/W)</b>
		63:0		See Table 2-2.
583H	1411	IA32_RTIT_ADDR1_B	Thread	<b>Region 1 End Address (R/W)</b>
		63:0		See Table 2-2.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>PPO Energy Status (R/O)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."
64DH	1613	MSR_PLATFORM_ENERGY_COUNTER	Platform*	Platform Energy Counter. (R/O). This MSR is valid only if both platform vendor hardware implementation and BIOS enablement support it. This MSR will read 0 if not valid.
		31:0		Total energy consumed by all devices in the platform that receive power from integrated power delivery mechanism, Included platform devices are processor cores, SOC, memory, add-on or peripheral devices that get powered directly from the platform power delivery means. The energy units are specified in the MSR_RAPL_POWER_UNIT.Energy_Status_Unit.
		63:32		Reserved.
64EH	1614	MSR_PPERF	Thread	Productive Performance Count. (R/O).
		63:0		Hardware's view of workload scalability. See Section 14.4.5.1
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in Processor Cores (R/W) (frequency refers to processor core frequency)</b>
		0		<b>PROCHOT Status (R0)</b> When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		4		<b>Residency State Regulation Status (R0)</b> When set, frequency is reduced below the operating system request due to residency state regulation limit.
		5		<b>Running Average Thermal Limit Status (R0)</b> When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).
		6		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).
		7		<b>VR Therm Design Current Status (R0)</b> When set, frequency is reduced below the operating system request due to VR thermal design current limit.
		8		<b>Other Status (R0)</b> When set, frequency is reduced below the operating system request due to electrical or other constraints.
		9		<b>Reserved</b>
		10		<b>Package/Platform-Level Power Limiting PL1 Status (R0)</b> When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.
		11		<b>Package/Platform-Level PL2 Power Limiting Status (R0)</b> When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.
		12		<b>Max Turbo Limit Status (R0)</b> When set, frequency is reduced below the operating system request due to multi-core turbo limits.
		13		<b>Turbo Transition Attenuation Status (R0)</b> When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
		15:14		<b>Reserved</b>
		16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		20		<b>Residency State Regulation Log</b> When set, indicates that the Residency State Regulation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		<b>Running Average Thermal Limit Log</b> When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		<b>VR Thermal Design Current Log</b> When set, indicates that the VR TDC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24		<b>Other Log</b> When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		<b>Reserved</b>
		26		<b>Package/Platform-Level PL1 Power Limiting Log</b> When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		<b>Package/Platform-Level PL2 Power Limiting Log</b> When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		<b>Max Turbo Limit Log</b> When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		<b>Turbo Transition Attenuation Log</b> When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved.
652H	1618	MSR_PKG_HDC_CONFIG	Package	<b>HDC Configuration (R/W).</b>

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		<b>PKG_Cx_Monitor.</b> Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY
		63:3		<b>Reserved</b>
653H	1619	MSR_CORE_HDC_RESIDENCY	Core	Core HDC Idle Residency. (R/O).
		63:0		Core_Cx_Duty_Cycle_Cnt.
655H	1621	MSR_PKG_HDC_SHALLOW_RESIDENCY	Package	Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle. (R/O).
		63:0		Pkg_C2_Duty_Cycle_Cnt.
656H	1622	MSR_PKG_HDC_DEEP_RESIDENCY	Package	Package Cx HDC Idle Residency. (R/O).
		63:0		Pkg_Cx_Duty_Cycle_Cnt.
658H	1624	MSR_WEIGHTED_CORE_CO	Package	Core-count Weighted C0 Residency. (R/O).
		63:0		Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.
659H	1625	MSR_ANY_CORE_CO	Package	Any Core C0 Residency. (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if any processor core in the package is in C0.
65AH	1626	MSR_ANY_GFXE_CO	Package	Any Graphics Engine C0 Residency. (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.
65BH	1627	MSR_CORE_GFXE_OVERLAP_CO	Package	Core and Graphics Engine Overlapped C0 Residency. (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.
65CH	1628	MSR_PLATFORM_POWER_LIMIT	Platform*	<b>Platform Power Limit Control (R/W-L)</b> Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor.  The processor implements an exponential-weighted algorithm in the placement of the time windows.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		14:0		Platform Power Limit #1. Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.
		15		Enable Platform Power Limit #1. When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #1 over the time window specified by Power Limit #1 Time Window.
		16		Platform Clamping Limitation #1. When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #1 value. This bit is writeable only when CPUID (EAX=6):EAX[4] is set.
		23:17		Time Window for Platform Power Limit #1. Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17]. The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, The unit is specified in MSR_RAPLPOWER_UNIT[Time Unit].
		31:24		Reserved
		46:32		Platform Power Limit #2. Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e. Platform Power Limit # 1)
		47		Enable Platform Power Limit #2. When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.
		48		Platform Clamping Limitation #2. When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.
		62:49		Reserved
		63		Lock. Setting this bit will lock all other bits of this MSR until system RESET.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
690H	1680	MSR_LASTBRANCH_16_FROM_IP	Thread	<b>Last Branch Record 16 From IP (R/W)</b> One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.12</li> </ul>
691H	1681	MSR_LASTBRANCH_17_FROM_IP	Thread	<b>Last Branch Record 17 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
692H	1682	MSR_LASTBRANCH_18_FROM_IP	Thread	<b>Last Branch Record 18 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
693H	1683	MSR_LASTBRANCH_19_FROM_IP	Thread	<b>Last Branch Record 19 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
694H	1684	MSR_LASTBRANCH_20_FROM_IP	Thread	<b>Last Branch Record 20 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
695H	1685	MSR_LASTBRANCH_21_FROM_IP	Thread	<b>Last Branch Record 21 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
696H	1686	MSR_LASTBRANCH_22_FROM_IP	Thread	<b>Last Branch Record 22 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
697H	1687	MSR_LASTBRANCH_23_FROM_IP	Thread	<b>Last Branch Record 23 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_LASTBRANCH_24_FROM_IP	Thread	<b>Last Branch Record 24 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
699H	1689	MSR_LASTBRANCH_25_FROM_IP	Thread	<b>Last Branch Record 25 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_LASTBRANCH_26_FROM_IP	Thread	<b>Last Branch Record 26 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_LASTBRANCH_27_FROM_IP	Thread	<b>Last Branch Record 27 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_LASTBRANCH_28_FROM_IP	Thread	<b>Last Branch Record 28 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
69DH	1693	MSR_LASTBRANCH_29_FROM_IP	Thread	<b>Last Branch Record 29 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_LASTBRANCH_30_FROM_IP	Thread	<b>Last Branch Record 30 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_LASTBRANCH_31_FROM_IP	Thread	<b>Last Branch Record 31 From IP (R/W)</b> See description of MSR_LASTBRANCH_0_FROM_IP.
6B0H	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in the Processor Graphics (R/W)</b> <b>(frequency refers to processor graphics frequency)</b>

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0		<b>PROCHOT Status (R0)</b> When set, frequency is reduced due to assertion of external PROCHOT.
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced due to a thermal event.
		4:2		Reserved.
		5		<b>Running Average Thermal Limit Status (R0)</b> When set, frequency is reduced due to running average thermal limit.
		6		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.
		7		<b>VR Thermal Design Current Status (R0)</b> When set, frequency is reduced due to VR TDC limit.
		8		<b>Other Status (R0)</b> When set, frequency is reduced due to electrical or other constraints.
		9		<b>Reserved</b>
		10		<b>Package/Platform-Level Power Limiting PL1 Status (R0)</b> When set, frequency is reduced due to package/platform-level power limiting PL1.
		11		<b>Package/Platform-Level PL2 Power Limiting Status (R0)</b> When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.
		12		<b>Inefficient Operation Status (R0)</b> When set, processor graphics frequency is operating below target frequency.
		15:13		<b>Reserved</b>
		16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20:18		Reserved.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		21		<b>Running Average Thermal Limit Log</b> When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		<b>VR Thermal Design Current Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24		<b>Other Log</b> When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		<b>Reserved</b>
		26		<b>Package/Platform-Level PL1 Power Limiting Log</b> When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		<b>Package/Platform-Level PL2 Power Limiting Log</b> When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		<b>Inefficient Operation Log</b> When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
				63:29
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in the Ring Interconnect (R/W) (frequency refers to ring interconnect in the uncore)</b>
		0		<b>PROCHOT Status (R0)</b> When set, frequency is reduced due to assertion of external PROCHOT.
		1		<b>Thermal Status (R0)</b> When set, frequency is reduced due to a thermal event.
		4:2		Reserved.



**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		5		<b>Running Average Thermal Limit Status (R0)</b> When set, frequency is reduced due to running average thermal limit.
		6		<b>VR Therm Alert Status (R0)</b> When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.
		7		<b>VR Thermal Design Current Status (R0)</b> When set, frequency is reduced due to VR TDC limit.
		8		<b>Other Status (R0)</b> When set, frequency is reduced due to electrical or other constraints.
		9		Reserved.
		10		<b>Package/Platform-Level Power Limiting PL1 Status (R0)</b> When set, frequency is reduced due to package/Platform-level power limiting PL1.
		11		<b>Package/Platform-Level PL2 Power Limiting Status (R0)</b> When set, frequency is reduced due to package/Platform-level power limiting PL2/PL3.
		15:12		<b>Reserved</b>
		16		<b>PROCHOT Log</b> When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		<b>Thermal Log</b> When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20:18		Reserved.
		21		<b>Running Average Thermal Limit Log</b> When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		<b>VR Therm Alert Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		<b>VR Thermal Design Current Log</b> When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		24		<b>Other Log</b> When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		<b>Reserved</b>
		26		<b>Package/Platform-Level PL1 Power Limiting Log</b> When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		<b>Package/Platform-Level PL2 Power Limiting Log</b> When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:28		Reserved.
6D0H	1744	MSR_LASTBRANCH_16_TO_IP	Thread	<b>Last Branch Record 16 To IP (R/W)</b> One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the <b>destination instruction</b> . See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.12</li> </ul>
6D1H	1745	MSR_LASTBRANCH_17_TO_IP	Thread	<b>Last Branch Record 17 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_LASTBRANCH_18_TO_IP	Thread	<b>Last Branch Record 18 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_LASTBRANCH_19_TO_IP	Thread	<b>Last Branch Record 19 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_LASTBRANCH_20_TO_IP	Thread	<b>Last Branch Record 20 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_LASTBRANCH_21_TO_IP	Thread	<b>Last Branch Record 21 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D6H	1750	MSR_LASTBRANCH_22_TO_IP	Thread	<b>Last Branch Record 22 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_LASTBRANCH_23_TO_IP	Thread	<b>Last Branch Record 23 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_LASTBRANCH_24_TO_IP	Thread	<b>Last Branch Record 24 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_LASTBRANCH_25_TO_IP	Thread	<b>Last Branch Record 25 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6DAH	1754	MSR_LASTBRANCH_26_TO_IP	Thread	<b>Last Branch Record 26 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DBH	1755	MSR_LASTBRANCH_27_TO_IP	Thread	<b>Last Branch Record 27 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_LASTBRANCH_28_TO_IP	Thread	<b>Last Branch Record 28 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_LASTBRANCH_29_TO_IP	Thread	<b>Last Branch Record 29 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DEH	1758	MSR_LASTBRANCH_30_TO_IP	Thread	<b>Last Branch Record 30 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_LASTBRANCH_31_TO_IP	Thread	<b>Last Branch Record 31 To IP (R/W)</b> See description of MSR_LASTBRANCH_0_TO_IP.
770H	1904	IA32_PM_ENABLE	Package	See Section 14.4.2, “Enabling HWP”
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 14.4.3, “HWP Performance Range and Dynamic Capabilities”
772H	1906	IA32_HWP_REQUEST_PKG	Package	See Section 14.4.4, “Managing HWP”
773H	1907	IA32_HWP_INTERRUPT	Thread	See Section 14.4.6, “HWP Notifications”
774H	1908	IA32_HWP_REQUEST	Thread	See Section 14.4.4, “Managing HWP”
		7:0		<b>Minimum Performance (R/W).</b>
		15:8		<b>Maximum Performance (R/W).</b>
		23:16		<b>Desired Performance (R/W).</b>
		31:24		<b>Energy/Performance Preference (R/W).</b>
		41:32		<b>Activity Window (R/W).</b>
		42		<b>Package Control (R/W).</b>
		63:43		Reserved.
777H	1911	IA32_HWP_STATUS	Thread	See Section 14.4.5, “HWP Feedback”
D90H	3472	IA32_BNDCFGS	Thread	See Table 2-2.
DA0H	3488	IA32_XSS	Thread	See Table 2-2.
DB0H	3504	IA32_PKG_HDC_CTL	Package	See Section 14.5.2, “Package level Enabling HDC”
DB1H	3505	IA32_PM_CTL1	Thread	See Section 14.5.3, “Logical-Processor Level HDC Control”
DB2H	3506	IA32_THREAD_STALL	Thread	See Section 14.5.4.1, “IA32_THREAD_STALL”
DC0H	3520	MSR_LBR_INFO_0	Thread	<b>Last Branch Record 0 Additional Information (R/W)</b> One of 32 triplet of last branch record registers on the last branch record stack. This part of the stack contains flag, TSX-related and elapsed cycle information. See also: <ul style="list-style-type: none"> <li>Last Branch Record Stack TOS at 1C9H</li> <li>Section 17.9.1, “LBR Stack.”</li> </ul>

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DC1H	3521	MSR_LBR_INFO_1	Thread	<b>Last Branch Record 1 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DC2H	3522	MSR_LBR_INFO_2	Thread	<b>Last Branch Record 2 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DC3H	3523	MSR_LBR_INFO_3	Thread	<b>Last Branch Record 3 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DC4H	3524	MSR_LBR_INFO_4	Thread	<b>Last Branch Record 4 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DC5H	3525	MSR_LBR_INFO_5	Thread	<b>Last Branch Record 5 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DC6H	3526	MSR_LBR_INFO_6	Thread	<b>Last Branch Record 6 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DC7H	3527	MSR_LBR_INFO_7	Thread	<b>Last Branch Record 7 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DC8H	3528	MSR_LBR_INFO_8	Thread	<b>Last Branch Record 8 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DC9H	3529	MSR_LBR_INFO_9	Thread	<b>Last Branch Record 9 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DCAH	3530	MSR_LBR_INFO_10	Thread	<b>Last Branch Record 10 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DCBH	3531	MSR_LBR_INFO_11	Thread	<b>Last Branch Record 11 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DCCH	3532	MSR_LBR_INFO_12	Thread	<b>Last Branch Record 12 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DCDH	3533	MSR_LBR_INFO_13	Thread	<b>Last Branch Record 13 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DCEH	3534	MSR_LBR_INFO_14	Thread	<b>Last Branch Record 14 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DCFH	3535	MSR_LBR_INFO_15	Thread	<b>Last Branch Record 15 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD0H	3536	MSR_LBR_INFO_16	Thread	<b>Last Branch Record 16 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD1H	3537	MSR_LBR_INFO_17	Thread	<b>Last Branch Record 17 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD2H	3538	MSR_LBR_INFO_18	Thread	<b>Last Branch Record 18 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD3H	3539	MSR_LBR_INFO_19	Thread	<b>Last Branch Record 19 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.

**Table 2-38. Additional MSRs Supported by 6th Generation Intel® Core™ Processors and the Intel® Xeon® Processor Scalable Family Based on Skylake Microarchitecture, 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture, and Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DD4H	3520	MSR_LBR_INFO_20	Thread	<b>Last Branch Record 20 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD5H	3521	MSR_LBR_INFO_21	Thread	<b>Last Branch Record 21 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD6H	3522	MSR_LBR_INFO_22	Thread	<b>Last Branch Record 22 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD7H	3523	MSR_LBR_INFO_23	Thread	<b>Last Branch Record 23 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD8H	3524	MSR_LBR_INFO_24	Thread	<b>Last Branch Record 24 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DD9H	3525	MSR_LBR_INFO_25	Thread	<b>Last Branch Record 25 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DDAH	3526	MSR_LBR_INFO_26	Thread	<b>Last Branch Record 26 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DDBH	3527	MSR_LBR_INFO_27	Thread	<b>Last Branch Record 27 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DDCH	3528	MSR_LBR_INFO_28	Thread	<b>Last Branch Record 28 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DDDH	3529	MSR_LBR_INFO_29	Thread	<b>Last Branch Record 29 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DDEH	3530	MSR_LBR_INFO_30	Thread	<b>Last Branch Record 30 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.
DDFH	3531	MSR_LBR_INFO_31	Thread	<b>Last Branch Record 31 Additional Information (R/W)</b> See description of MSR_LBR_INFO_0.

Table 2-39 lists the MSRs of uncore PMU for Intel processors with CPUID DisplayFamily\_DisplayModel signatures of 06\_4EH, 06\_5EH, 06\_8EH, 06\_9EH, and 06\_66H.

**Table 2-39. Uncore PMU MSRs Supported by 6th Generation Intel® Core™ Processors, 7th Generation Intel® Core™ Processors, and Future Intel® Core™ Processors**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore fixed counter control (R/W)
		19:0		Reserved
		20		Enable overflow propagation
		21		Reserved
		22		Enable counting

**Table 2-39. Uncore PMU MSRs Supported by 6th Generation Intel® Core™ Processors, 7th Generation Intel® Core™ Processors, and Future Intel® Core™ Processors**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:23		Reserved.
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore fixed counter
		43:0		Current count
		63:44		Reserved.
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box configuration information (R/O)
		3:0		Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics),
		63:4		Reserved.
3B0H	946	MSR_UNC_ARB_PERFCTRO	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb unit, performance counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb unit, counter 0 event select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, counter 1 event select MSR
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, counter 0 event select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, counter 1 event select MSR
706H	1798	MSR_UNC_CBO_0_PERFCTRO	Package	Uncore C-Box 0, performance counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, performance counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, counter 0 event select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, counter 1 event select MSR
716H	1814	MSR_UNC_CBO_1_PERFCTRO	Package	Uncore C-Box 1, performance counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, performance counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, counter 0 event select MSR
721H	1825	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, counter 1 event select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTRO	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, performance counter 1
730H	1840	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, counter 0 event select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, counter 1 event select MSR.
736H	1846	MSR_UNC_CBO_3_PERFCTRO	Package	Uncore C-Box 3, performance counter 0.

**Table 2-39. Uncore PMU MSRs Supported by 6th Generation Intel® Core™ Processors, 7th Generation Intel® Core™ Processors, and Future Intel® Core™ Processors**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, performance counter 1.
E01H	3585	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Slice 0 select
		1		Slice 1 select
		2		Slice 2 select
		3		Slice 3 select
		4		Slice 4select
		18:5		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
63:32		Reserved.		
E02H	3586	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU main status
		0		Fixed counter overflowed
		1		An ARB counter overflowed
		2		Reserved
		3		A CBox counter overflowed (on any slice)
		63:4		Reserved.

## 2.16.1 MSRs Specific to 7th Generation Intel® Core™ Processors based on Kaby Lake Microarchitecture

Table 2-41 lists additional MSRs for 7th generation Intel Core processors with a CPUID DisplayFamily\_DisplayModel signature of 06\_8EH and 06\_9EH. For an MSR listed in Table 2-41 that also appears in the model-specific tables of prior generations, Table 2-41 supersedes prior generation tables.

**Table 2-40. Additional MSRs Supported by 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
80H	128	MSR_TRACE_HUB_STH ACPIBAR_BASE	Package	NPK address used by AET messages (R/W)
		0		Lock Bit If set then this MSR cannot be re-written anymore. Lock bit has to be set in order for the AET packets to be directed to NPK MMIO.
		17:1		Reserved.
		63:18		ACPIBAR_BASE_ADDRESS AET target address in NPK MMIO space.
1F4H	500	MSR_PRMRR_PHYS_BASE	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MemType PRMRR BASE MemType.
		11:3		Reserved.
		45:12		Base PRMRR Base Address.
		63:46		Reserved.
1F5H	501	MSR_PRMRR_PHYS_MASK	Core	Processor Reserved Memory Range Register - Physical Mask Control Register (R/W)
		9:0		Reserved.
		10		Lock Lock bit for the PRMRR.
		11		VLD Enable bit for the PRMRR.
		45:12		Mask PRMRR MASK bits.
		63:46		Reserved.
1FBH	507	MSR_PRMRR_VALID_CONFIG	Core	Valid PRMRR Configurations (R/W)
		0		1M supported MEE size.
		4:1		Reserved.
		5		32M supported MEE size.
		6		64M supported MEE size.
		7		128M supported MEE size.



**Table 2-40. Additional MSRs Supported by 7th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:8		Reserved.
2F4H	756	MSR_UNCORE_PRMRR_PHYS_B ASE	Package	(R/W) The PRMRR range is used to protect Xucode memory from unauthorized reads and writes. Any IO access to this range is aborted. This register controls the location of the PRMRR range by indicating its starting address. It functions in tandem with the PRMRR mask register.
		11:0		Reserved.
		38:12		Range Base This field corresponds to bits 38:12 of the base address memory range which is allocated to PRMRR memory.
		63:39		Reserved.
2F5H	757	MSR_UNCORE_PRMRR_PHYS_ MASK	Package	(R/W) This register controls the size of the PRMRR range by indicating which address bits must match the PRMRR base register value.
		9:0		Reserved.
		10		Lock Setting this bit locks all writeable settings in this register, including itself.
		11		Range_En Indicates whether the PRMRR range is enabled and valid.
		38:12		Range_Mask This field indicates which address bits must match PRMRR base in order to qualify as an PRMRR access.
		63:39		Reserved.
620H	1568	MSR_RING_RATIO_LIMIT	Package	Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.
		6:0		MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.
		7		Reserved.
		14:8		MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.
		63:15		Reserved.

## 2.16.2 MSRs Specific to Future Intel® Core™ Processors

Table 2-41 lists additional MSRs for Future Intel Core processors with a CPUID DisplayFamily\_DisplayModel signature of 06\_66H. For an MSR listed in Table 2-41 that also appears in the model-specific tables of prior generations, Table 2-41 supersedes prior generation tables.

**Table 2-41. Additional MSRs Supported by Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	<b>Control Features in Intel 64 Processor (R/W)</b> See Table 2-2.
		0		<b>Lock (R/WL)</b>
		1		<b>Enable VMX inside SMX operation (R/WL)</b>
		2		<b>Enable VMX outside SMX operation (R/WL)</b>
		14:8		<b>SENTER local functions enables (R/WL)</b>
		15		<b>SENTER global functions enable (R/WL)</b>
		17		<b>SGX Launch Control Enable (R/WL)</b> This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Available only if CPUID.(EAX=07H, ECX=0H): ECX[30] = 1.
		18		<b>SGX global functions enable (R/WL)</b>
		63:21		Reserved.
350H	848	MSR_BR_DETECT_CTRL		Branch Monitoring Global Control (R/W)
		0		EnMonitoring Global enable for branch monitoring.
		1		EnExcept Enable branch monitoring event signaling on threshold trip. The branch monitoring event handler is signaled via the existing PMI signaling mechanism as programmed from the corresponding local APIC LVT entry.
		2		EnLBRFrz Enable LBR freeze on threshold trip. This will result in causing the LBR frozen bit 58 to be set in IA32_PERF_GLOBAL_STATUS when a triggering condition occurs and this bit is enabled.
		3		DisableInGuest When set to '1', branch monitoring, event triggering and LBR freeze actions are disabled when operating at VMX non-root operation.
		7:4		Reserved.
		17:8		WindowSize Window size defined by WindowCntSel. Values 0 - 1023 are supported.
		23:18		Reserved.

Table 2-41. Additional MSRs Supported by Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		25:24		WindowCntSel Window event count select: '00 = Instructions retired. '01 = Branch instructions retired '10 = Return instructions retired. '11 = Indirect branch instructions retired.
		26		CntAndMode When set to '1', overall branch monitoring event triggering condition is true only if all enabled counters' threshold conditions are true. When '0', the threshold tripping condition is true if any enabled counters' threshold is true.
		63:27		Reserved.
351H	849	MSR_BR_DETECT_STATUS		Branch Monitoring Global Status (R/W)
		0		Branch Monitoring Event Signaled When set to '1', Branch Monitoring event signaling is blocked until this bit is cleared by software.
		1		LBRsValid This status bit is set to '1' if the LBR state is considered valid for sampling by branch monitoring software.
		7:2		Reserved.
		8		CntrHit0 Branch monitoring counter #0 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.
		9		CntrHit1 Branch monitoring counter #1 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.
		15:10		Reserved. Reserved for additional branch monitoring counters threshold hit status.
		25:16		CountWindow The current value of window counter. The count value is frozen on a valid branch monitoring triggering condition. This is an 10-bit unsigned value.
		31:26		Reserved. Reserved for future extension of CountWindow.

**Table 2-41. Additional MSRs Supported by Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		39:32		Count0 The current value of counter 0 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit0 will also be set). This is an 8-bit signed value (2's complement). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).
		47:40		Count1 The current value of counter 1 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit1 will also be set). This is an 8-bit signed value (2's complement). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).
		63:48		Reserved.
354H - 355H	852 - 853	MSR_BR_DETECT_COUNTER_CONFIG_i		Branch Monitoring Detect Counter Configuration (R/W)
		0		CntrEn Enable counter.
		7:1		CntrEvSel Event select (other values #GP) '0000000 = RETs. '0000001 = RET-CALL bias. '0000010 = RET mispredicts. '0000011 = Branch (all) mispredicts. '0000100 = Indirect branch mispredicts. '0000101 = Far branch instructions.
		14:8		CntrThreshold Threshold (an unsigned value of 0 to 127 supported). The value 0 of counter threshold will result in event signaled after every instruction. #GP if threshold is < 2.
		15		MispredEventCnt Mispredict events counting behavior: '0 = Mispredict events are counted in a window. '1 = Mispredict events are counted based on a consecutive occurrence. CntrThreshold is treated as # of consecutive mispredicts. This control bit only applies to events specified by CntrEvSel that involve a prediction (0000010, 0000011, 0000100). Setting this bit for other events is ignored.
		63:16		Reserved.

**Table 2-41. Additional MSRs Supported by Future Intel® Core™ Processors Based on Cannon Lake Microarchitecture**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Package C3 Residency Counter (R/O)
		63:0		Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
620H	1568	MSR_RING_RATIO_LIMIT	Package	Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.
		6:0		MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.
		7		Reserved.
		14:8		MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.
		63:15		Reserved.
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Core C1 Residency Counter (R/O)
		63:0		Value since last reset for the Core C1 residency. Counter rate is the Max Non-Turbo frequency (same as TSC). This counter count in case that both of the core's thread are in idle state and at least one of the core's thread residency in C1 state or in one of its sub state. The counter is updated only after core C state exit. Note: Always reads 0 if core C1 is unsupported. A value of zero indicates that this processor does not support core C1 or never entered core C1 level state.
662H	1634	MSR_CORE_C3_RESIDENCY	Core	Core C3 Residency Counter (R/O)
		63:0		Will always return 0.

### 2.16.3 MSRs Specific to Intel® Xeon® Processor Scalable Family

Intel® Xeon® Processor Scalable Family (CUID DisplayFamily\_DisplayModel = 06\_55H) support the MSRs listed in Table 2-42.

**Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	<b>Control Features in Intel 64 Processor (R/W)</b> See Table 2-2.
		0		<b>Lock (R/WL)</b>
		1		<b>Enable VMX inside SMX operation (R/WL)</b>
		2		<b>Enable VMX outside SMX operation (R/WL)</b>
		14:8		<b>SENTER local functions enables (R/WL)</b>
		15		<b>SENTER global functions enable (R/WL)</b>
		18		<b>SGX global functions enable (R/WL)</b>

**Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		20		<b>LMCE_ON (R/WL)</b>
		63:21		Reserved.
4EH	78	MSR_PPIN_CTL	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 2-25.
		1		Enable_PPIN (R/W) See Table 2-25.
		63:2		Reserved.
4FH	79	MSR_PPIN	Package	Protected Processor Inventory Number (R/O)
		63:0		<b>Protected Processor Inventory Number (R/O)</b> See Table 2-25.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> See Table 2-25.
		22:16		Reserved.
		23	Package	<b>PPIN_CAP (R/O)</b> See Table 2-25.
		27:24		Reserved.
		28	Package	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> See Table 2-25.
		29	Package	<b>Programmable TDP Limit for Turbo Mode (R/O)</b> See Table 2-25.
		30	Package	<b>Programmable TJ OFFSET (R/O)</b> See Table 2-25.
		39:31		Reserved.
		47:40	Package	<b>Maximum Efficiency Ratio (R/O)</b> See Table 2-25.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	<b>C-State Configuration Control (R/W)</b> Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See <a href="http://biosbits.org">http://biosbits.org</a> .

Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		<b>Package C-State Limit (R/W)</b> Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		<b>I/O MWAIT Redirection Enable (R/W)</b>
		14:11		Reserved
		15		<b>CFG Lock (R/W0)</b>
		16		<b>Automatic C-State Conversion Enable (R/W)</b> If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6)
		24:17		Reserved
		25		<b>C3 State Auto Demotion Enable (R/W)</b>
		26		<b>C1 State Auto Demotion Enable (R/W)</b>
		27		<b>Enable C3 Undemotion (R/W)</b>
		28		<b>Enable C1 Undemotion (R/W)</b>
		29		<b>Package C State Demotion Enable (R/W)</b>
		30		<b>Package C State UnDemotion Enable (R/W)</b>
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	<b>Global Machine Check Capability (R/O)</b>
		7:0		<b>Count</b>
		8		<b>MCG_CTL_P</b>
		9		<b>MCG_EXT_P</b>
		10		<b>MCP_CMCI_P</b>
		11		<b>MCG_TES_P</b>
		15:12		Reserved.
		23:16		<b>MCG_EXT_CNT</b>
		24		<b>MCG_SER_P</b>
		25		<b>MCG_EM_P</b>
		26		<b>MCG_ELOG_P</b>
		63:27		Reserved.

**Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17DH	390	MSR_SMM_MCA_CAP	THREAD	<b>Enhanced SMM Capabilities (SMM-RO)</b> Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		<b>Reserved</b>
		58		<b>SMM_Code_Access_Chk (SMM-RO)</b> If set to 1 indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		<b>Long_Flow_Indication (SMM-RO)</b> If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
19CH	412	IA32_THERM_STATUS	Core	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.
		0		<b>Thermal status (RO)</b> See Table 2-2.
		1		<b>Thermal status log (R/WCO)</b> See Table 2-2.
		2		<b>PROTCHOT # or FORCEPR# status (RO)</b> See Table 2-2.
		3		<b>PROTCHOT # or FORCEPR# log (R/WCO)</b> See Table 2-2.
		4		<b>Critical Temperature status (RO)</b> See Table 2-2.
		5		<b>Critical Temperature status log (R/WCO)</b> See Table 2-2.
		6		<b>Thermal threshold #1 status (RO)</b> See Table 2-2.
		7		<b>Thermal threshold #1 log (R/WCO)</b> See Table 2-2.
		8		<b>Thermal threshold #2 status (RO)</b> See Table 2-2.
		9		<b>Thermal threshold #2 log (R/WCO)</b> See Table 2-2.
		10		<b>Power Limitation status (RO)</b> See Table 2-2.
		11		<b>Power Limitation log (R/WCO)</b> See Table 2-2.
12		<b>Current Limit status (RO)</b> See Table 2-2.		



Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		13		<b>Current Limit log (R/WC0)</b> See Table 2-2.
		14		<b>Cross Domain Limit status (RO)</b> See Table 2-2.
		15		<b>Cross Domain Limit log (R/WC0)</b> See Table 2-2.
		22:16		<b>Digital Readout (RO)</b> See Table 2-2.
		26:23		Reserved.
		30:27		<b>Resolution in degrees Celsius (RO)</b> See Table 2-2.
		31		<b>Reading Valid (RO)</b> See Table 2-2.
		63:32		Reserved.
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved.
		23:16		<b>Temperature Target (RO)</b> See Table 2-25.
		27:24		<b>TCC Activation Offset (R/W)</b> See Table 2-25.
		63:28		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	This register defines the ratio limits. RATIO[0:7] must be populated in ascending order. RATIO[i+1] must be less than or equal to RATIO[i]. Entries with RATIO[i] will be ignored. If any of the rules above are broken, the configuration is silently rejected. If the programmed ratio is: <ul style="list-style-type: none"> <li>Above the fused ratio for that core count, it will be clipped to the fuse limits (assuming !OC).</li> <li>Below the min supported ratio, it will be clipped.</li> </ul>
		7:0		RATIO_0 Defines ratio limits.
		15:8		RATIO_1 Defines ratio limits.
		23:16		RATIO_2 Defines ratio limits.
		31:24		RATIO_3 Defines ratio limits.
		39:32		RATIO_4 Defines ratio limits.

**Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		47:40		RATIO_5 Defines ratio limits.
		55:48		RATIO_6 Defines ratio limits.
		63:56		RATIO_7 Defines ratio limits.
1AEH	430	MSR_TURBO_RATIO_LIMIT_CORES	Package	This register defines the active core ranges for each frequency point. NUMCORE[0:7] must be populated in ascending order. NUMCORE[i+1] must be greater than NUMCORE[i]. Entries with NUMCORE[i] == 0 will be ignored. The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, the configuration is silently rejected.
		7:0		NUMCORE_0 Defines the active core ranges for each frequency point.
		15:8		NUMCORE_1 Defines the active core ranges for each frequency point.
		23:16		NUMCORE_2 Defines the active core ranges for each frequency point.
		31:24		NUMCORE_3 Defines the active core ranges for each frequency point.
		39:32		NUMCORE_4 Defines the active core ranges for each frequency point.
		47:40		NUMCORE_5 Defines the active core ranges for each frequency point.
		55:48		NUMCORE_6 Defines the active core ranges for each frequency point.
		63:56		NUMCORE_7 Defines the active core ranges for each frequency point.
280H	640	IA32_MC0_CTL2	Core	See Table 2-2.
281H	641	IA32_MC1_CTL2	Core	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	See Table 2-2.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.

**Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC0 reports MC error from the IFU module.
401H	1025	IA32_MC0_STATUS	Core	
402H	1026	IA32_MC0_ADDR	Core	
403H	1027	IA32_MC0_MISC	Core	
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC error from the DCU module.
405H	1029	IA32_MC1_STATUS	Core	
406H	1030	IA32_MC1_ADDR	Core	
407H	1031	IA32_MC1_MISC	Core	
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC error from the DTLB module.
409H	1033	IA32_MC2_STATUS	Core	
40AH	1034	IA32_MC2_ADDR	Core	
40BH	1035	IA32_MC2_MISC	Core	
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC error from the MLC module.
40DH	1037	IA32_MC3_STATUS	Core	
40EH	1038	IA32_MC3_ADDR	Core	
40FH	1039	IA32_MC3_MISC	Core	
410H	1040	IA32_MC4_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC error from the PCU module.
411H	1041	IA32_MC4_STATUS	Package	
412H	1042	IA32_MC4_ADDR	Package	
413H	1043	IA32_MC4_MISC	Package	
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC error from a link interconnect module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC error from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	

**Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC error from the M2M 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC error from the M2M 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC error from the CHA
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC error from the CHA.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC error from the CHA.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC error from each channel of a link interconnect module.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC error from the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC error from the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC error from the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	

Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC error from the integrated memory controllers
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC error from the integrated memory controllers.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC error from the integrated memory controllers.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC error from a link interconnect module.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	<b>Unit Multipliers used in RAPL Interfaces (R/O)</b>
		3:0	Package	<b>Power Units</b> See Section 14.9.1, "RAPL Interfaces."
		7:4	Package	<b>Reserved</b>
		12:8	Package	<b>Energy Status Units</b> Energy related information (in Joules) is based on the multiplier, $1/2^{\wedge}ESU$ ; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)
		15:13	Package	<b>Reserved</b>
		19:16	Package	<b>Time Units</b> See Section 14.9.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	<b>DRAM RAPL Power Limit Control (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	<b>DRAM Energy Status (R/O)</b> Energy consumed by DRAM devices
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	<b>DRAM Performance Throttling Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	<b>DRAM RAPL Parameters (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."

**Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved.
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved.
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>Reserved (R/O)</b> Reads return 0
C8DH	3213	IA32_QM_EVTSEL	THREAD	<b>Monitoring Event Select Register (R/W)</b> if CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1
		7:0		<b>EventID (RW)</b> <b>Event encoding:</b> <b>0x00: no monitoring</b> <b>0x01: L3 occupancy monitoring</b> <b>0x02: Total memory bandwidth monitoring</b> <b>0x03: Local memory bandwidth monitoring</b> <b>All other encoding reserved</b>
		31:8		Reserved.
		41:32		<b>RMID (RW)</b>
		63:42		Reserved.
C8FH	3215	IA32_PQR_ASSOC	THREAD	<b>Resource Association Register (R/W)</b>
		9:0		<b>RMID</b>
		31:10		<b>Reserved</b>
		51:32		<b>COS (R/W).</b>
		63: 52		<b>Reserved</b>
C90H	3216	IA32_L3_QOS_MASK_0	Package	<b>L3 Class Of Service Mask - COS 0 (R/W)</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0
		0:19		<b>CBM: Bit vector of available L3 ways for COS 0 enforcement</b>
		63:20		<b>Reserved</b>
C91H	3217	IA32_L3_QOS_MASK_1	Package	<b>L3 Class Of Service Mask - COS 1 (R/W)</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1
		0:19		<b>CBM: Bit vector of available L3 ways for COS 1 enforcement</b>
		63:20		<b>Reserved</b>

Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C92H	3218	IA32_L3_QOS_MASK_2	Package	<b>L3 Class Of Service Mask - COS 2 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2
		0:19		<b>CBM: Bit vector of available L3 ways for COS 2 enforcement</b>
		63:20		<b>Reserved</b>
C93H	3219	IA32_L3_QOS_MASK_3	Package	<b>L3 Class Of Service Mask - COS 3 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3
		0:19		<b>CBM: Bit vector of available L3 ways for COS 3 enforcement</b>
		63:20		<b>Reserved</b>
C94H	3220	IA32_L3_QOS_MASK_4	Package	<b>L3 Class Of Service Mask - COS 4 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4
		0:19		<b>CBM: Bit vector of available L3 ways for COS 4 enforcement</b>
		63:20		<b>Reserved</b>
C95H	3221	IA32_L3_QOS_MASK_5	Package	<b>L3 Class Of Service Mask - COS 5 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5
		0:19		<b>CBM: Bit vector of available L3 ways for COS 5 enforcement</b>
		63:20		<b>Reserved</b>
C96H	3222	IA32_L3_QOS_MASK_6	Package	<b>L3 Class Of Service Mask - COS 6 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6
		0:19		<b>CBM: Bit vector of available L3 ways for COS 6 enforcement</b>
		63:20		<b>Reserved</b>
C97H	3223	IA32_L3_QOS_MASK_7	Package	<b>L3 Class Of Service Mask - COS 7 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7
		0:19		<b>CBM: Bit vector of available L3 ways for COS 7 enforcement</b>
		63:20		<b>Reserved</b>
C98H	3224	IA32_L3_QOS_MASK_8	Package	<b>L3 Class Of Service Mask - COS 8 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8
		0:19		<b>CBM: Bit vector of available L3 ways for COS 8 enforcement</b>
		63:20		<b>Reserved</b>
C99H	3225	IA32_L3_QOS_MASK_9	Package	<b>L3 Class Of Service Mask - COS 9 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9
		0:19		<b>CBM: Bit vector of available L3 ways for COS 9 enforcement</b>
		63:20		<b>Reserved</b>
C9AH	3226	IA32_L3_QOS_MASK_10	Package	<b>L3 Class Of Service Mask - COS 10 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=10
		0:19		<b>CBM: Bit vector of available L3 ways for COS 10 enforcement</b>
		63:20		<b>Reserved</b>

**Table 2-42. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily\_DisplayModel 06\_55H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C9BH	3227	IA32_L3_QOS_MASK_11	Package	<b>L3 Class Of Service Mask - COS 11 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11
		0:19		<b>CBM: Bit vector of available L3 ways for COS 11 enforcement</b>
		63:20		<b>Reserved</b>
C9CH	3228	IA32_L3_QOS_MASK_12	Package	<b>L3 Class Of Service Mask - COS 12 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=12
		0:19		<b>CBM: Bit vector of available L3 ways for COS 12 enforcement</b>
		63:20		<b>Reserved</b>
C9DH	3229	IA32_L3_QOS_MASK_13	Package	<b>L3 Class Of Service Mask - COS 13 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=13
		0:19		<b>CBM: Bit vector of available L3 ways for COS 13 enforcement</b>
		63:20		<b>Reserved</b>
C9EH	3230	IA32_L3_QOS_MASK_14	Package	<b>L3 Class Of Service Mask - COS 14 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=14
		0:19		<b>CBM: Bit vector of available L3 ways for COS 14 enforcement</b>
		63:20		<b>Reserved</b>
C9FH	3231	IA32_L3_QOS_MASK_15	Package	<b>L3 Class Of Service Mask - COS 15 (R/W).</b> if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=15
		0:19		<b>CBM: Bit vector of available L3 ways for COS 15 enforcement</b>
		63:20		<b>Reserved</b>

## 2.17 MSRS IN INTEL® XEON PHI™ PROCESSOR 3200/5200/7200 SERIES AND FUTURE INTEL® XEON PHI™ PROCESSOR

Intel® Xeon Phi™ processor 3200, 5200, 7200 series, with CPUID DisplayFamily\_DisplayModel signature 06\_57H, supports the MSR interfaces listed in Table 2-43. These processors are based on the Knights Landing microarchitecture. Future Intel® Xeon Phi™ Processor, with CPUID DisplayFamily\_DisplayModel signature 06\_85H, supports the MSR interfaces listed in Table 2-43 and Table 2-44. Some MSRs are shared between a pair of processor cores, the scope is marked as module.

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 2.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 2.22, "MSRs in Pentium Processors."



**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	<b>Platform ID (R)</b> See Table 2-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	<b>SMI Counter (R/O)</b>
		31:0		<b>SMI Count (R/O)</b>
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	<b>Control Features in Intel 64 Processor (R/W)</b> See Table 2-2.
		0		<b>Lock (R/WL)</b>
		1		<b>Reserved</b>
		2		<b>Enable VMX outside SMX operation (R/WL)</b>
3BH	59	IA32_TSC_ADJUST	THREAD	<b>Per-Logical-Processor TSC ADJUST (R/W)</b> See Table 2-2.
4EH	78	MSR_PPIN_CTL	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) Set 1 to prevent further writes to MSR_PPIN_CTL. Writing 1 to MSR_PPIN_CTL[bit 0] is permitted only if MSR_PPIN_CTL[bit 1] is clear, Default is 0. BIOS should provide an opt-in menu to enable the user to turn on MSR_PPIN_CTL[bit 1] for privileged inventory initialization agent to access MSR_PPIN. After reading MSR_PPIN, the privileged inventory initialization agent should write '01b' to MSR_PPIN_CTL to disable further access to MSR_PPIN and prevent unauthorized modification to MSR_PPIN_CTL.
		1		Enable_PPIN (R/W) If 1, enables MSR_PPIN to be accessible using RDMSR. Once set, attempt to write 1 to MSR_PPIN_CTL[bit 0] will cause #GP. If 0, an attempt to read MSR_PPIN will cause #GP. Default is 0.
		63:2		Reserved.
4FH	79	MSR_PPIN	Package	Protected Processor Inventory Number (R/O)
		63:0		<b>Protected Processor Inventory Number (R/O)</b> A unique value within a given CPUID family/model/stepping signature that a privileged inventory initialization agent can access to identify each physical processor, when access to MSR_PPIN is enabled. Access to MSR_PPIN is permitted only if MSR_PPIN_CTL[bits 1:0] = '10b'

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
79H	121	IA32_BIOS_UPDT_TRIG	Core	<b>BIOS Update Trigger Register (W)</b> See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	THREAD	<b>BIOS Update Signature ID (R0)</b> See Table 2-2.
C1H	193	IA32_PMC0	THREAD	<b>Performance counter register</b> See Table 2-2.
C2H	194	IA32_PMC1	THREAD	<b>Performance Counter Register</b> See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information; contains power management and other model specific features enumeration. See <a href="http://biosbits.org">http://biosbits.org</a> .
		7:0		Reserved.
		15:8	Package	<b>Maximum Non-Turbo Ratio (R/O)</b> The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	<b>Programmable Ratio Limit for Turbo Mode (R/O)</b> 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	<b>Programmable TDP Limit for Turbo Mode (R/O)</b> 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	<b>Maximum Efficiency Ratio (R/O)</b> The is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Package	<b>C-State Configuration Control (R/W)</b>

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		<p><b>Package C-State Limit (R/W)</b></p> <p>Specifies the lowest C-state for the package. This feature does not limit the processor core C-state. The power-on default value from bit[2:0] of this register reports the deepest package C-state the processor is capable to support when manufactured. It is recommended that BIOS always read the power-on default value reported from this bit field to determine the supported deepest C-state on the processor and leave it as default without changing it.</p> <p>000b - C0/C1 (No package C-state support)  001b - C2  010b - C6 (non retention)*  011b - C6 (Retention)*  100b - Reserved  101b - Reserved  110b - Reserved  111b - No package C-state limit. All C-States supported by the processor are available.</p> <p>Note: C6 retention mode provides more power saving than C6 non-retention mode. Limiting the package to C6 non retention mode does prevent the MSR_PKG_C6_RESIDENCY counter (MSR 3F9h) from being incremented.</p>
		9:3		Reserved.
		10		<p><b>I/O MWAIT Redirection Enable (R/W)</b></p> <p>When set, will map IO_read instructions sent to IO registers at MSR_PMG_IO_CAPTURE_BASE[15:0] to MWAIT instructions.</p>
		14:11		Reserved.
		15		<p><b>CFG Lock (RO)</b></p> <p>When set, locks bits [15:0] of this register for further writes until the next reset occurs.</p>
		25		Reserved.
		26		<p><b>C1 State Auto Demotion Enable (R/W)</b></p> <p>When set, processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.</p>
		27		Reserved.
		28		<p><b>C1 State Auto Undemotion Enable (R/W)</b></p> <p>When set, enables Undemotion from Demoted C1.</p>
		29		<p><b>PKG C-State Auto Demotion Enable (R/W)</b></p> <p>When set, enables Package C state demotion.</p>
		63:30		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Tile	<b>Power Management IO Capture Base (R/W)</b>

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:0		<b>LVL_2 Base Address (R/W)</b> Microcode will compare IO-read zone to this base address to determine if an MWAIT(C2/3/4) needs to be issued instead of the IO-read. Should be programmed to the chipset Plevel_2 IO address.
		22:16		<b>C-State Range (R/W)</b> The IO-port block size in which IO-redirection will be executed (0-127). Should be programmed based on the number of LVLx registers existing in the chipset.
		63:23		Reserved.
E7H	231	IA32_MPERF	Thread	<b>Maximum Performance Frequency Clock Count (RW)</b> See Table 2-2.
E8H	232	IA32_APERF	Thread	<b>Actual Performance Frequency Clock Count (RW)</b> See Table 2-2.
FEH	254	IA32_MTRRCAP	Core	<b>Memory Type Range Register (R)</b> See Table 2-2.
13CH	52	MSR_FEATURE_CONFIG	Core	<b>AES Configuration (RW-L)</b> Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		<b>AES Configuration (RW-L)</b> Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
140H	320	MISC_FEATURE_ENABLES	Thread	<b>MISC_FEATURE_ENABLES</b>
		0		Reserved.
		1		<b>User Mode MONITOR and MWAIT (R/W)</b> If set to 1, the MONITOR and MWAIT instructions do not cause invalid-opcode exceptions when executed with CPL > 0 or in virtual-8086 mode. If MWAIT is executed when CPL > 0 or in virtual-8086 mode, and if EAX indicates a C-state other than C0 or C1, the instruction operates as if EAX indicated the C-state C1.
		63:2		Reserved.
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	See Table 2-2.

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
17DH	390	MSR_SMM_MCA_CAP	Thread	<b>Enhanced SMM Capabilities (SMM-RO)</b> Reports SMM capability Enhancement. Accessible only while in SMM.
		31:0		<b>Bank Support (SMM-RO)</b> One bit per MCA bank. If the bit is set, that bank supports Enhanced MCA (Default all 0; does not support EMCA).
		55:32		Reserved.
		56		<b>Targeted SMI (SMM-RO)</b> Set if targeted SMI is supported.
		57		<b>SMM_CPU_SVRSTR (SMM-RO)</b> Set if SMM SRAM save/restore feature is supported.
		58		<b>SMM_CODE_ACCESS_CHK (SMM-RO)</b> Set if SMM code access check feature is supported.
		59		<b>Long_Flow_Indication (SMM-RO)</b> If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved.
186H	390	IA32_PERFEVTSELO	Thread	Performance Monitoring Event Select Register (R/W) See Table 2-2.
		7:0		<b>Event Select</b>
		15:8		<b>UMask</b>
		16		<b>USR</b>
		17		<b>OS</b>
		18		<b>Edge</b>
		19		<b>PC</b>
		20		<b>INT</b>
		21		<b>AnyThread</b>
		22		<b>EN</b>
		23		<b>INV</b>
		31:24		<b>CMASK</b>
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
198H	408	IA32_PERF_STATUS	Package	See Table 2-2.
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	<b>Clock Modulation (R/W)</b> See Table 2-2.
19BH	411	IA32_THERM_INTERRUPT	Module	<b>Thermal Interrupt Control (R/W)</b> See Table 2-2.

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
19CH	412	IA32_THERM_STATUS	Module	<b>Thermal Monitor Status (R/W)</b> See Table 2-2.
		0		<b>Thermal status (RO)</b>
		1		<b>Thermal status log (R/WCO)</b>
		2		<b>PROTCHOT # or FORCEPR# status (RO)</b>
		3		<b>PROTCHOT # or FORCEPR# log (R/WCO)</b>
		4		<b>Critical Temperature status (RO)</b>
		5		<b>Critical Temperature status log (R/WCO)</b>
		6		<b>Thermal threshold #1 status (RO)</b>
		7		<b>Thermal threshold #1 log (R/WCO)</b>
		8		<b>Thermal threshold #2 status (RO)</b>
		9		<b>Thermal threshold #2 log (R/WCO)</b>
		10		<b>Power Limitation status (RO)</b>
		11		<b>Power Limitation log (R/WCO)</b>
		15:12		Reserved.
		22:16		<b>Digital Readout (RO)</b>
		26:23		Reserved.
		30:27		<b>Resolution in degrees Celsius (RO)</b>
		31		<b>Reading Valid (RO)</b>
63:32		Reserved.		
1A0H	416	IA32_MISC_ENABLE	Thread	<b>Enable Misc. Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.
		0		<b>Fast-Strings Enable</b>
		2:1		Reserved.
		3		<b>Automatic Thermal Control Circuit Enable (R/W)</b>
		6:4		Reserved.
		7		<b>Performance Monitoring Available (R)</b>
		10:8		Reserved.
		11		<b>Branch Trace Storage Unavailable (RO)</b>
		12		<b>Processor Event Based Sampling Unavailable (RO)</b>
		15:13		Reserved.
		16		<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b>
		18		<b>ENABLE MONITOR FSM (R/W)</b>
		21:19		Reserved.
		22		<b>Limit CPUID Maxval (R/W)</b>
		23		<b>xTPR Message Disable (R/W)</b>
33:24		Reserved.		

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		34		<b>XD Bit Disable (R/W)</b>
		37:35		Reserved.
		38		<b>Turbo Mode Disable (R/W)</b>
		63:39		Reserved.
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved.
		23:16		<b>Temperature Target (R)</b>
		29:24		<b>Target Offset (R/W)</b>
		63:30		Reserved.
1A4H	420	MSR_MISC_FEATURE_CONTROL		<b>Miscellaneous Feature Control (R/W)</b>
		0	Core	<b>DCU Hardware Prefetcher Disable (R/W)</b> If 1, disables the L1 data cache prefetcher.
		1	Core	<b>L2 Hardware Prefetcher Disable (R/W)</b> If 1, disables the L2 hardware prefetcher.
		63:2		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Shared	<b>Offcore Response Event Select Register (R/W)</b>
1A7H	423	MSR_OFFCORE_RSP_1	Shared	<b>Offcore Response Event Select Register (R/W)</b>
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	<b>Maximum Ratio Limit of Turbo Mode for Groups of Cores (RW)</b>
		0		<b>Reserved</b>
		7:1	Package	<b>Maximum Number of Cores in Group 0</b> Number active processor cores which operates under the maximum ratio limit for group 0.
		15:8	Package	<b>Maximum Ratio Limit for Group 0</b> Maximum turbo ratio limit when the number of active cores are not more than the group 0 maximum core count.
		20:16	Package	<b>Number of Incremental Cores Added to Group 1</b> Group 1, which includes the specified number of additional cores plus the cores in group 0, operates under the group 1 turbo max ratio limit = "group 0 Max ratio limit" - "group ratio delta for group 1".
		23:21	Package	<b>Group Ratio Delta for Group 1</b> An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.
		28:24	Package	<b>Number of Incremental Cores Added to Group 2</b> Group 2, which includes the specified number of additional cores plus all the cores in group 1, operates under the group 2 turbo max ratio limit = "group 1 Max ratio limit" - "group ratio delta for group 2".

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:29	Package	<b>Group Ratio Delta for Group 2</b> An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.
		36:32	Package	<b>Number of Incremental Cores Added to Group 3</b> Group 3, which includes the specified number of additional cores plus all the cores in group 2, operates under the group 3 turbo max ratio limit = "group 2 Max ratio limit" - "group ratio delta for group 3".
		39:37	Package	<b>Group Ratio Delta for Group 3</b> An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.
		44:40	Package	<b>Number of Incremental Cores Added to Group 4</b> Group 4, which includes the specified number of additional cores plus all the cores in group 3, operates under the group 4 turbo max ratio limit = "group 3 Max ratio limit" - "group ratio delta for group 4".
		47:45	Package	<b>Group Ratio Delta for Group 4</b> An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.
		52:48	Package	<b>Number of Incremental Cores Added to Group 5</b> Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".
		55:53	Package	<b>Group Ratio Delta for Group 5</b> An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.
		60:56	Package	<b>Number of Incremental Cores Added to Group 6</b> Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".
		63:61	Package	<b>Group Ratio Delta for Group 6</b> An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.
1B0H	432	IA32_ENERGY_PERF_BIAS	Thread	See Table 2-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 2-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 2-2.
1C8H	456	MSR_LBR_SELECT	Thread	<b>Last Branch Record Filtering Select Register (R/W)</b> See Section 17.9.2, "Filtering of Last Branch Records."
		0		<b>CPL_EQ_0</b>
		1		<b>CPL_NEQ_0</b>



**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		<b>JCC</b>
		3		<b>NEAR_REL_CALL</b>
		4		<b>NEAR_IND_CALL</b>
		5		<b>NEAR_RET</b>
		6		<b>NEAR_IND_JMP</b>
		7		<b>NEAR_REL_JMP</b>
		8		<b>FAR_BRANCH</b>
		63:9		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Thread	<b>Last Branch Record Stack TOS (R/W)</b> Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.
1D9H	473	IA32_DEBUGCTL	Thread	<b>Debug Control (R/W)</b>
		0		<b>LBR</b> Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.
		1		<b>BTF</b> Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.
		5:2		Reserved.
		6		<b>TR</b> Setting this bit to 1 enables branch trace messages to be sent.
		7		<b>BTS</b> Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.
		8		<b>BTINT</b> When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.
		9		<b>BTS_OFF_OS</b> When set, BTS or BTM is skipped if CPL = 0.
		10		<b>BTS_OFF_USR</b> When set, BTS or BTM is skipped if CPL > 0.
		11		<b>FREEZE_LBRS_ON_PMI</b> When set, the LBR stack is frozen on a PMI request.
		12		<b>FREEZE_PERFMON_ON_PMI</b> When set, each ENABLE bit of the global counter control MSR are frozen (address 3BFH) on a PMI request.
		13		Reserved.

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		14		<b>FREEZE_WHILE_SMM_EN</b> When set, freezes perfmon and trace messages while in SMM.
		31:15		Reserved.
1DDH	477	MSR_LER_FROM_LIP	Thread	<b>Last Exception Record From Linear IP (R)</b>
1DEH	478	MSR_LER_TO_LIP	Thread	<b>Last Exception Record To Linear IP (R)</b>
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 2-2.
277H	631	IA32_PAT	Core	See Table 2-2.

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
2FFH	767	IA32_MTRR_DEF_TYPE	Core	<b>Default Memory Types (R/W)</b> See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	<b>Fixed-Function Performance Counter Register 0 (R/W)</b> See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	<b>Fixed-Function Performance Counter Register 1 (R/W)</b> See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	<b>Fixed-Function Performance Counter Register 2 (R/W)</b> See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Package	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Thread	<b>Fixed-Function-Counter Control Register (R/W)</b> See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATU S	Thread	See Table 2-2.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2.
390H	912	IA32_PERF_GLOBAL_OVF_ CTRL	Thread	See Table 2-2.
3F1H	1009	MSR_PEBBS_ENABLE	Thread	See Table 2-2.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O)
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	
		63:0		Package C6 Residency Counter. (R/O)
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	
		63:0		Package C7 Residency Counter. (R/O)
3FCH	1020	MSR_MC0_RESIDENCY	Module	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Module C0 Residency Counter. (R/O)
3FDH	1021	MSR_MC6_RESIDENCY	Module	
		63:0		Module C6 Residency Counter. (R/O)
3FFH	1023	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter. (R/O)
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
402H	1026	IA32_MC0_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
4C1H	1217	IA32_A_PMC0	Thread	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 2-2.
600H	1536	IA32_DS_AREA	Thread	<b>DS Save Area (R/W)</b> See Table 2-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	<b>Unit Multipliers used in RAPL Interfaces (R/O)</b>
		3:0	Package	<b>Power Units</b> See Section 14.9.1, "RAPL Interfaces."
		7:4	Package	<b>Reserved</b>
		12:8	Package	<b>Energy Status Units</b> Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)
		15:13	Package	<b>Reserved</b>
		19:16	Package	<b>Time Units</b> See Section 14.9.1, "RAPL Interfaces."
		63:20		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C2 Residency Counter. (R/O)
610H	1552	MSR_PKG_POWER_LIMIT	Package	<b>PKG RAPL Power Limit Control (R/W)</b> See Section 14.9.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	<b>PKG Energy Status (R/O)</b> See Section 14.9.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	<b>PKG Perf Status (R/O)</b> See Section 14.9.3, "Package RAPL Domain."

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description	
Hex	Dec				
614H	1556	MSR_PKG_POWER_INFO	Package	<b>PKG RAPL Parameters (R/W)</b> See Section 14.9.3, "Package RAPL Domain."	
618H	1560	MSR_DRAM_POWER_LIMIT	Package	<b>DRAM RAPL Power Limit Control (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."	
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	<b>DRAM Energy Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."	
61BH	1563	MSR_DRAM_PERF_STATUS	Package	<b>DRAM Performance Throttling Status (R/O)</b> See Section 14.9.5, "DRAM RAPL Domain."	
61CH	1564	MSR_DRAM_POWER_INFO	Package	<b>DRAM RAPL Parameters (R/W)</b> See Section 14.9.5, "DRAM RAPL Domain."	
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.	
				63:15	Reserved.
				14:8	<b>MIN_RATIO</b> Writing to this field controls the minimum possible ratio of the LLC/Ring.
				7	Reserved.
				6:0	<b>MAX_RATIO</b> This field is used to limit the max ratio of the LLC/Ring.
638H	1592	MSR_PPO_POWER_LIMIT	Package	<b>PPO RAPL Power Limit Control (R/W)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."	
639H	1593	MSR_PPO_ENERGY_STATUS	Package	<b>PPO Energy Status (R/O)</b> See Section 14.9.4, "PPO/PP1 RAPL Domains."	
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	<b>Base TDP Ratio (R/O)</b> See Table 2-24	
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O). See Table 2-24	
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O). See Table 2-24	
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	<b>ConfigTDP Control (R/W)</b> See Table 2-24	
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	<b>ConfigTDP Control (R/W)</b> See Table 2-24	
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	<b>Indicator of Frequency Clipping in Processor Cores (R/W)</b> (frequency refers to processor core frequency)	
				0	<b>PROCHOT Status (R/O)</b>
				1	<b>Thermal Status (R/O)</b>
				5:2	Reserved.
				6	<b>VR Therm Alert Status (R/O)</b>

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
		7		Reserved.
		8		<b>Electrical Design Point Status (R0)</b>
		63:9		Reserved.
6E0H	1760	IA32_TSC_DEADLINE	Core	<b>TSC Target of Local APIC's TSC Deadline Mode (R/W)</b> See Table 2-2
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)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status register (R/W)

**Table 2-43. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signatures 06\_57H and 06\_85H**

Address		Register Name	Scope	Bit Description
Hex	Dec			
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	<b>Extended Feature Enables</b> See Table 2-2.
C000_0081H		IA32_STAR	Thread	<b>System Call Target Address (R/W)</b> See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	<b>IA-32e Mode System Call Target Address (R/W)</b> See Table 2-2.
C000_0084H		IA32_FMASK	Thread	<b>System Call Flag Mask (R/W)</b> See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	<b>Map of BASE Address of FS (R/W)</b> See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	<b>Map of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	<b>Swap Target of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	<b>AUXILIARY TSC Signature. (R/W)</b> See Table 2-2

Table 2-44 lists model-specific registers that are supported by future Intel® Xeon Phi™ Processors based on the Knights Mill microarchitecture.

**Table 2-44. Additional MSRs Supported by Future Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signature 06\_85H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
9BH	155	IA32_SMM_MONITOR_CTL	Core	SMM Monitor Configuration (R/W). This MSR is readable only if VMX is enabled, and writeable only if VMX is enabled and in SMM mode, and is used to configure the VMX MSEG base address. See Table 2-2.
480H	1152	IA32_VMX_BASIC	Core	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2.
481H	1153	IA32_VMX_PINBASED_CTL	Core	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2.
482H	1154	IA32_VMX_PROCBASED_CTL	Core	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)
483H	1155	IA32_VMX_EXIT_CTL	Core	Capability Reporting Register of VM-exit Controls (R/O) See Table 2-2.
484H	1156	IA32_VMX_ENTRY_CTL	Core	Capability Reporting Register of VM-entry Controls (R/O) See Table 2-2.
485H	1157	IA32_VMX_MISC	Core	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2.
486H	1158	IA32_VMX_CRO_FIXED0	Core	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2.
487H	1159	IA32_VMX_CRO_FIXED1	Core	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2.
488H	1160	IA32_VMX_CR4_FIXED0	Core	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2.
489H	1161	IA32_VMX_CR4_FIXED1	Core	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2.
48AH	1162	IA32_VMX_VMCS_ENUM	Core	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2.
48BH	1163	IA32_VMX_PROCBASED_CTL2	Core	Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O) See Table 2-2.
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.
48DH	1165	IA32_VMX_TRUE_PINBASED_CTL	Core	Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O) See Table 2-2.
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTL	Core	Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O) See Table 2-2.
48FH	1167	IA32_VMX_TRUE_EXIT_CTL	Core	Capability Reporting Register of VM-exit Flex Controls (R/O) See Table 2-2.



**Table 2-44. Additional MSRs Supported by Future Intel® Xeon Phi™ Processors with DisplayFamily\_DisplayModel Signature 06\_85H**

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
490H	1168	IA32_VMX_TRUE_ENTRY_C TLS	Core	Capability Reporting Register of VM-entry Flex Controls (R/O) See Table 2-2.
491H	1169	IA32_VMX_FMFUNC	Core	Capability Reporting Register of VM-function Controls (R/O) See Table 2-2.

## 2.18 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 2-45 lists MSRs (architectural and model-specific) that are defined across processor generations based on Intel NetBurst microarchitecture. The processor can be identified by its CPUID signatures of DisplayFamily encoding of 0FH, see Table 2-1.

- MSRs with an "IA32\_" prefix are designated as "architectural." This means that the functions of these MSRs and their addresses remain the same for succeeding families of IA-32 processors.
- MSRs with an "MSR\_" prefix are model specific with respect to address functionalities. The column "Model Availability" lists the model encoding value(s) within the Pentium 4 and Intel Xeon processor family at the specified register address. The model encoding value of a processor can be queried using CPUID. See "CPUID—CPU Identification" in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
0H	0	IA32_P5_MC_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 2.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3, 4, 6	Shared	See Section 2.22, "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	<b>Time Stamp Counter</b> See Table 2-2.
					On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.
17H	23	IA32_PLATFORM_ID	0, 1, 2, 3, 4, 6	Shared	<b>Platform ID (R)</b> See Table 2-2.  The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	0, 1, 2, 3, 4, 6	Unique	<b>APIC Location and Status (R/W)</b> See Table 2-2. See Section 10.4.4, "Local APIC Status and Location."

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
2AH	42	MSR_EBC_HARD_POWERON	0, 1, 2, 3, 4, 6	Shared	<b>Processor Hard Power-On Configuration (R/W)</b> Enables and disables processor features; (R) indicates current processor configuration.
		0			<b>Output Tri-state Enabled (R)</b> 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			<b>Execute BIST (R)</b> 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			<b>In Order Queue Depth (R)</b> 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			<b>MCERR# Observation Disabled (R)</b> 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			<b>BINIT# Observation Enabled (R)</b> 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			<b>APIC Cluster ID (R)</b> Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		7			<b>Bus Park Disable (R)</b> 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.

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
		13:12			<b>Agent ID (R)</b> 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	<b>Processor Soft Power-On Configuration (R/W)</b> Enables and disables processor features.
		0			<b>RCNT/SCNT On Request Encoding Enable (R/W)</b> Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).
		1			<b>Data Error Checking Disable (R/W)</b> Set to disable system data bus parity checking; clear to enable parity checking.
		2			<b>Response Error Checking Disable (R/W)</b> Set to disable (default); clear to enable.
		3			<b>Address/Request Error Checking Disable (R/W)</b> Set to disable (default); clear to enable.
		4			<b>Initiator MCERR# Disable (R/W)</b> Set to disable MCERR# driving for initiator bus requests (default); clear to enable.
		5			<b>Internal MCERR# Disable (R/W)</b> Set to disable MCERR# driving for initiator internal errors (default); clear to enable.
		6			<b>BINIT# Driver Disable (R/W)</b> Set to disable BINIT# driver (default); clear to enable driver.
		63:7			Reserved.
2CH	44	MSR_EBC_FREQUENCY_ID	2,3, 4, 6	Shared	<b>Processor Frequency Configuration</b> The bit field layout of this MSR varies according to the MODEL value in the CPUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2.  (R) The field Indicates the current processor frequency configuration.
		15:0			Reserved.

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
		18:16			<b>Scalable Bus Speed (R/W)</b> Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6)
					133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
					266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.
		23:19			Reserved.
		31:24			<b>Core Clock Frequency to System Bus Frequency Ratio (R)</b> The processor core clock frequency to system bus frequency ratio observed at the de-assertion of the reset pin.
		63:25			Reserved.
		2CH	44	MSR_EBC_FREQUENCY_ID	0, 1
		20:0			Reserved.
		23:21			<b>Scalable Bus Speed (R/W)</b> Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz All others values reserved.
		63:24			Reserved.

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
3AH	58	IA32_FEATURE_CONTROL	3, 4, 6	Unique	<b>Control Features in IA-32 Processor (R/W)</b> See Table 2-2 (If CPUID.01H:ECX.[bit 5])
79H	121	IA32_BIOS_UPDT_TRIG	0, 1, 2, 3, 4, 6	Shared	<b>BIOS Update Trigger Register (W)</b> See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3, 4, 6	Unique	<b>BIOS Update Signature ID (R/W)</b> See Table 2-2.
9BH	155	IA32_SMM_MONITOR_CTL	3, 4, 6	Unique	<b>SMM Monitor Configuration (R/W)</b> See Table 2-2.
FEH	254	IA32_MTRRCAP	0, 1, 2, 3, 4, 6	Unique	<b>MTRR Information</b> See Section 11.11.1, "MTRR Feature Identification."
174H	372	IA32_SYSENTER_CS	0, 1, 2, 3, 4, 6	Unique	<b>CS register target for CPL 0 code (R/W)</b> See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
175H	373	IA32_SYSENTER_ESP	0, 1, 2, 3, 4, 6	Unique	<b>Stack pointer for CPL 0 stack (R/W)</b> See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3, 4, 6	Unique	<b>CPL 0 code entry point (R/W)</b> See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
179H	377	IA32_MCG_CAP	0, 1, 2, 3, 4, 6	Unique	<b>Machine Check Capabilities (R)</b> See Table 2-2. See Section 15.3.1.1, "IA32_MCG_CAP MSR."
17AH	378	IA32_MCG_STATUS	0, 1, 2, 3, 4, 6	Unique	<b>Machine Check Status. (R)</b> See Table 2-2. See Section 15.3.1.2, "IA32_MCG_STATUS MSR."
17BH	379	IA32_MCG_CTL			<b>Machine Check Feature Enable (R/W)</b> See Table 2-2. See Section 15.3.1.3, "IA32_MCG_CTL MSR."
180H	384	MSR_MCG_RAX	0, 1, 2, 3, 4, 6	Unique	<b>Machine Check EAX/RAX Save State</b> See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
181H	385	MSR_MCG_RBX	0, 1, 2, 3, 4, 6	Unique	<b>Machine Check EBX/RBX Save State</b> 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	<b>Machine Check ECX/RCX Save State</b> 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	<b>Machine Check EDX/RDX Save State</b> 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	<b>Machine Check ESI/RSI Save State</b> 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.
185H	389	MSR_MCG_RDI	0, 1, 2, 3, 4, 6	Unique	<b>Machine Check EDI/RDI Save State</b> 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	<b>Machine Check EBP/RBP Save State</b> 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	<b>Machine Check ESP/RSP Save State</b> 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	<b>Machine Check EFLAGS/RFLAG Save State</b> See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	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.
189H	393	MSR_MCG_RIP	0, 1, 2, 3, 4, 6	Unique	<b>Machine Check EIP/RIP Save State</b> 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	<b>Machine Check Miscellaneous</b> See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		0			<b>DS</b> 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	<b>Machine Check R8</b> 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	<b>Machine Check R9D/R9</b> 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	<b>Machine Check R10</b> See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
193H	403	MSR_MCG_R11	0, 1, 2, 3, 4, 6	Unique	<b>Machine Check R11</b> See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
194H	404	MSR_MCG_R12	0, 1, 2, 3, 4, 6	Unique	<b>Machine Check R12</b> 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	<b>Machine Check R13</b> 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	<b>Machine Check R14</b> 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	<b>Machine Check R15</b> See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
198H	408	IA32_PERF_STATUS	3, 4, 6	Unique	See Table 2-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."



Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
199H	409	IA32_PERF_CTL	3, 4, 6	Unique	See Table 2-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."
19AH	410	IA32_CLOCK_MODULATION	0, 1, 2, 3, 4, 6	Unique	<b>Thermal Monitor Control (R/W)</b> See Table 2-2. See Section 14.7.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	0, 1, 2, 3, 4, 6	Unique	<b>Thermal Interrupt Control (R/W)</b> See Section 14.7.2, "Thermal Monitor," and see Table 2-2.
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3, 4, 6	Shared	<b>Thermal Monitor Status (R/W)</b> See Section 14.7.2, "Thermal Monitor," and see Table 2-2.
19DH	413	MSR_THERM2_CTL			Thermal Monitor 2 Control.
			3,	Shared	For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.
			4, 6	Shared	For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.
1A0H	416	IA32_MISC_ENABLE	0, 1, 2, 3, 4, 6	Shared	<b>Enable Miscellaneous Processor Features (R/W)</b>
		0			Fast-Strings Enable. See Table 2-2.
		1			Reserved.
		2			<b>x87 FPU Fopcode Compatibility Mode Enable</b>
		3			<b>Thermal Monitor 1 Enable</b> See Section 14.7.2, "Thermal Monitor," and see Table 2-2.
		4			<b>Split-Lock Disable</b> 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.

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique <sup>7</sup>	Bit Description
Hex	Dec				
		6			<p><b>Third-Level Cache Disable (R/W)</b>                      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.</p> <p>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."</p>
		7			<p><b>Performance Monitoring Available (R)</b>                      See Table 2-2.</p>
		8			<p><b>Suppress Lock Enable</b>                      When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.</p>
		9			<p><b>Prefetch Queue Disable</b>                      When set, disables the prefetch queue. When clear (default), enables the prefetch queue.</p>
		10			<p><b>FERR# Interrupt Reporting Enable (R/W)</b>                      When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled.</p>
					<p>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.</p>
					<p>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.</p>
		11			<p><b>Branch Trace Storage Unavailable (BTS_UNAVAILABLE) (R)</b>                      See Table 2-2.                      When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.</p>
		12			<p><b>PEBS_UNAVAILABLE: Processor Event Based Sampling Unavailable (R)</b>                      See Table 2-2.                      When set, the processor does not support processor event-based sampling (PEBS); when clear, PEBS is supported.</p>

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique <sup>1</sup>	Bit Description
Hex	Dec				
		13	3		<p><b>TM2 Enable (R/W)</b></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> <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.</p>
		17:14			Reserved.
		18	3, 4, 6		<p><b>ENABLE MONITOR FSM (R/W)</b></p> <p>See Table 2-2.</p>
		19			<p><b>Adjacent Cache Line Prefetch Disable (R/W)</b></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><b>Limit CPUID MAXVAL (R/W)</b></p> <p>See Table 2-2.</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><b>xTPR Message Disable (R/W)</b></p> <p>See Table 2-2.</p>

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
		24			<p><b>L1 Data Cache Context Mode (R/W)</b> 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><b>XD Bit Disable (R/W)</b> See Table 2-2.</p>
		63:35			Reserved.
1A1H	417	MSR_PLATFORM_BRV	3, 4, 6	Shared	<b>Platform Feature Requirements (R)</b>
		17:0			Reserved.
		18			<p><b>PLATFORM Requirements</b> 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><b>Last Exception Record From Linear IP (R)</b> Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.13.3, "Last Exception Records."</p>
		31:0			<p><b>From Linear IP</b> Linear address of the last branch instruction.</p>
		63:32			Reserved.
1D7H	471	63:0		Unique	<p><b>From Linear IP</b> Linear address of the last branch instruction (If IA-32e mode is active).</p>
1D8H	472	MSR_LER_TO_LIP	0, 1, 2, 3, 4, 6	Unique	<p><b>Last Exception Record To Linear IP (R)</b> This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.13.3, "Last Exception Records."</p>

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
		31:0			<b>From Linear IP</b> Linear address of the target of the last branch instruction.
		63:32			Reserved.
1D8H	472	63:0		Unique	<b>From Linear IP</b> 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	<b>Debug Control (R/W)</b> Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 17.13.1, "MSR_DEBUGCTLA MSR."
1DAH	474	MSR_LASTBRANCH_TOS	0, 1, 2, 3, 4, 6	Unique	<b>Last Branch Record Stack TOS (R/O)</b> Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 17.13.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture"; and addresses 1DBH-1DEH and 680H-68FH.
1DBH	475	MSR_LASTBRANCH_0	0, 1, 2	Unique	<b>Last Branch Record 0 (R/O)</b> One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH. See Section 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
1DCH	477	MSR_LASTBRANCH_1	0, 1, 2	Unique	<b>Last Branch Record 1</b> See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DDH	477	MSR_LASTBRANCH_2	0, 1, 2	Unique	<b>Last Branch Record 2</b> See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DEH	478	MSR_LASTBRANCH_3	0, 1, 2	Unique	<b>Last Branch Record 3</b> See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
200H	512	IA32_MTRR_PHYSBASE0	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Base MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
201H	513	IA32_MTRR_PHYSMASK0	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
202H	514	IA32_MTRR_PHYSBASE1	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
203H	515	IA32_MTRR_PHYSMASK1	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
204H	516	IA32_MTRR_PHYSBASE2	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
205H	517	IA32_MTRR_PHYSMASK2	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
206H	518	IA32_MTRR_PHYSBASE3	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
207H	519	IA32_MTRR_PHYSMASK3	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
208H	520	IA32_MTRR_PHYSBASE4	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
209H	521	IA32_MTRR_PHYSMASK4	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
20AH	522	IA32_MTRR_PHYSBASE5	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
20BH	523	IA32_MTRR_PHYSMASK5	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
20CH	524	IA32_MTRR_PHYSBASE6	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
20DH	525	IA32_MTRR_PHYSMASK6	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
20EH	526	IA32_MTRR_PHYSBASE7	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
20FH	527	IA32_MTRR_PHYSMASK7	0, 1, 2, 3, 4, 6	Shared	<b>Variable Range Mask MTRR</b> See Section 11.11.2.3, "Variable Range MTRRs."
250H	592	IA32_MTRR_FIX64K_00000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."
258H	600	IA32_MTRR_FIX16K_80000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."
259H	601	IA32_MTRR_FIX16K_A0000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."
268H	616	IA32_MTRR_FIX4K_C0000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
269H	617	IA32_MTRR_FIX4K_C8000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs".
26AH	618	IA32_MTRR_FIX4K_D0000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs".
26BH	619	IA32_MTRR_FIX4K_D8000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."
26CH	620	IA32_MTRR_FIX4K_E0000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."
26DH	621	IA32_MTRR_FIX4K_E8000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."
26EH	622	IA32_MTRR_FIX4K_F0000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."
26FH	623	IA32_MTRR_FIX4K_F8000	0, 1, 2, 3, 4, 6	Shared	<b>Fixed Range MTRR</b> See Section 11.11.2.2, "Fixed Range MTRRs."
277H	631	IA32_PAT	0, 1, 2, 3, 4, 6	Unique	<b>Page Attribute Table</b> See Section 11.11.2.2, "Fixed Range MTRRs."
2FFH	767	IA32_MTRR_DEF_TYPE	0, 1, 2, 3, 4, 6	Shared	<b>Default Memory Types (R/W)</b> See Table 2-2. 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 18.6.3.2, "Performance Counters."
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
307H	775	MSR_MS_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
308H	776	MSR_FLAME_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
309H	777	MSR_FLAME_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
30AH	778	MSR_FLAME_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
30BH	779	MSR_FLAME_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
30CH	780	MSR_IQ_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
30DH	781	MSR_IQ_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
30EH	782	MSR_IQ_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
30FH	783	MSR_IQ_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.2, "Performance Counters."
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
364H	868	MSR_MS_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
365H	869	MSR_MS_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
366H	870	MSR_MS_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
367H	871	MSR_MS_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
368H	872	MSR_FLAME_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
369H	873	MSR_FLAME_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
36AH	874	MSR_FLAME_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
36BH	875	MSR_FLAME_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
36CH	876	MSR_IQ_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."



Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.3, "CCCR MSRs."
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A6H	934	MSR_FLAME_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A7H	935	MSR_FLAME_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3AEH	942	MSR_SAAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3AFH	943	MSR_SAAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	Shared	See Section 18.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family OFH, models 01H-02H.
3BBH	955	MSR_IQ_ESCR1	0, 1, 2	Shared	See Section 18.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family OFH, models 01H-02H.
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3C0H	960	MSR_MS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3C9H	969	MSR_IX_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3F0H	1008	MSR_TC_PRECISE_EVENT	0, 1, 2, 3, 4, 6	Shared	See Section 18.6.3.1, "ESCR MSRs."
3F1H	1009	MSR_PEBS_ENABLE	0, 1, 2, 3, 4, 6	Shared	<b>Processor Event Based Sampling (PEBS) (R/W)</b> Controls the enabling of processor event sampling and replay tagging.
		12:0			See Table 19-36.
		23:13			Reserved.
		24			<b>UOP Tag</b> Enables replay tagging when set.
		25			<b>ENABLE_PEBS_MY_THR (R/W)</b> Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 18.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is called ENABLE_PEBS in IA-32 processors that do not support Intel Hyper-Threading Technology.
		26			<b>ENABLE_PEBS_OTH_THR (R/W)</b> Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 18.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is reserved for IA-32 processors that do not support Intel Hyper-Threading Technology.
		63:27			Reserved.
3F2H	1010	MSR_PEBS_MATRIX_VERT	0, 1, 2, 3, 4, 6	Shared	See Table 19-36.
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."

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
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.
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 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	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 ADDRIV 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 ADDRIV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC4_MISC			See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
480H	1152	IA32_VMX_BASIC	3, 4, 6	Unique	<b>Reporting Register of Basic VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	3, 4, 6	Unique	<b>Capability Reporting Register of Pin-based VM-execution Controls (R/O)</b> See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	3, 4, 6	Unique	<b>Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls," and see Table 2-2.
483H	1155	IA32_VMX_EXIT_CTL	3, 4, 6	Unique	<b>Capability Reporting Register of VM-exit Controls (R/O)</b> See Appendix A.4, "VM-Exit Controls," and see Table 2-2.
484H	1156	IA32_VMX_ENTRY_CTL	3, 4, 6	Unique	<b>Capability Reporting Register of VM-entry Controls (R/O)</b> See Appendix A.5, "VM-Entry Controls," and see Table 2-2.
485H	1157	IA32_VMX_MISC	3, 4, 6	Unique	<b>Reporting Register of Miscellaneous VMX Capabilities (R/O)</b> See Appendix A.6, "Miscellaneous Data," and see Table 2-2.
486H	1158	IA32_VMX_CR0_FIXED0	3, 4, 6	Unique	<b>Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)</b> See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 2-2.
487H	1159	IA32_VMX_CR0_FIXED1	3, 4, 6	Unique	<b>Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)</b> See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 2-2.
488H	1160	IA32_VMX_CR4_FIXED0	3, 4, 6	Unique	<b>Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)</b> See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 2-2.
489H	1161	IA32_VMX_CR4_FIXED1	3, 4, 6	Unique	<b>Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)</b> See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 2-2.
48AH	1162	IA32_VMX_VMCS_ENUM	3, 4, 6	Unique	<b>Capability Reporting Register of VMCS Field Enumeration (R/O)</b> See Appendix A.9, "VMCS Enumeration," and see Table 2-2.

Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
48BH	1163	IA32_VMX_PROCBASED_CTLSS2	3, 4, 6	Unique	<b>Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls," and see Table 2-2.
600H	1536	IA32_DS_AREA	0, 1, 2, 3, 4, 6	Unique	<b>DS Save Area (R/W)</b> See Table 2-2. See Section 18.6.3.4, "Debug Store (DS) Mechanism."
680H	1664	MSR_LASTBRANCH_0_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 0 (R/W)</b> 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 <b>source instruction</b> 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 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
681H	1665	MSR_LASTBRANCH_1_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 1</b> See description of MSR_LASTBRANCH_0 at 680H.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 2</b> See description of MSR_LASTBRANCH_0 at 680H.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 3</b> See description of MSR_LASTBRANCH_0 at 680H.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 4</b> See description of MSR_LASTBRANCH_0 at 680H.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 5</b> See description of MSR_LASTBRANCH_0 at 680H.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 6</b> See description of MSR_LASTBRANCH_0 at 680H.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 7</b> See description of MSR_LASTBRANCH_0 at 680H.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 8</b> See description of MSR_LASTBRANCH_0 at 680H.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 9</b> See description of MSR_LASTBRANCH_0 at 680H.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 10</b> See description of MSR_LASTBRANCH_0 at 680H.

**Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)**

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>7</sup>	Bit Description
Hex	Dec				
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 11</b> See description of MSR_LASTBRANCH_0 at 680H.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 12</b> See description of MSR_LASTBRANCH_0 at 680H.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 13</b> See description of MSR_LASTBRANCH_0 at 680H.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 14</b> See description of MSR_LASTBRANCH_0 at 680H.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	3, 4, 6	Unique	<b>Last Branch Record 15</b> See description of MSR_LASTBRANCH_0 at 680H.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 0 (R/W)</b> One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 1</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 2</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 3</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 4</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 5</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 6</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 7</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 8</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 9</b> See description of MSR_LASTBRANCH_0 at 6C0H.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 10</b> See description of MSR_LASTBRANCH_0 at 6C0H.



Table 2-45. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique <sup>1</sup>	Bit Description
Hex	Dec				
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 11</b> See description of MSR_LASTBRANCH_0 at 6COH.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 12</b> See description of MSR_LASTBRANCH_0 at 6COH.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 13</b> See description of MSR_LASTBRANCH_0 at 6COH.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 14</b> See description of MSR_LASTBRANCH_0 at 6COH.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	3, 4, 6	Unique	<b>Last Branch Record 15</b> See description of MSR_LASTBRANCH_0 at 6COH.
C000_0080H		IA32_EFER	3, 4, 6	Unique	<b>Extended Feature Enables</b> See Table 2-2.
C000_0081H		IA32_STAR	3, 4, 6	Unique	<b>System Call Target Address (R/W)</b> See Table 2-2.
C000_0082H		IA32_LSTAR	3, 4, 6	Unique	<b>IA-32e Mode System Call Target Address (R/W)</b> See Table 2-2.
C000_0084H		IA32_FMASK	3, 4, 6	Unique	<b>System Call Flag Mask (R/W)</b> See Table 2-2.
C000_0100H		IA32_FS_BASE	3, 4, 6	Unique	<b>Map of BASE Address of FS (R/W)</b> See Table 2-2.
C000_0101H		IA32_GS_BASE	3, 4, 6	Unique	<b>Map of BASE Address of GS (R/W)</b> See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	3, 4, 6	Unique	<b>Swap Target of BASE Address of GS (R/W)</b> See Table 2-2.

**NOTES**

1. For HT-enabled processors, there may be more than one logical processors per physical unit. If an MSR is Shared, this means that one MSR is shared between logical processors. If an MSR is unique, this means that each logical processor has its own MSR.

### 2.18.1 MSRs Unique to Intel® Xeon® Processor MP with L3 Cache

The MSRs listed in Table 2-46 apply to Intel® Xeon® Processor MP with up to 8MB level three cache. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 3 or 4 (see CPUID instruction for more details).

**Table 2-46. MSRs Unique to 64-bit Intel® Xeon® Processor MP with Up to an 8 MB L3 Cache**

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique	Bit Description
107CCH		MSR_IFSB_BUSQ0	3, 4	Shared	<b>IFSB BUSQ Event Control and Counter Register (R/W)</b> See Section 18.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CDH		MSR_IFSB_BUSQ1	3, 4	Shared	<b>IFSB BUSQ Event Control and Counter Register (R/W)</b>
107CEH		MSR_IFSB_SNPQ0	3, 4	Shared	<b>IFSB SNPQ Event Control and Counter Register (R/W)</b> See Section 18.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CFH		MSR_IFSB_SNPQ1	3, 4	Shared	<b>IFSB SNPQ Event Control and Counter Register (R/W)</b>
107D0H		MSR_EFSB_DRDY0	3, 4	Shared	<b>EFSB DRDY Event Control and Counter Register (R/W)</b> See Section 18.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D1H		MSR_EFSB_DRDY1	3, 4	Shared	<b>EFSB DRDY Event Control and Counter Register (R/W)</b>
107D2H		MSR_IFSB_CTL6	3, 4	Shared	<b>IFSB Latency Event Control Register (R/W)</b> See Section 18.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D3H		MSR_IFSB_CNTR7	3, 4	Shared	<b>IFSB Latency Event Counter Register (R/W)</b> See Section 18.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."

The MSRs listed in Table 2-47 apply to Intel® Xeon® Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 6 (See CPUID instruction for more details.). The performance monitoring MSRs listed in Table 2-47 are shared between logical processors in the same core, but are replicated for each core.

Table 2-47. MSRs Unique to Intel® Xeon® Processor 7100 Series

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique	Bit Description
107CCH		MSR_EMON_L3_CTR_CTL0	6	Shared	<b>GBUSQ Event Control and Counter Register (R/W)</b> See Section 18.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CDH		MSR_EMON_L3_CTR_CTL1	6	Shared	<b>GBUSQ Event Control and Counter Register (R/W)</b>
107CEH		MSR_EMON_L3_CTR_CTL2	6	Shared	<b>GSNPQ Event Control and Counter Register (R/W)</b> See Section 18.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CFH		MSR_EMON_L3_CTR_CTL3	6	Shared	<b>GSNPQ Event Control and Counter Register (R/W)</b>
107D0H		MSR_EMON_L3_CTR_CTL4	6	Shared	<b>FSB Event Control and Counter Register (R/W)</b> See Section 18.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D1H		MSR_EMON_L3_CTR_CTL5	6	Shared	<b>FSB Event Control and Counter Register (R/W)</b>
107D2H		MSR_EMON_L3_CTR_CTL6	6	Shared	<b>FSB Event Control and Counter Register (R/W)</b>
107D3H		MSR_EMON_L3_CTR_CTL7	6	Shared	<b>FSB Event Control and Counter Register (R/W)</b>

## 2.19 MSRS IN INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Model-specific registers (MSRs) for Intel Core Solo, Intel Core Duo processors, and Dual-core Intel Xeon processor LV are listed in Table 2-48. The column "Shared/Unique" applies to Intel Core Duo processor. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	P5_MC_ADDR	Unique	See Section 2.22, "MSRs in Pentium Processors," and see Table 2-2.
1H	1	P5_MC_TYPE	Unique	See Section 2.22, "MSRs in Pentium Processors," and see Table 2-2.
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and see Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.

**Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
17H	23	IA32_PLATFORM_ID	Shared	<b>Platform ID (R)</b> See Table 2-2. The operating system can use this MSR to determine “slot” information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, “Local APIC Status and Location,” and see Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	<b>Processor Hard Power-On Configuration (R/W)</b> Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved.
		1		<b>Data Error Checking Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		2		<b>Response Error Checking Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		<b>MCERR# Drive Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		4		<b>Address Parity Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		6: 5		Reserved
		7		<b>BINIT# Driver Enable (R/W)</b> 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		<b>Output Tri-state Enabled (R/O)</b> 1 = Enabled; 0 = Disabled
		9		<b>Execute BIST (R/O)</b> 1 = Enabled; 0 = Disabled
		10		<b>MCERR# Observation Enabled (R/O)</b> 1 = Enabled; 0 = Disabled
		11		Reserved
		12		<b>BINIT# Observation Enabled (R/O)</b> 1 = Enabled; 0 = Disabled
		13		Reserved
		14		<b>1 MByte Power on Reset Vector (R/O)</b> 1 = 1 MByte; 0 = 4 GBytes
15		Reserved		
17:16		<b>APIC Cluster ID (R/O)</b>		

Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		18		<b>System Bus Frequency (R/O)</b> 0 = 100 MHz 1 = Reserved
		19		Reserved.
		21:20		<b>Symmetric Arbitration ID (R/O)</b>
		26:22		<b>Clock Frequency Ratio (R/O)</b>
3AH	58	IA32_FEATURE_CONTROL	Unique	<b>Control Features in IA-32 Processor (R/W)</b> See Table 2-2.
40H	64	MSR_LASTBRANCH_0	Unique	<b>Last Branch Record 0 (R/W)</b> One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the 'to' address. See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."</li> </ul>
41H	65	MSR_LASTBRANCH_1	Unique	<b>Last Branch Record 1 (R/W)</b> See description of MSR_LASTBRANCH_0.
42H	66	MSR_LASTBRANCH_2	Unique	<b>Last Branch Record 2 (R/W)</b> See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	Unique	<b>Last Branch Record 3 (R/W)</b> See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	Unique	<b>Last Branch Record 4 (R/W)</b> See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	Unique	<b>Last Branch Record 5 (R/W)</b> See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	Unique	<b>Last Branch Record 6 (R/W)</b> See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	Unique	<b>Last Branch Record 7 (R/W)</b> See description of MSR_LASTBRANCH_0.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	<b>BIOS Update Trigger Register (W)</b> See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	<b>BIOS Update Signature ID (RO)</b> See Table 2-2.
C1H	193	IA32_PMC0	Unique	<b>Performance counter register</b> See Table 2-2.
C2H	194	IA32_PMC1	Unique	<b>Performance counter register</b> See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	<b>Scaleable Bus Speed (RO)</b> This field indicates the scaleable bus clock speed:

**Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2:0		<ul style="list-style-type: none"> <li>▪ 101B: 100 MHz (FSB 400)</li> <li>▪ 001B: 133 MHz (FSB 533)</li> <li>▪ 011B: 167 MHz (FSB 667)</li> </ul> <p>133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.</p>
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	<b>Maximum Performance Frequency Clock Count. (RW)</b> See Table 2-2.
E8H	232	IA32_APERF	Unique	<b>Actual Performance Frequency Clock Count. (RW)</b> See Table 2-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 2-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	Control register 3. Used to configure the L2 Cache.
		0		<b>L2 Hardware Enabled (RO)</b> 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		<b>L2 Enabled (R/W)</b> 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		<b>L2 Not Present (RO)</b> 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		<b>RIPV</b> When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.

Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		1		<b>EIPV</b> When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		<b>MCIP</b> When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSELO	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	<b>Clock Modulation (R/W)</b> See Table 2-2.
19BH	411	IA32_THERM_INTERRUPT	Unique	<b>Thermal Interrupt Control (R/W)</b> See Table 2-2. See Section 14.7.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	Unique	<b>Thermal Monitor Status (R/W)</b> See Table 2-2. See Section 14.7.2, "Thermal Monitor".
19DH	413	MSR_THERM2_CTL	Unique	Thermal Monitor 2 Control
		15:0		Reserved.
		16		<b>TM_SELECT (R/W)</b> Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16		Reserved.
1A0H	416	IA32_MISC_ENABLE		<b>Enable Miscellaneous Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.
		2:0		Reserved.
		3	Unique	<b>Automatic Thermal Control Circuit Enable (R/W)</b> See Table 2-2.
		6:4		Reserved.
		7	Shared	<b>Performance Monitoring Available (R)</b> See Table 2-2.

**Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)**

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		9:8		Reserved.
		10	Shared	<b>FERR# Multiplexing Enable (R/W)</b> 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	<b>Branch Trace Storage Unavailable (RO)</b> See Table 2-2.
		12		Reserved.
		13	Shared	<b>TM2 Enable (R/W)</b> When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.  When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state.  If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		15:14		Reserved.
		16	Shared	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> 1 = Enhanced Intel SpeedStep Technology enabled
		18	Shared	<b>ENABLE MONITOR FSM (R/W)</b> See Table 2-2.
		19		<b>Reserved.</b>
		22	Shared	<b>Limit CPUID Maxval (R/W)</b> See Table 2-2. Setting this bit may cause behavior in software that depends on the availability of CPUID leaves greater than 2.
		33:23		Reserved.
		34	Shared	<b>XD Bit Disable (R/W)</b> See Table 2-2.
		63:35		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Unique	<b>Last Branch Record Stack TOS (R/W)</b> Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).



Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Unique	<b>Debug Control (R/W)</b> Controls how several debug features are used. Bit definitions are discussed in Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	<b>Last Exception Record From Linear IP (R)</b> 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	<b>Last Exception Record To Linear IP (R)</b> This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	MTRRphysBase0	Unique	Memory Type Range Registers
201H	513	MTRRphysMask0	Unique	Memory Type Range Registers
202H	514	MTRRphysBase1	Unique	Memory Type Range Registers
203H	515	MTRRphysMask1	Unique	Memory Type Range Registers
204H	516	MTRRphysBase2	Unique	Memory Type Range Registers
205H	517	MTRRphysMask2	Unique	Memory Type Range Registers
206H	518	MTRRphysBase3	Unique	Memory Type Range Registers
207H	519	MTRRphysMask3	Unique	Memory Type Range Registers
208H	520	MTRRphysBase4	Unique	Memory Type Range Registers
209H	521	MTRRphysMask4	Unique	Memory Type Range Registers
20AH	522	MTRRphysBase5	Unique	Memory Type Range Registers
20BH	523	MTRRphysMask5	Unique	Memory Type Range Registers
20CH	524	MTRRphysBase6	Unique	Memory Type Range Registers
20DH	525	MTRRphysMask6	Unique	Memory Type Range Registers
20EH	526	MTRRphysBase7	Unique	Memory Type Range Registers
20FH	527	MTRRphysMask7	Unique	Memory Type Range Registers
250H	592	MTRRfix64K_00000	Unique	Memory Type Range Registers
258H	600	MTRRfix16K_80000	Unique	Memory Type Range Registers
259H	601	MTRRfix16K_A0000	Unique	Memory Type Range Registers
268H	616	MTRRfix4K_C0000	Unique	Memory Type Range Registers
269H	617	MTRRfix4K_C8000	Unique	Memory Type Range Registers
26AH	618	MTRRfix4K_D0000	Unique	Memory Type Range Registers
26BH	619	MTRRfix4K_D8000	Unique	Memory Type Range Registers
26CH	620	MTRRfix4K_E0000	Unique	Memory Type Range Registers
26DH	621	MTRRfix4K_E8000	Unique	Memory Type Range Registers
26EH	622	MTRRfix4K_F0000	Unique	Memory Type Range Registers
26FH	623	MTRRfix4K_F8000	Unique	Memory Type Range Registers

Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	<b>Default Memory Types (R/W)</b> See Table 2-2. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
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.
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	IA32_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC3_STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	MSR_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the 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	Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.
414H	1044	MSR_MC5_CTL	Unique	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).

Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
415H	1045	MSR_MC5_STATUS	Unique	Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
416H	1046	MSR_MC5_ADDR	Unique	Machine Check Error Reporting Register - contains the address of the code or data memory location that produced the machine-check error if the ADDR flag in the IA32_MCI_STATUS register is set.
417H	1047	MSR_MC5_MISC	Unique	Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISC flag in the IA32_MCI_STATUS register is set.
480H	1152	IA32_VMX_BASIC	Unique	<b>Reporting Register of Basic VMX Capabilities (R/O)</b> See Table 2-2. See Appendix A.1, “Basic VMX Information” (If CPUID.01H:ECX.[bit 9])
481H	1153	IA32_VMX_PINBASED_CTL	Unique	<b>Capability Reporting Register of Pin-based VM-execution Controls (R/O)</b> See Appendix A.3, “VM-Execution Controls” (If CPUID.01H:ECX.[bit 9])
482H	1154	IA32_VMX_PROCBASED_CTL	Unique	<b>Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, “VM-Execution Controls” (If CPUID.01H:ECX.[bit 9])
483H	1155	IA32_VMX_EXIT_CTL	Unique	<b>Capability Reporting Register of VM-exit Controls (R/O)</b> See Appendix A.4, “VM-Exit Controls” (If CPUID.01H:ECX.[bit 9])
484H	1156	IA32_VMX_ENTRY_CTL	Unique	<b>Capability Reporting Register of VM-entry Controls (R/O)</b> See Appendix A.5, “VM-Entry Controls” (If CPUID.01H:ECX.[bit 9])
485H	1157	IA32_VMX_MISC	Unique	<b>Reporting Register of Miscellaneous VMX Capabilities (R/O)</b> See Appendix A.6, “Miscellaneous Data” (If CPUID.01H:ECX.[bit 9])
486H	1158	IA32_VMX_CR0_FIXED0	Unique	<b>Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)</b> See Appendix A.7, “VMX-Fixed Bits in CR0” (If CPUID.01H:ECX.[bit 9])
487H	1159	IA32_VMX_CR0_FIXED1	Unique	<b>Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)</b> See Appendix A.7, “VMX-Fixed Bits in CR0” (If CPUID.01H:ECX.[bit 9])
488H	1160	IA32_VMX_CR4_FIXED0	Unique	<b>Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)</b> See Appendix A.8, “VMX-Fixed Bits in CR4” (If CPUID.01H:ECX.[bit 9])

**Table 2-48. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)**

Register Address		Register Name	Shared/Unique	Bit Description
Hex	Dec			
489H	1161	IA32_VMX_CR4_FIXED1	Unique	<b>Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)</b> See Appendix A.8, "VMX-Fixed Bits in CR4" (If CPUID.01H:ECX.[bit 9])
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	<b>Capability Reporting Register of VMCS Field Enumeration (R/O)</b> See Appendix A.9, "VMCS Enumeration" (If CPUID.01H:ECX.[bit 9])
48BH	1163	IA32_VMX_PROCBASED_CTLD2	Unique	<b>Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)</b> See Appendix A.3, "VM-Execution Controls" (If CPUID.01H:ECX.[bit 9] and IA32_VMX_PROCBASED_CTLD[bit 63])
600H	1536	IA32_DS_AREA	Unique	<b>DS Save Area (R/W)</b> See Table 2-2. See Section 18.6.3.4, "Debug Store (DS) Mechanism."
		31:0		<b>DS Buffer Management Area</b> Linear address of the first byte of the DS buffer management area.
		63:32		Reserved.
C000_0080H		IA32_EFER	Unique	See Table 2-2.
		10:0		Reserved.
		11		<b>Execute Disable Bit Enable</b>
		63:12		Reserved.

## 2.20 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section 2.21 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

**Table 2-49. MSRs in Pentium M Processors**

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 2.22, "MSRs in Pentium Processors."
1H	1	P5_MC_TYPE	See Section 2.22, "MSRs in Pentium Processors."
10H	16	IA32_TIME_STAMP_COUNTER	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	<b>Platform ID (R)</b> See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.

Table 2-49. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
2AH	42	MSR_EBL_CR_POWERON	<b>Processor Hard Power-On Configuration (R/W)</b> Enables and disables processor features. <b>(R)</b> Indicates current processor configuration.
		0	Reserved.
		1	<b>Data Error Checking Enable (R)</b> 0 = Disabled Always 0 on the Pentium M processor.
		2	<b>Response Error Checking Enable (R)</b> 0 = Disabled Always 0 on the Pentium M processor.
		3	<b>MCERR# Drive Enable (R)</b> 0 = Disabled Always 0 on the Pentium M processor.
		4	<b>Address Parity Enable (R)</b> 0 = Disabled Always 0 on the Pentium M processor.
		6:5	Reserved.
		7	<b>BINIT# Driver Enable (R)</b> 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		8	<b>Output Tri-state Enabled (R/O)</b> 1 = Enabled; 0 = Disabled
		9	<b>Execute BIST (R/O)</b> 1 = Enabled; 0 = Disabled
		10	<b>MCERR# Observation Enabled (R/O)</b> 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		11	Reserved.
		12	<b>BINIT# Observation Enabled (R/O)</b> 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		13	Reserved.
		14	<b>1 MByte Power on Reset Vector (R/O)</b> 1 = 1 MByte; 0 = 4 GBytes Always 0 on the Pentium M processor.
15	Reserved.		
17:16	<b>APIC Cluster ID (R/O)</b> Always 00B on the Pentium M processor.		

**Table 2-49. MSRs in Pentium M Processors (Contd.)**

Register Address		Register Name	Bit Description
Hex	Dec		
		18	<b>System Bus Frequency (R/O)</b> 0 = 100 MHz 1 = Reserved Always 0 on the Pentium M processor.
		19	Reserved.
		21:20	<b>Symmetric Arbitration ID (R/O)</b> Always 00B on the Pentium M processor.
		26:22	Clock Frequency Ratio (R/O)
40H	64	MSR_LASTBRANCH_0	<b>Last Branch Record 0 (R/W)</b> One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: <ul style="list-style-type: none"> <li>▪ Last Branch Record Stack TOS at 1C9H</li> <li>▪ Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)"</li> </ul>
41H	65	MSR_LASTBRANCH_1	<b>Last Branch Record 1 (R/W)</b> See description of MSR_LASTBRANCH_0.
42H	66	MSR_LASTBRANCH_2	<b>Last Branch Record 2 (R/W)</b> See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	<b>Last Branch Record 3 (R/W)</b> See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	<b>Last Branch Record 4 (R/W)</b> See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	<b>Last Branch Record 5 (R/W)</b> See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	<b>Last Branch Record 6 (R/W)</b> See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	<b>Last Branch Record 7 (R/W)</b> See description of MSR_LASTBRANCH_0.
119H	281	MSR_BBL_CR_CTL	Control register Used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.
		63:0	Reserved.
11EH	281	MSR_BBL_CR_CTL3	Control register 3 Used to configure the L2 Cache.
		0	<b>L2 Hardware Enabled (RO)</b> 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		4:1	Reserved.

Table 2-49. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		5	<b>ECC Check Enable (RO)</b> This bit enables ECC checking on the cache data bus. ECC is always generated on write cycles. 0 = Disabled (default) 1 = Enabled For the Pentium M processor, ECC checking on the cache data bus is always enabled.
		7:6	Reserved.
		8	<b>L2 Enabled (R/W)</b> 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	<b>L2 Not Present (RO)</b> 0 = L2 Present 1 = L2 Not Present
		63:24	Reserved.
179H	377	IA32_MCG_CAP	Read-only register that provides information about the machine-check architecture of the processor.
		7:0	<b>Count (RO)</b> Indicates the number of hardware unit error reporting banks available in the processor.
		8	<b>IA32_MCG_CTL Present (RO)</b> 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
		63:9	Reserved.
17AH	378	IA32_MCG_STATUS	Global Machine Check Status
		0	<b>RIPV</b> When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1	<b>EIPV</b> When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2	<b>MCIP</b> When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.

**Table 2-49. MSRs in Pentium M Processors (Contd.)**

Register Address		Register Name	Bit Description
Hex	Dec		
		63:3	Reserved.
198H	408	IA32_PERF_STATUS	See Table 2-2.
199H	409	IA32_PERF_CTL	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	<b>Clock Modulation (R/W).</b> See Table 2-2. See Section 14.7.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	<b>Thermal Interrupt Control (R/W)</b> See Table 2-2. See Section 14.7.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	<b>Thermal Monitor Status (R/W)</b> See Table 2-2. See Section 14.7.2, "Thermal Monitor."
19DH	413	MSR_THERM2_CTL	Thermal Monitor 2 Control
		15:0	Reserved.
		16	<b>TM_SELECT (R/W)</b> Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16	Reserved.
1A0H	416	IA32_MISC_ENABLE	<b>Enable Miscellaneous Processor Features (R/W)</b> Allows a variety of processor functions to be enabled and disabled.
		2:0	Reserved.
		3	<b>Automatic Thermal Control Circuit Enable (R/W)</b> 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
		6:4	Reserved.



Table 2-49. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		7	<b>Performance Monitoring Available (R)</b> 1 = Performance monitoring enabled 0 = Performance monitoring disabled
		9:8	Reserved.
		10	<b>FERR# Multiplexing Enable (R/W)</b> 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.
			<b>Branch Trace Storage Unavailable (RO)</b> 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported
		12	<b>Processor Event Based Sampling Unavailable (RO)</b> 1 = Processor does not support processor event based sampling (PEBS); 0 = PEBS is supported. The Pentium M processor does not support PEBS.
		15:13	Reserved.
		16	<b>Enhanced Intel SpeedStep Technology Enable (R/W)</b> 1 = Enhanced Intel SpeedStep Technology enabled. On the Pentium M processor, this bit may be configured to be read-only.
		22:17	Reserved.
		23	<b>xTPR Message Disable (R/W)</b> When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority. The default is processor specific.
		63:24	Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	<b>Last Branch Record Stack TOS (R/W)</b> Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: <ul style="list-style-type: none"> <li>▪ MSR_LASTBRANCH_0_FROM_IP (at 40H)</li> <li>▪ Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)"</li> </ul>
1D9H	473	MSR_DEBUGCTLB	<b>Debug Control (R/W)</b> Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."

**Table 2-49. MSRs in Pentium M Processors (Contd.)**

Register Address		Register Name	Bit Description
Hex	Dec		
1DDH	477	MSR_LER_TO_LIP	<p><b>Last Exception Record To Linear IP (R)</b></p> <p>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.</p> <p>See Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.16.2, "Last Branch and Last Exception MSRs."</p>
1DEH	478	MSR_LER_FROM_LIP	<p><b>Last Exception Record From Linear IP (R)</b></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 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.16.2, "Last Branch and Last Exception MSRs."</p>
2FFH	767	IA32_MTRR_DEF_TYPE	<p><b>Default Memory Types (R/W)</b></p> <p>Sets the memory type for the regions of physical memory that are not mapped by the MTRRs.</p> <p>See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."</p>
400H	1024	IA32_MCO_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	<p>See Section 14.3.2.3, "IA32_MCi_ADDR MSRs."</p> <p>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.</p>
404H	1028	IA32_MC1_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	<p>See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."</p> <p>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.</p>
408H	1032	IA32_MC2_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	See Chapter 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	<p>See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."</p> <p>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.</p>
40CH	1036	MSR_MC4_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-49. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
40EH	1038	MSR_MC4_ADDR	See Section 15.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the 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	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.
600H	1536	IA32_DS_AREA	<b>DS Save Area (R/W)</b> See Table 2-2. Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 18.6.3.4, "Debug Store (DS) Mechanism."
		31:0	<b>DS Buffer Management Area</b> Linear address of the first byte of the DS buffer management area.
		63:32	Reserved.

## 2.21 MSRS IN THE P6 FAMILY PROCESSORS

The following MSRs are defined for the P6 family processors. The MSRs in this table that are shaded are available only in the Pentium II and Pentium III processors. Beginning with the Pentium 4 processor, some of the MSRs in this list have been designated as "architectural" and have had their names changed. See Table 2-2 for a list of the architectural MSRs.

Table 2-50. MSRs in the P6 Family Processors

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 2.22, "MSRs in Pentium Processors."
1H	1	P5_MC_TYPE	See Section 2.22, "MSRs in Pentium Processors."
10H	16	TSC	See Section 17.17, "Time-Stamp Counter."
17H	23	IA32_PLATFORM_ID	<b>Platform ID (R)</b> The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
		49:0	Reserved.

**Table 2-50. MSRs in the P6 Family Processors (Contd.)**

Register Address		Register Name	Bit Description
Hex	Dec		
		52:50	<b>Platform Id (R)</b> Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
		56:53	L2 Cache Latency Read.
		59:57	Reserved.
		60	Clock Frequency Ratio Read.
		63:61	Reserved.
1BH	27	APIC_BASE	Section 10.4.4, "Local APIC Status and Location."
		7:0	Reserved.
		8	<b>Boot Strap Processor indicator Bit</b> 1 = BSP
		10:9	Reserved.
		11	<b>APIC Global Enable Bit - Permanent till reset</b> 1 = Enabled 0 = Disabled
		31:12	APIC Base Address.
		63:32	Reserved.
2AH	42	EBL_CR_POWERON	<b>Processor Hard Power-On Configuration (R/W)</b> Enables and disables processor features; <b>(R)</b> indicates current processor configuration.
		0	Reserved. <sup>1</sup>
		1	<b>Data Error Checking Enable (R/W)</b> 1 = Enabled 0 = Disabled
		2	<b>Response Error Checking Enable FRCERR Observation Enable (R/W)</b> 1 = Enabled 0 = Disabled
		3	<b>AERR# Drive Enable (R/W)</b> 1 = Enabled 0 = Disabled
		4	<b>BERR# Enable for Initiator Bus Requests (R/W)</b> 1 = Enabled 0 = Disabled
		5	Reserved.

Table 2-50. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		6	<b>BERR# Driver Enable for Initiator Internal Errors (R/W)</b> 1 = Enabled 0 = Disabled
		7	<b>BINIT# Driver Enable (R/W)</b> 1 = Enabled 0 = Disabled
		8	<b>Output Tri-state Enabled (R)</b> 1 = Enabled 0 = Disabled
		9	<b>Execute BIST (R)</b> 1 = Enabled 0 = Disabled
		10	<b>AERR# Observation Enabled (R)</b> 1 = Enabled 0 = Disabled
		11	Reserved.
		12	<b>BINIT# Observation Enabled (R)</b> 1 = Enabled 0 = Disabled
		13	<b>In Order Queue Depth (R)</b> 1 = 1 0 = 8
		14	<b>1-MByte Power on Reset Vector (R)</b> 1 = 1MByte 0 = 4GBytes
		15	<b>FRC Mode Enable (R)</b> 1 = Enabled 0 = Disabled
		17:16	<b>APIC Cluster ID (R)</b>
		19:18	<b>System Bus Frequency (R)</b> 00 = 66MHz 10 = 100Mhz 01 = 133MHz 11 = Reserved
		21:20	<b>Symmetric Arbitration ID (R)</b>
		25:22	<b>Clock Frequency Ratio (R)</b>
		26	<b>Low Power Mode Enable (R/W)</b>
		27	<b>Clock Frequency Ratio</b>
		63:28	Reserved. <sup>1</sup>
33H	51	TEST_CTL	<b>Test Control Register</b>

**Table 2-50. MSRs in the P6 Family Processors (Contd.)**

Register Address		Register Name	Bit Description
Hex	Dec		
		29:0	Reserved.
		30	<b>Streaming Buffer Disable</b>
		31	<b>Disable LOCK#</b> Assertion for split locked access.
79H	121	BIOS_UPDT_TRIG	BIOS Update Trigger Register.
88H	136	BBL_CR_D0[63:0]	Chunk 0 data register D[63:0]; used to write to and read from the L2
89H	137	BBL_CR_D1[63:0]	Chunk 1 data register D[63:0]; used to write to and read from the L2
8AH	138	BBL_CR_D2[63:0]	Chunk 2 data register D[63:0]; used to write to and read from the L2
8BH	139	BIOS_SIGN/BBL_CR_D3[63:0]	<b>BIOS Update Signature Register or Chunk 3 data register D[63:0]</b> Used to write to and read from the L2 depending on the usage model.
C1H	193	PerfCtr0 (PERFCTR0)	Performance Counter Register See Table 2-2.
C2H	194	PerfCtr1 (PERFCTR1)	Performance Counter Register See Table 2-2.
FEH	254	MTRRcap	Memory Type Range Registers
116H	278	BBL_CR_ADDR [63:0] BBL_CR_ADDR [63:32] BBL_CR_ADDR [31:3] BBL_CR_ADDR [2:0]	Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses. Reserved, Address bits [35:3] Reserved Set to 0.
118H	280	BBL_CR_DECC[63:0]	Data ECC register D[7:0]: used to write ECC and read ECC to/from L2
119H	281	BBL_CR_CTL BL_CR_CTL[63:22] BBL_CR_CTL[21]  BBL_CR_CTL[20:19] BBL_CR_CTL[18] BBL_CR_CTL[17] BBL_CR_CTL[16] BBL_CR_CTL[15:14] BBL_CR_CTL[13:12]  BBL_CR_CTL[11:10]  BBL_CR_CTL[9:8] BBL_CR_CTL[7] BBL_CR_CTL[6:5]	Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response Reserved Processor number <sup>2</sup> Disable = 1 Enable = 0 Reserved  User supplied ECC Reserved L2 Hit Reserved State from L2 Modified - 11, Exclusive - 10, Shared - 01, Invalid - 00 Way from L2 Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11 Way to L2 Reserved State to L2

Table 2-50. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		BBL_CR_CTL[4:0] 01100 01110 01111 00010 00011 010 + MESI encode 111 + MESI encode 100 + MESI encode	L2 Command Data Read w/ LRU update (RLU) Tag Read w/ Data Read (TRR) Tag Inquire (TI) L2 Control Register Read (CR) L2 Control Register Write (CW) Tag Write w/ Data Read (TWR) Tag Write w/ Data Write (TWW) Tag Write (TW)
11AH	282	BBL_CR_TRIG	Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.
11BH	283	BBL_CR_BUSY	Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY
11EH	286	BBL_CR_CTL3  BBL_CR_CTL3[63:26] BBL_CR_CTL3[25] BBL_CR_CTL3[24] BBL_CR_CTL3[23]  BBL_CR_CTL3[22:20] 111 110 101 100 011 010 001 000  BBL_CR_CTL3[19] BBL_CR_CTL3[18]	Control register 3: used to configure the L2 Cache  Reserved Cache bus fraction (read only) Reserved L2 Hardware Disable (read only)  L2 Physical Address Range support 64GBytes 32GBytes 16GBytes 8GBytes 4GBytes 2GBytes 1GBytes 512MBytes  Reserved Cache State error checking enable (read/write)

**Table 2-50. MSRs in the P6 Family Processors (Contd.)**

Register Address		Register Name	Bit Description
Hex	Dec		
		BBL_CR_CTL3[17:13] 00001 00010 00100 01000 10000  BBL_CR_CTL3[12:11] BBL_CR_CTL3[10:9] 00 01 10 11  BBL_CR_CTL3[8] BBL_CR_CTL3[7] BBL_CR_CTL3[6] BBL_CR_CTL3[5] BBL_CR_CTL3[4:1] BBL_CR_CTL3[0]	Cache size per bank (read/write) 256KBytes 512KBytes 1MByte 2MByte 4MBytes  Number of L2 banks (read only) L2 Associativity (read only) Direct Mapped 2 Way 4 Way Reserved  L2 Enabled (read/write) CRTN Parity Check Enable (read/write) Address Parity Check Enable (read/write) ECC Check Enable (read/write) L2 Cache Latency (read/write) L2 Configured (read/write )
174H	372	SYSENTER_CS_MSR	CS register target for CPL 0 code
175H	373	SYSENTER_ESP_MSR	Stack pointer for CPL 0 stack
176H	374	SYSENTER_EIP_MSR	CPL 0 code entry point
179H	377	MCG_CAP	Machine Check Global Control Register
17AH	378	MCG_STATUS	Machine Check Error Reporting Register - contains information related to machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
17BH	379	MCG_CTL	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
186H	390	PerfEvtSel0 (EVNTSEL0)	Performance Event Select Register 0 (R/W)
		7:0	<b>Event Select</b> Refer to Performance Counter section for a list of event encodings.
		15:8	<b>UMASK (Unit Mask)</b> Unit mask register set to 0 to enable all count options.
		16	<b>USER</b> Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	<b>OS</b> Controls the counting of events at Privilege level of 0.



Table 2-50. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		18	<b>E</b> Occurrence/Duration Mode Select 1 = Occurrence 0 = Duration
		19	<b>PC</b> Enabled the signaling of performance counter overflow via BPO pin
		20	<b>INT</b> Enables the signaling of counter overflow via input to APIC 1 = Enable 0 = Disable
		22	<b>ENABLE</b> Enables the counting of performance events in both counters 1 = Enable 0 = Disable
		23	<b>INV</b> Inverts the result of the CMASK condition 1 = Inverted 0 = Non-Inverted
		31:24	CMASK (Counter Mask).
187H	391	PerfEvtSel1 (EVNTSEL1)	Performance Event Select for Counter 1 (R/W)
		7:0	<b>Event Select</b> Refer to Performance Counter section for a list of event encodings.
		15:8	<b>UMASK (Unit Mask)</b> Unit mask register set to 0 to enable all count options.
		16	<b>USER</b> Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	<b>OS</b> Controls the counting of events at Privilege level of 0
		18	<b>E</b> Occurrence/Duration Mode Select 1 = Occurrence 0 = Duration
		19	<b>PC</b> Enabled the signaling of performance counter overflow via BPO pin.

**Table 2-50. MSRs in the P6 Family Processors (Contd.)**

Register Address		Register Name	Bit Description
Hex	Dec		
		20	<b>INT</b> Enables the signaling of counter overflow via input to APIC 1 = Enable 0 = Disable
		23	<b>INV</b> Inverts the result of the CMASK condition 1 = Inverted 0 = Non-Inverted
		31:24	<b>CMASK (Counter Mask)</b>
1D9H	473	DEBUGCTLMR	Enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.
		0	Enable/Disable Last Branch Records
		1	Branch Trap Flag
		2	Performance Monitoring/Break Point Pins
		3	Performance Monitoring/Break Point Pins
		4	Performance Monitoring/Break Point Pins
		5	Performance Monitoring/Break Point Pins
		6	Enable/Disable Execution Trace Messages
		31:7	Reserved
1DBH	475	LASTBRANCHFROMIP	32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.
1DCH	476	LASTBRANCHTOIP	32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.
1DDH	477	LASTINTFROMIP	Last INT from IP
1DEH	478	LASTINTTOIP	Last INT to IP
200H	512	MTRRphysBase0	Memory Type Range Registers
201H	513	MTRRphysMask0	Memory Type Range Registers
202H	514	MTRRphysBase1	Memory Type Range Registers
203H	515	MTRRphysMask1	Memory Type Range Registers
204H	516	MTRRphysBase2	Memory Type Range Registers
205H	517	MTRRphysMask2	Memory Type Range Registers
206H	518	MTRRphysBase3	Memory Type Range Registers
207H	519	MTRRphysMask3	Memory Type Range Registers
208H	520	MTRRphysBase4	Memory Type Range Registers
209H	521	MTRRphysMask4	Memory Type Range Registers
20AH	522	MTRRphysBase5	Memory Type Range Registers

Table 2-50. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
20BH	523	MTRRphysMask5	Memory Type Range Registers
20CH	524	MTRRphysBase6	Memory Type Range Registers
20DH	525	MTRRphysMask6	Memory Type Range Registers
20EH	526	MTRRphysBase7	Memory Type Range Registers
20FH	527	MTRRphysMask7	Memory Type Range Registers
250H	592	MTRRfix64K_00000	Memory Type Range Registers
258H	600	MTRRfix16K_80000	Memory Type Range Registers
259H	601	MTRRfix16K_A0000	Memory Type Range Registers
268H	616	MTRRfix4K_C0000	Memory Type Range Registers
269H	617	MTRRfix4K_C8000	Memory Type Range Registers
26AH	618	MTRRfix4K_D0000	Memory Type Range Registers
26BH	619	MTRRfix4K_D8000	Memory Type Range Registers
26CH	620	MTRRfix4K_E0000	Memory Type Range Registers
26DH	621	MTRRfix4K_E8000	Memory Type Range Registers
26EH	622	MTRRfix4K_F0000	Memory Type Range Registers
26FH	623	MTRRfix4K_F8000	Memory Type Range Registers
2FFH	767	MTRRdefType	Memory Type Range Registers
		2:0	Default memory type
		10	Fixed MTRR enable
		11	MTRR Enable
400H	1024	MCO_CTL	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
401H	1025	MCO_STATUS	Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
		15:0	MC_STATUS_MCACOD
		31:16	MC_STATUS_MSCOD
		57	MC_STATUS_DAM
		58	MC_STATUS_ADDRV
		59	MC_STATUS_MISCV
		60	MC_STATUS_EN. (Note: For MCO_STATUS only, this bit is hardcoded to 1.)
		61	MC_STATUS_UC
		62	MC_STATUS_O
63	MC_STATUS_V		
402H	1026	MCO_ADDR	
403H	1027	MCO_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

**Table 2-50. MSRs in the P6 Family Processors (Contd.)**

Register Address		Register Name	Bit Description
Hex	Dec		
404H	1028	MC1_CTL	
405H	1029	MC1_STATUS	Bit definitions same as MCO_STATUS.
406H	1030	MC1_ADDR	
407H	1031	MC1_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
408H	1032	MC2_CTL	
409H	1033	MC2_STATUS	Bit definitions same as MCO_STATUS.
40AH	1034	MC2_ADDR	
40BH	1035	MC2_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
40CH	1036	MC4_CTL	
40DH	1037	MC4_STATUS	Bit definitions same as MCO_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.
40EH	1038	MC4_ADDR	Defined in MCA architecture but not implemented in P6 Family processors.
40FH	1039	MC4_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
410H	1040	MC3_CTL	
411H	1041	MC3_STATUS	Bit definitions same as MCO_STATUS.
412H	1042	MC3_ADDR	
413H	1043	MC3_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

**NOTES**

1. Bit 0 of this register has been redefined several times, and is no longer used in P6 family processors.
2. The processor number feature may be disabled by setting bit 21 of the BBL\_CR\_CTL MSR (model-specific register address 119h) to "1". Once set, bit 21 of the BBL\_CR\_CTL may not be cleared. This bit is write-once. The processor number feature will be disabled until the processor is reset.
3. The Pentium III processor will prevent FSB frequency overclocking with a new shutdown mechanism. If the FSB frequency selected is greater than the internal FSB frequency the processor will shutdown. If the FSB selected is less than the internal FSB frequency the BIOS may choose to use bit 11 to implement its own shutdown policy.

## 2.22 MSRS IN PENTIUM PROCESSORS

The following MSRs are defined for the Pentium processors. The P5\_MC\_ADDR, P5\_MC\_TYPE, and TSC MSRs (named IA32\_P5\_MC\_ADDR, IA32\_P5\_MC\_TYPE, and IA32\_TIME\_STAMP\_COUNTER in the Pentium 4 processor) are architectural; that is, code that accesses these registers will run on Pentium 4 and P6 family processors without generating exceptions (see Section 2.1, "Architectural MSRs"). The CESR, CTR0, and CTR1 MSRs are unique to Pentium processors; code that accesses these registers will generate exceptions on Pentium 4 and P6 family processors.

Table 2-51. MSRs in the Pentium Processor

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 15.10.2, "Pentium Processor Machine-Check Exception Handling."
1H	1	P5_MC_TYPE	See Section 15.10.2, "Pentium Processor Machine-Check Exception Handling."
10H	16	TSC	See Section 17.17, "Time-Stamp Counter."
11H	17	CESR	See Section 18.6.9.1, "Control and Event Select Register (CESR)."
12H	18	CTRO	Section 18.6.9.3, "Events Counted."
13H	19	CTR1	Section 18.6.9.3, "Events Counted."

## 2.23 MSR INDEX

MSRs of recent processors are indexed here for convenience. IA32 MSRs are excluded from this index.

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_ALF_ESCR0	
0FH .....	See Table 2-45
MSR_ALF_ESCR1	
0FH .....	See Table 2-45
MSR_ANY_CORE_C0	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
MSR_ANY_GFXE_C0	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
MSR_BO_PMON_BOX_CTRL	
06_2EH .....	See Table 2-16
MSR_BO_PMON_BOX_OVF_CTRL	
06_2EH .....	See Table 2-16
MSR_BO_PMON_BOX_STATUS	
06_2EH .....	See Table 2-16
MSR_BO_PMON_CTRO	
06_2EH .....	See Table 2-16
MSR_BO_PMON_CTR1	
06_2EH .....	See Table 2-16
MSR_BO_PMON_CTR2	
06_2EH .....	See Table 2-16
MSR_BO_PMON_CTR3	
06_2EH .....	See Table 2-16
MSR_BO_PMON_EVNT_SELO	
06_2EH .....	See Table 2-16
MSR_BO_PMON_EVNT_SEL1	
06_2EH .....	See Table 2-16

## MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_BO_PMON_EVNT_SEL2 06_2EH .....	See Table 2-16
MSR_BO_PMON_EVNT_SEL3 06_2EH .....	See Table 2-16
MSR_BO_PMON_MASK 06_2EH .....	See Table 2-16
MSR_BO_PMON_MATCH 06_2EH .....	See Table 2-16
MSR_B1_PMON_BOX_CTRL 06_2EH .....	See Table 2-16
MSR_B1_PMON_BOX_OVF_CTRL 06_2EH .....	See Table 2-16
MSR_B1_PMON_BOX_STATUS 06_2EH .....	See Table 2-16
MSR_B1_PMON_CTRL0 06_2EH .....	See Table 2-16
MSR_B1_PMON_CTRL1 06_2EH .....	See Table 2-16
MSR_B1_PMON_CTRL2 06_2EH .....	See Table 2-16
MSR_B1_PMON_CTRL3 06_2EH .....	See Table 2-16
MSR_B1_PMON_EVNT_SELO 06_2EH .....	See Table 2-16
MSR_B1_PMON_EVNT_SEL1 06_2EH .....	See Table 2-16
MSR_B1_PMON_EVNT_SEL2 06_2EH .....	See Table 2-16
MSR_B1_PMON_EVNT_SEL3 06_2EH .....	See Table 2-16
MSR_B1_PMON_MASK 06_2EH .....	See Table 2-16
MSR_B1_PMON_MATCH 06_2EH .....	See Table 2-16
MSR_BBL_CR_CTL 06_09H .....	See Table 2-49
MSR_BBL_CR_CTL3 06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_0EH .....	See Table 2-48
06_09H .....	See Table 2-49

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_BPU_CCCR0	
OFH .....	See Table 2-45
MSR_BPU_CCCR1	
OFH .....	See Table 2-45
MSR_BPU_CCCR2	
OFH .....	See Table 2-45
MSR_BPU_CCCR3	
OFH .....	See Table 2-45
MSR_BPU_COUNTER0	
OFH .....	See Table 2-45
MSR_BPU_COUNTER1	
OFH .....	See Table 2-45
MSR_BPU_COUNTER2	
OFH .....	See Table 2-45
MSR_BPU_COUNTER3	
OFH .....	See Table 2-45
MSR_BPU_ESCR0	
OFH .....	See Table 2-45
MSR_BPU_ESCR1	
OFH .....	See Table 2-45
MSR_BR_DETECT_COUNTER_CONFIG_j	
06_66H.....	See Table 2-41
MSR_BR_DETECT_CTRL	
06_66H.....	See Table 2-41
MSR_BR_DETECT_STATUS	
06_66H.....	See Table 2-41
MSR_BSU_ESCR0	
OFH .....	See Table 2-45
MSR_BSU_ESCR1	
OFH .....	See Table 2-45
MSR_CO_PMON_BOX_CTRL	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_CO_PMON_BOX_FILTER	
06_2DH .....	See Table 2-23
MSR_CO_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_CO_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_CO_PMON_BOX_OVF_CTRL	

MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2EH .....	See Table 2-16
<b>MSR_CO_PMON_BOX_STATUS</b>	
06_2EH .....	See Table 2-16
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_CTR0</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_CTR1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_CTR2</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_CTR3</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_CTR4</b>	
06_2EH .....	See Table 2-16
<b>MSR_CO_PMON_CTR5</b>	
06_2EH .....	See Table 2-16
<b>MSR_CO_PMON_EVNT_SELO</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_EVNT_SEL1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_CTR1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_EVNT_SEL2</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_CTR2</b>	
06_2EH .....	See Table 2-16



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_EVNT_SEL3</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_CO_PMON_EVNT_SEL4</b>	
06_2EH .....	See Table 2-16
<b>MSR_CO_PMON_EVNT_SEL5</b>	
06_2EH .....	See Table 2-16
<b>MSR_C1_PMON_BOX_CTRL</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C1_PMON_BOX_FILTER</b>	
06_2DH .....	See Table 2-23
<b>MSR_C1_PMON_BOX_FILTER0</b>	
06_3FH .....	See Table 2-32
<b>MSR_C1_PMON_BOX_FILTER1</b>	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C1_PMON_BOX_OVF_CTRL</b>	
06_2EH .....	See Table 2-16
<b>MSR_C1_PMON_BOX_STATUS</b>	
06_2EH .....	See Table 2-16
06_3FH .....	See Table 2-32
<b>MSR_C1_PMON_CTRL0</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C1_PMON_CTRL1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C1_PMON_CTRL2</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C1_PMON_CTRL3</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C1_PMON_CTR4	
06_2EH .....	See Table 2-16
MSR_C1_PMON_CTR5	
06_2EH .....	See Table 2-16
MSR_C1_PMON_EVNT_SEL0	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C1_PMON_EVNT_SEL1	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C1_PMON_EVNT_SEL2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C1_PMON_EVNT_SEL3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C1_PMON_EVNT_SEL4	
06_2EH .....	See Table 2-16
MSR_C1_PMON_EVNT_SEL5	
06_2EH .....	See Table 2-16
MSR_C10_PMON_BOX_FILTER	
06_3EH .....	See Table 2-27
MSR_C10_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C10_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C11_PMON_BOX_FILTER	
06_3EH .....	See Table 2-27
MSR_C11_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C11_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C12_PMON_BOX_FILTER	
06_3EH .....	See Table 2-27
MSR_C12_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_C12_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C13_PMON_BOX_FILTER	
06_3EH .....	See Table 2-27
MSR_C13_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C13_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C14_PMON_BOX_FILTER	
06_3EH .....	See Table 2-27
MSR_C14_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C14_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C15_PMON_BOX_CTL	
06_3FH .....	See Table 2-32
MSR_C15_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C15_PMON_BOX_FILTER1	
06_3FH .....	See Table 2-32
MSR_C15_PMON_BOX_STATUS	
06_3FH .....	See Table 2-32
MSR_C15_PMON_CTR0	
06_3FH .....	See Table 2-32
MSR_C15_PMON_CTR1	
06_3FH .....	See Table 2-32
MSR_C15_PMON_CTR2	
06_3FH .....	See Table 2-32
MSR_C15_PMON_CTR3	
06_3FH .....	See Table 2-32
MSR_C15_PMON_EVNTSELO	
06_3FH .....	See Table 2-32
MSR_C15_PMON_EVNTSEL1	
06_3FH .....	See Table 2-32
MSR_C15_PMON_EVNTSEL2	
06_3FH .....	See Table 2-32
MSR_C15_PMON_EVNTSEL3	
06_3FH .....	See Table 2-32
MSR_C16_PMON_BOX_CTL	

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3FH .....	See Table 2-32
MSR_C16_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C16_PMON_BOX_FILTER1	
06_3FH .....	See Table 2-32
MSR_C16_PMON_BOX_STATUS	
06_3FH .....	See Table 2-32
MSR_C16_PMON_CTR0	
06_3FH .....	See Table 2-32
MSR_C16_PMON_CTR3	
06_3FH .....	See Table 2-32
MSR_C16_PMON_CTR2	
06_3FH .....	See Table 2-32
MSR_C16_PMON_CTR3	
06_3FH .....	See Table 2-32
MSR_C16_PMON_EVNTSELO	
06_3FH .....	See Table 2-32
MSR_C16_PMON_EVNTSEL1	
06_3FH .....	See Table 2-32
MSR_C16_PMON_EVNTSEL2	
06_3FH .....	See Table 2-32
MSR_C16_PMON_EVNTSEL3	
06_3FH .....	See Table 2-32
MSR_C17_PMON_BOX_CTL	
06_3FH .....	See Table 2-32
MSR_C17_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C17_PMON_BOX_FILTER1	
06_3FH .....	See Table 2-32
MSR_C17_PMON_BOX_STATUS	
06_3FH .....	See Table 2-32
MSR_C17_PMON_CTR0	
06_3FH .....	See Table 2-32
MSR_C17_PMON_CTR1	
06_3FH .....	See Table 2-32
MSR_C17_PMON_CTR2	
06_3FH .....	See Table 2-32
MSR_C17_PMON_CTR3	
06_3FH .....	See Table 2-32
MSR_C17_PMON_EVNTSELO	
06_3FH .....	See Table 2-32
MSR_C17_PMON_EVNTSEL1	

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3FH .....	See Table 2-32
MSR_C17_PMON_EVNTSEL2	
06_3FH .....	See Table 2-32
MSR_C17_PMON_EVNTSEL3	
06_3FH .....	See Table 2-32
MSR_C2_PMON_BOX_CTRL	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C2_PMON_BOX_FILTER	
06_2DH .....	See Table 2-23
MSR_C2_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C2_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C2_PMON_BOX_OVF_CTRL	
06_2EH .....	See Table 2-16
MSR_C2_PMON_BOX_STATUS	
06_2EH .....	See Table 2-16
06_3FH .....	See Table 2-32
MSR_C2_PMON_CTRL0	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C2_PMON_CTRL1	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C2_PMON_CTRL2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C2_PMON_CTRL3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C2_PMON_CTRL4	
06_2EH .....	See Table 2-16
MSR_C2_PMON_CTRL5	
06_2EH .....	See Table 2-16
MSR_C2_PMON_EVNT_SELO	

## MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C2_PMON_EVNT_SEL1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C2_PMON_EVNT_SEL2</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C2_PMON_EVNT_SEL3</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C2_PMON_EVNT_SEL4</b>	
06_2EH .....	See Table 2-16
<b>MSR_C2_PMON_EVNT_SEL5</b>	
06_2EH .....	See Table 2-16
<b>MSR_C3_PMON_BOX_CTRL</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C3_PMON_BOX_FILTER</b>	
06_2DH .....	See Table 2-23
<b>MSR_C3_PMON_BOX_FILTER0</b>	
06_3FH .....	See Table 2-32
<b>MSR_C3_PMON_BOX_FILTER1</b>	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C3_PMON_BOX_OVF_CTRL</b>	
06_2EH .....	See Table 2-16
<b>MSR_C3_PMON_BOX_STATUS</b>	
06_2EH .....	See Table 2-16
06_3FH .....	See Table 2-32
<b>MSR_C3_PMON_CTRL0</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C3_PMON_CTRL1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3FH .....	See Table 2-32
MSR_C3_PMON_CTR2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C3_PMON_CTR3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C3_PMON_CTR4	
06_2EH .....	See Table 2-16
MSR_C3_PMON_CTR5	
06_2EH .....	See Table 2-16
MSR_C3_PMON_EVNT_SEL0	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C3_PMON_EVNT_SEL1	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C3_PMON_EVNT_SEL2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C3_PMON_EVNT_SEL3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C3_PMON_EVNT_SEL4	
06_2EH .....	See Table 2-16
MSR_C3_PMON_EVNT_SEL5	
06_2EH .....	See Table 2-16
MSR_C4_PMON_BOX_CTRL	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C4_PMON_BOX_FILTER	
06_2DH .....	See Table 2-23
MSR_C4_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C4_PMON_BOX_FILTER1	

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C4_PMON_BOX_OVF_CTRL	
06_2EH .....	See Table 2-16
MSR_C4_PMON_BOX_STATUS	
06_2EH .....	See Table 2-16
06_3FH .....	See Table 2-32
MSR_C4_PMON_CTRL0	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C4_PMON_CTRL1	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C4_PMON_CTRL2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C4_PMON_CTRL3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C4_PMON_CTRL4	
06_2EH .....	See Table 2-16
MSR_C4_PMON_CTRL5	
06_2EH .....	See Table 2-16
MSR_C4_PMON_EVNT_SELO	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C4_PMON_EVNT_SEL1	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C4_PMON_EVNT_SEL2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C4_PMON_EVNT_SEL3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3FH .....	See Table 2-32
MSR_C4_PMON_EVNT_SEL4	
06_2EH .....	See Table 2-16
MSR_C4_PMON_EVNT_SEL5	
06_2EH .....	See Table 2-16
MSR_C5_PMON_BOX_CTRL	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C5_PMON_BOX_FILTER	
06_2DH .....	See Table 2-23
MSR_C5_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C5_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C5_PMON_BOX_OVF_CTRL	
06_2EH .....	See Table 2-16
MSR_C5_PMON_BOX_STATUS	
06_2EH .....	See Table 2-16
06_3FH .....	See Table 2-32
MSR_C5_PMON_CTRL0	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C5_PMON_CTRL1	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C5_PMON_CTRL2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C5_PMON_CTRL3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C5_PMON_CTRL4	
06_2EH .....	See Table 2-16
MSR_C5_PMON_CTRL5	
06_2EH .....	See Table 2-16
MSR_C5_PMON_EVNT_SELO	

## MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C5_PMON_EVNT_SEL1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C5_PMON_EVNT_SEL2</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C5_PMON_EVNT_SEL3</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C5_PMON_EVNT_SEL4</b>	
06_2EH .....	See Table 2-16
<b>MSR_C5_PMON_EVNT_SEL5</b>	
06_2EH .....	See Table 2-16
<b>MSR_C6_PMON_BOX_CTRL</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C6_PMON_BOX_FILTER</b>	
06_2DH .....	See Table 2-23
<b>MSR_C6_PMON_BOX_FILTER0</b>	
06_3FH .....	See Table 2-32
<b>MSR_C6_PMON_BOX_FILTER1</b>	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C6_PMON_BOX_OVF_CTRL</b>	
06_2EH .....	See Table 2-16
<b>MSR_C6_PMON_BOX_STATUS</b>	
06_2EH .....	See Table 2-16
06_3FH .....	See Table 2-32
<b>MSR_C6_PMON_CTRL0</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C6_PMON_CTRL1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3FH .....	See Table 2-32
MSR_C6_PMON_CTR2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C6_PMON_CTR3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C6_PMON_CTR4	
06_2EH .....	See Table 2-16
MSR_C6_PMON_CTR5	
06_2EH .....	See Table 2-16
MSR_C6_PMON_EVNT_SELO	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C6_PMON_EVNT_SEL1	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C6_PMON_EVNT_SEL2	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C6_PMON_EVNT_SEL3	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C6_PMON_EVNT_SEL4	
06_2EH .....	See Table 2-16
MSR_C6_PMON_EVNT_SEL5	
06_2EH .....	See Table 2-16
MSR_C7_PMON_BOX_CTRL	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
MSR_C7_PMON_BOX_FILTER	
06_2DH .....	See Table 2-23
MSR_C7_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C7_PMON_BOX_FILTER1	

## MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_BOX_OVF_CTRL</b>	
06_2EH .....	See Table 2-16
<b>MSR_C7_PMON_BOX_STATUS</b>	
06_2EH .....	See Table 2-16
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_CTRL0</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_CTRL1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_CTRL2</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_CTRL3</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_CTRL4</b>	
06_2EH .....	See Table 2-16
<b>MSR_C7_PMON_CTRL5</b>	
06_2EH .....	See Table 2-16
<b>MSR_C7_PMON_EVNT_SELO</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_EVNT_SEL1</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_EVNT_SEL2</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23
06_3FH .....	See Table 2-32
<b>MSR_C7_PMON_EVNT_SEL3</b>	
06_2EH .....	See Table 2-16
06_2DH .....	See Table 2-23

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3FH .....	See Table 2-32
MSR_C7_PMON_EVNT_SEL4	
06_2EH .....	See Table 2-16
MSR_C7_PMON_EVNT_SEL5	
06_2EH .....	See Table 2-16
MSR_C8_PMON_BOX_CTRL	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C8_PMON_BOX_FILTER	
06_3EH .....	See Table 2-27
MSR_C8_PMON_BOX_FILTER0	
06_3FH .....	See Table 2-32
MSR_C8_PMON_BOX_FILTER1	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C8_PMON_BOX_OVF_CTRL	
06_2FH .....	See Table 2-18
MSR_C8_PMON_BOX_STATUS	
06_2FH .....	See Table 2-18
06_3FH .....	See Table 2-32
MSR_C8_PMON_CTRL0	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C8_PMON_CTRL1	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C8_PMON_CTRL2	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C8_PMON_CTRL3	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
MSR_C8_PMON_CTRL4	
06_2FH .....	See Table 2-18
MSR_C8_PMON_CTRL5	
06_2FH .....	See Table 2-18
MSR_C8_PMON_EVNT_SEL0	

MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C8_PMON_EVNT_SEL1</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C8_PMON_EVNT_SEL2</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C8_PMON_EVNT_SEL3</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C8_PMON_EVNT_SEL4</b>	
06_2FH .....	See Table 2-18
<b>MSR_C8_PMON_EVNT_SEL5</b>	
06_2FH .....	See Table 2-18
<b>MSR_C9_PMON_BOX_CTRL</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_BOX_FILTER</b>	
06_3EH .....	See Table 2-27
<b>MSR_C9_PMON_BOX_FILTER0</b>	
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_BOX_FILTER1</b>	
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_BOX_OVF_CTRL</b>	
06_2FH .....	See Table 2-18
<b>MSR_C9_PMON_BOX_STATUS</b>	
06_2FH .....	See Table 2-18
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_CTRL0</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_CTRL1</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_CTR2</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_CTR3</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_CTR4</b>	
06_2FH .....	See Table 2-18
<b>MSR_C9_PMON_CTR5</b>	
06_2FH .....	See Table 2-18
<b>MSR_C9_PMON_EVNT_SELO</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_EVNT_SEL1</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_EVNT_SEL2</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_EVNT_SEL3</b>	
06_2FH .....	See Table 2-18
06_3EH .....	See Table 2-27
06_3FH .....	See Table 2-32
<b>MSR_C9_PMON_EVNT_SEL4</b>	
06_2FH .....	See Table 2-18
<b>MSR_C9_PMON_EVNT_SEL5</b>	
06_2FH .....	See Table 2-18
<b>MSR_CC6_DEMOTION_POLICY_CONFIG</b>	
06_37H .....	See Table 2-9
<b>MSR_CONFIG_TDP_CONTROL</b>	
06_3AH .....	See Table 2-24
06_3CH, 06_45H, 06_46H .....	See Table 2-28
06_57H .....	See Table 2-43
<b>MSR_CONFIG_TDP_LEVEL1</b>	
06_3AH .....	See Table 2-24
06_3CH, 06_45H, 06_46H .....	See Table 2-28

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_57H .....	See Table 2-43
<b>MSR_CONFIG_TDP_LEVEL2</b>	
06_3AH .....	See Table 2-24
06_3CH, 06_45H, 06_46H .....	See Table 2-28
06_57H .....	See Table 2-43
<b>MSR_CONFIG_TDP_NOMINAL</b>	
06_3AH .....	See Table 2-24
06_3CH, 06_45H, 06_46H .....	See Table 2-28
06_57H .....	See Table 2-43
<b>MSR_CORE_C1_RESIDENCY</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH .....	See Table 2-6
06_66H .....	See Table 2-41
<b>MSR_CORE_C3_RESIDENCY</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH .....	See Table 2-14
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H .....	See Table 2-19
<b>MSR_CORE_C6_RESIDENCY</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH .....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH .....	See Table 2-14
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H .....	See Table 2-19
06_57H .....	See Table 2-43
<b>MSR_CORE_C7_RESIDENCY</b>	
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H .....	See Table 2-19
<b>MSR_CORE_GFXE_OVERLAP_CO</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_CORE_HDC_RESIDENCY</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_CORE_PERF_LIMIT_REASONS</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_3F .....	See Table 2-31
06_56H, 06_4FH .....	See Table 2-35
06_57H .....	See Table 2-43
<b>MSR_CORE_THREAD_COUNT</b>	
06_3FH .....	See Table 2-31
<b>MSR_CRU_ESCR0</b>	
0FH .....	See Table 2-45
<b>MSR_CRU_ESCR1</b>	
0FH .....	See Table 2-45
<b>MSR_CRU_ESCR2</b>	
0FH .....	See Table 2-45
<b>MSR_CRU_ESCR3</b>	



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
0FH .....	See Table 2-45
<b>MSR_CRU_ESCR4</b>	
0FH .....	See Table 2-45
<b>MSR_CRU_ESCR5</b>	
0FH .....	See Table 2-45
<b>MSR_DAC_ESCR0</b>	
0FH .....	See Table 2-45
<b>MSR_DAC_ESCR1</b>	
0FH .....	See Table 2-45
<b>MSR_DRAM_ENERGY_STATUS</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_2DH .....	See Table 2-22
06_3EH, 06_3FH .....	See Table 2-25
06_3CH, 06_45H, 06_46H .....	See Table 2-28
06_3F .....	See Table 2-31
06_56H, 06_4FH .....	See Table 2-35
06_57H .....	See Table 2-43
<b>MSR_DRAM_PERF_STATUS</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_2DH .....	See Table 2-22
06_3EH, 06_3FH .....	See Table 2-25
06_3CH, 06_45H, 06_46H .....	See Table 2-28
06_3F .....	See Table 2-31
06_56H, 06_4FH .....	See Table 2-35
06_57H .....	See Table 2-43
<b>MSR_DRAM_POWER_INFO</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_2DH .....	See Table 2-22
06_3EH, 06_3FH .....	See Table 2-25
06_3F .....	See Table 2-31
06_56H, 06_4FH .....	See Table 2-35
06_57H .....	See Table 2-43
<b>MSR_DRAM_POWER_LIMIT</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_2DH .....	See Table 2-22
06_3EH, 06_3FH .....	See Table 2-25
06_3F .....	See Table 2-31
06_56H, 06_4FH .....	See Table 2-35
06_57H .....	See Table 2-43
<b>MSR_EBC_FREQUENCY_ID</b>	
0FH .....	See Table 2-45
<b>MSR_EBC_HARD_POWERON</b>	

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
0FH .....	See Table 2-45
MSR_EBC_SOFT_POWERON	
0FH .....	See Table 2-45
MSR_EBL_CR_POWERON	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH .....	See Table 2-6
06_0EH .....	See Table 2-48
06_09H .....	See Table 2-49
MSR_EFSB_DRDY0	
0F_03H, 0F_04H .....	See Table 2-46
MSR_EFSB_DRDY1	
0F_03H, 0F_04H .....	See Table 2-46
MSR_EMON_L3_CTR_CTL0	
06_0FH, 06_17H .....	See Table 2-3
0F_06H .....	See Table 2-47
MSR_EMON_L3_CTR_CTL1	
06_0FH, 06_17H .....	See Table 2-3
0F_06H .....	See Table 2-47
MSR_EMON_L3_CTR_CTL2	
06_0FH, 06_17H .....	See Table 2-3
0F_06H .....	See Table 2-47
MSR_EMON_L3_CTR_CTL3	
06_0FH, 06_17H .....	See Table 2-3
0F_06H .....	See Table 2-47
MSR_EMON_L3_CTR_CTL4	
06_0FH, 06_17H .....	See Table 2-3
0F_06H .....	See Table 2-47
MSR_EMON_L3_CTR_CTL5	
06_0FH, 06_17H .....	See Table 2-3
0F_06H .....	See Table 2-47
MSR_EMON_L3_CTR_CTL6	
06_0FH, 06_17H .....	See Table 2-3
0F_06H .....	See Table 2-47
MSR_EMON_L3_CTR_CTL7	
06_0FH, 06_17H .....	See Table 2-3
0F_06H .....	See Table 2-47
MSR_EMON_L3_GL_CTL	
06_0FH, 06_17H .....	See Table 2-3
MSR_ERROR_CONTROL	
06_2DH .....	See Table 2-22
06_3EH .....	See Table 2-25

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3F .....	See Table 2-31
<b>MSR_FEATURE_CONFIG</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH .....	See Table 2-6
06_25H, 06_2CH .....	See Table 2-17
06_2FH .....	See Table 2-18
06_2AH, 06_2DH .....	See Table 2-19
06_57H .....	See Table 2-43
<b>MSR_FIRM_ESCRO</b>	
0FH .....	See Table 2-45
<b>MSR_FIRM_ESCR1</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_CCCRO</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_CCCR1</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_CCCR2</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_CCCR3</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_COUNTER0</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_COUNTER1</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_COUNTER2</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_COUNTER3</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_ESCRO</b>	
0FH .....	See Table 2-45
<b>MSR_FLAME_ESCR1</b>	
0FH .....	See Table 2-45
<b>MSR_FSB_ESCRO</b>	
0FH .....	See Table 2-45
<b>MSR_FSB_ESCR1</b>	
0FH .....	See Table 2-45
<b>MSR_FSB_FREQ</b>	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_4CH .....	See Table 2-11
06_0EH .....	See Table 2-48
<b>MSR_GQ_SNOOP_MESF</b>	

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH .....	See Table 2-15
MSR_GRAPHICS_PERF_LIMIT_REASONS	
06_3CH, 06_45H, 06_46H .....	See Table 2-29
MSR_IFSB_BUSQ0	
0F_03H, 0F_04H .....	See Table 2-46
MSR_IFSB_BUSQ1	
0F_03H, 0F_04H .....	See Table 2-46
MSR_IFSB_CNTR7	
0F_03H, 0F_04H .....	See Table 2-46
MSR_IFSB_CTL6	
0F_03H, 0F_04H .....	See Table 2-46
MSR_IFSB_SNPQ0	
0F_03H, 0F_04H .....	See Table 2-46
MSR_IFSB_SNPQ1	
0F_03H, 0F_04H .....	See Table 2-46
MSR_IQ_CCCRO	
0FH .....	See Table 2-45
MSR_IQ_CCCR1	
0FH .....	See Table 2-45
MSR_IQ_CCCR2	
0FH .....	See Table 2-45
MSR_IQ_CCCR3	
0FH .....	See Table 2-45
MSR_IQ_CCCR4	
0FH .....	See Table 2-45
MSR_IQ_CCCR5	
0FH .....	See Table 2-45
MSR_IQ_COUNTER0	
0FH .....	See Table 2-45
MSR_IQ_COUNTER1	
0FH .....	See Table 2-45
MSR_IQ_COUNTER2	
0FH .....	See Table 2-45
MSR_IQ_COUNTER3	
0FH .....	See Table 2-45
MSR_IQ_COUNTER4	
0FH .....	See Table 2-45
MSR_IQ_COUNTER5	
0FH .....	See Table 2-45
MSR_IQ_ESCR0	
0FH .....	See Table 2-45
MSR_IQ_ESCR1	

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
0FH .....	See Table 2-45
<b>MSR_IS_ESCR0</b>	
0FH .....	See Table 2-45
<b>MSR_IS_ESCR1</b>	
0FH .....	See Table 2-45
<b>MSR_ITLB_ESCR0</b>	
0FH .....	See Table 2-45
<b>MSR_ITLB_ESCR1</b>	
0FH .....	See Table 2-45
<b>MSR_IX_ESCR0</b>	
0FH .....	See Table 2-45
<b>MSR_IX_ESCR1</b>	
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_0</b>	
0FH .....	See Table 2-45
06_0EH .....	See Table 2-48
06_09H .....	See Table 2-49
<b>MSR_LASTBRANCH_0_FROM_IP</b>	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH .....	See Table 2-12
06_7AH .....	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_0_TO_IP</b>	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH .....	See Table 2-12
06_7AH .....	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_1_FROM_IP</b>	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_1_TO_IP</b>	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_10_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_10_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_11_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_11_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_12_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_12_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_13_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_13_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_14_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_14_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_15_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_15_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_16_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_16_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_17_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_17_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_18_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
<b>MSR_LASTBRANCH_18_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_19_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_19_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_2</b>	
0FH .....	See Table 2-45
06_0EH .....	See Table 2-48
06_09H .....	See Table 2-49
<b>MSR_LASTBRANCH_2_FROM_IP</b>	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_2_TO_IP</b>	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_20_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_20_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_21_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_21_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_22_FROM_IP</b>	



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_22_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_23_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_23_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_24_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_24_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_25_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_25_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_26_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_26_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_27_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_27_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_28_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_28_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
<b>MSR_LASTBRANCH_29_FROM_IP</b>	

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_LASTBRANCH_29_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_LASTBRANCH_3	
0FH	See Table 2-45
06_0EH	See Table 2-48
06_09H	See Table 2-49
MSR_LASTBRANCH_3_FROM_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-14
06_2AH, 06_2DH	See Table 2-19
0FH	See Table 2-45
MSR_LASTBRANCH_3_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-14
06_2AH, 06_2DH	See Table 2-19
0FH	See Table 2-45
MSR_LASTBRANCH_30_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_LASTBRANCH_30_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_LASTBRANCH_31_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_LASTBRANCH_31_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_LASTBRANCH_4	
06_0EH	See Table 2-48
06_09H	See Table 2-49
MSR_LASTBRANCH_4_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_4_TO_IP</b>	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_5</b>	
06_0EH .....	See Table 2-48
06_09H .....	See Table 2-49
<b>MSR_LASTBRANCH_5_FROM_IP</b>	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_5_TO_IP</b>	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_6</b>	
06_0EH .....	See Table 2-48
06_09H .....	See Table 2-49
<b>MSR_LASTBRANCH_6_FROM_IP</b>	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_6_TO_IP</b>	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_7</b>	
06_0EH .....	See Table 2-48
06_09H .....	See Table 2-49
<b>MSR_LASTBRANCH_7_FROM_IP</b>	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_7_TO_IP</b>	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_8_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_8_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_9_FROM_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_9_TO_IP</b>	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
0FH .....	See Table 2-45
<b>MSR_LASTBRANCH_TOS</b>	

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH .....	See Table 2-7
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_57H .....	See Table 2-43
06_0EH .....	See Table 2-48
06_09H .....	See Table 2-49
<b>MSR_LASTBRANCH_INFO_0</b>	
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_1</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_10</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_11</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_12</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_13</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_14</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_15</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_16</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_17</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13
<b>MSR_LBR_INFO_18</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
06_7AH .....	See Table 2-13

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LBR_INFO_19	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_2	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_20	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_21	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_22	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_23	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_24	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_25	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_26	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_27	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_28	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_29	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_3	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
MSR_LBR_INFO_30	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
<b>MSR_LBR_INFO_31</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
<b>MSR_LBR_INFO_4</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
<b>MSR_LBR_INFO_5</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
<b>MSR_LBR_INFO_6</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
<b>MSR_LBR_INFO_7</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
<b>MSR_LBR_INFO_8</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
<b>MSR_LBR_INFO_9</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
06_7AH.....	See Table 2-13
<b>MSR_LBR_SELECT</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
06_3CH, 06_45H, 06_46H.....	See Table 2-28
06_57H.....	See Table 2-43
<b>MSR_LER_FROM_LIP</b>	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
06_57H.....	See Table 2-43
0FH.....	See Table 2-45
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
<b>MSR_LER_TO_LIP</b>	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6

## MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_2AH, 06_2DH .....	See Table 2-19
06_57H.....	See Table 2-43
0FH.....	See Table 2-45
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
<b>MSR_MO_PMON_ADDR_MASK</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_ADDR_MATCH</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_BOX_CTRL</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_BOX_OVF_CTRL</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_BOX_STATUS</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_CTRL0</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_CTRL1</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_CTRL2</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_CTRL3</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_CTRL4</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_CTRL5</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_DSP</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_EVNT_SELO</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_EVNT_SEL1</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_EVNT_SEL2</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_EVNT_SEL3</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_EVNT_SEL4</b>	
06_2EH.....	See Table 2-16
<b>MSR_MO_PMON_EVNT_SEL5</b>	
06_2EH.....	See Table 2-16



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_M0_PMON_ISS 06_2EH.....	See Table 2-16
MSR_M0_PMON_MAP 06_2EH.....	See Table 2-16
MSR_M0_PMON_MM_CONFIG 06_2EH.....	See Table 2-16
MSR_M0_PMON_MSC_THR 06_2EH.....	See Table 2-16
MSR_M0_PMON_PGT 06_2EH.....	See Table 2-16
MSR_M0_PMON_PLD 06_2EH.....	See Table 2-16
MSR_M0_PMON_TIMESTAMP 06_2EH.....	See Table 2-16
MSR_M0_PMON_ZDP 06_2EH.....	See Table 2-16
MSR_M1_PMON_ADDR_MASK 06_2EH.....	See Table 2-16
MSR_M1_PMON_ADDR_MATCH 06_2EH.....	See Table 2-16
MSR_M1_PMON_BOX_CTRL 06_2EH.....	See Table 2-16
MSR_M1_PMON_BOX_OVF_CTRL 06_2EH.....	See Table 2-16
MSR_M1_PMON_BOX_STATUS 06_2EH.....	See Table 2-16
MSR_M1_PMON_CTR0 06_2EH.....	See Table 2-16
MSR_M1_PMON_CTR1 06_2EH.....	See Table 2-16
MSR_M1_PMON_CTR2 06_2EH.....	See Table 2-16
MSR_M1_PMON_CTR3 06_2EH.....	See Table 2-16
MSR_M1_PMON_CTR4 06_2EH.....	See Table 2-16
MSR_M1_PMON_CTR5 06_2EH.....	See Table 2-16
MSR_M1_PMON_DSP 06_2EH.....	See Table 2-16
MSR_M1_PMON_EVNT_SELO 06_2EH.....	See Table 2-16

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_M1_PMON_EVNT_SEL1 06_2EH.....	See Table 2-16
MSR_M1_PMON_EVNT_SEL2 06_2EH.....	See Table 2-16
MSR_M1_PMON_EVNT_SEL3 06_2EH.....	See Table 2-16
MSR_M1_PMON_EVNT_SEL4 06_2EH.....	See Table 2-16
MSR_M1_PMON_EVNT_SEL5 06_2EH.....	See Table 2-16
MSR_M1_PMON_ISS 06_2EH.....	See Table 2-16
MSR_M1_PMON_MAP 06_2EH.....	See Table 2-16
MSR_M1_PMON_MM_CONFIG 06_2EH.....	See Table 2-16
MSR_M1_PMON_MSC_THR 06_2EH.....	See Table 2-16
MSR_M1_PMON_PGT 06_2EH.....	See Table 2-16
MSR_M1_PMON_PLD 06_2EH.....	See Table 2-16
MSR_M1_PMON_TIMESTAMP 06_2EH.....	See Table 2-16
MSR_M1_PMON_ZDP 06_2EH.....	See Table 2-16
IA32_MCO_MISC / MSR_MCO_MISC 06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
MSR_MCO_RESIDENCY 06_57H.....	See Table 2-43
IA32_MC1_MISC / MSR_MC1_MISC 06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
IA32_MC10_ADDR / MSR_MC10_ADDR 06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
IA32_MC10_CTL / MSR_MC10_CTL 06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC10_MISC / MSR_MC10_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC10_STATUS / MSR_MC10_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC11_ADDR / MSR_MC11_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC11_CTL / MSR_MC11_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC11_MISC / MSR_MC11_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC11_STATUS / MSR_MC11_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37

MSR Name and CPUID DisplayFamily_DisplayModel	Location
<b>IA32_MC12_ADDR / MSR_MC12_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC12_CTL / MSR_MC12_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC12_MISC / MSR_MC12_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC12_STATUS / MSR_MC12_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC13_ADDR / MSR_MC13_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC13_CTL / MSR_MC13_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC13_MISC / MSR_MC13_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
<b>IA32_MC13_STATUS / MSR_MC13_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC14_ADDR / MSR_MC14_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC14_CTL / MSR_MC14_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC14_MISC / MSR_MC14_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC14_STATUS / MSR_MC14_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC15_ADDR / MSR_MC15_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC15_CTL / MSR_MC15_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37

MSR Name and CPUID DisplayFamily_DisplayModel	Location
<b>IA32_MC15_MISC / MSR_MC15_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC15_STATUS / MSR_MC15_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC16_ADDR / MSR_MC16_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC16_CTL / MSR_MC16_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC16_MISC / MSR_MC16_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC16_STATUS / MSR_MC16_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC17_ADDR / MSR_MC17_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_4FH.....	See Table 2-37
<b>IA32_MC17_CTL / MSR_MC17_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC17_MISC / MSR_MC17_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC17_STATUS / MSR_MC17_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC18_ADDR / MSR_MC18_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC18_CTL / MSR_MC18_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC18_MISC / MSR_MC18_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_4FH.....	See Table 2-37
<b>IA32_MC18_STATUS / MSR_MC18_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC19_ADDR / MSR_MC19_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC19_CTL / MSR_MC19_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC19_MISC / MSR_MC19_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC19_STATUS / MSR_MC19_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC2_MISC / MSR_MC2_MISC</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
<b>IA32_MC20_ADDR / MSR_MC20_ADDR</b>	
06_2EH.....	See Table 2-16
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_4FH.....	See Table 2-37
IA32_MC20_CTL / MSR_MC20_CTL	
06_2EH.....	See Table 2-16
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
IA32_MC20_MISC / MSR_MC20_MISC	
06_2EH.....	See Table 2-16
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
IA32_MC20_STATUS / MSR_MC20_STATUS	
06_2EH.....	See Table 2-16
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
IA32_MC21_ADDR / MSR_MC21_ADDR	
06_2EH.....	See Table 2-16
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4F.....	See Table 2-37
IA32_MC21_CTL / MSR_MC21_CTL	
06_2EH.....	See Table 2-16
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4F.....	See Table 2-37
IA32_MC21_MISC / MSR_MC21_MISC	
06_2EH.....	See Table 2-16
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4F.....	See Table 2-37
IA32_MC21_STATUS / MSR_MC21_STATUS	
06_2EH.....	See Table 2-16
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4F.....	See Table 2-37
IA32_MC22_ADDR / MSR_MC22_ADDR	
06_3EH.....	See Table 2-25
IA32_MC22_CTL / MSR_MC22_CTL	
06_3EH.....	See Table 2-25
IA32_MC22_MISC / MSR_MC22_MISC	
06_3EH.....	See Table 2-25

## MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
IA32_MC22_STATUS / MSR_MC22_STATUS 06_3EH.....	See Table 2-25
IA32_MC23_ADDR / MSR_MC23_ADDR 06_3EH.....	See Table 2-25
IA32_MC23_CTL / MSR_MC23_CTL 06_3EH.....	See Table 2-25
IA32_MC23_MISC / MSR_MC23_MISC 06_3EH.....	See Table 2-25
IA32_MC23_STATUS / MSR_MC23_STATUS 06_3EH.....	See Table 2-25
IA32_MC24_ADDR / MSR_MC24_ADDR 06_3EH.....	See Table 2-25
IA32_MC24_CTL / MSR_MC24_CTL 06_3EH.....	See Table 2-25
IA32_MC24_MISC / MSR_MC24_MISC 06_3EH.....	See Table 2-25
IA32_MC24_STATUS / MSR_MC24_STATUS 06_3EH.....	See Table 2-25
IA32_MC25_ADDR / MSR_MC25_ADDR 06_3EH.....	See Table 2-25
IA32_MC25_CTL / MSR_MC25_CTL 06_3EH.....	See Table 2-25
IA32_MC25_MISC / MSR_MC25_MISC 06_3EH.....	See Table 2-25
IA32_MC25_STATUS / MSR_MC25_STATUS 06_3EH.....	See Table 2-25
IA32_MC26_ADDR / MSR_MC26_ADDR 06_3EH.....	See Table 2-25
IA32_MC26_CTL / MSR_MC26_CTL 06_3EH.....	See Table 2-25
IA32_MC26_MISC / MSR_MC26_MISC 06_3EH.....	See Table 2-25
IA32_MC26_STATUS / MSR_MC26_STATUS 06_3EH.....	See Table 2-25
IA32_MC27_ADDR / MSR_MC27_ADDR 06_3EH.....	See Table 2-25
IA32_MC27_CTL / MSR_MC27_CTL 06_3EH.....	See Table 2-25
IA32_MC27_MISC / MSR_MC27_MISC 06_3EH.....	See Table 2-25
IA32_MC27_STATUS / MSR_MC27_STATUS 06_3EH.....	See Table 2-25

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
IA32_MC28_ADDR / MSR_MC28_ADDR 06_3EH.....	See Table 2-25
IA32_MC28_CTL / MSR_MC28_CTL 06_3EH.....	See Table 2-25
IA32_MC28_MISC / MSR_MC28_MISC 06_3EH.....	See Table 2-25
IA32_MC28_STATUS / MSR_MC28_STATUS 06_3EH.....	See Table 2-25
IA32_MC29_ADDR / MSR_MC29_ADDR 06_3EH.....	See Table 2-26
IA32_MC29_CTL / MSR_MC29_CTL 06_3EH.....	See Table 2-26
IA32_MC29_MISC / MSR_MC29_MISC 06_3EH.....	See Table 2-26
IA32_MC29_STATUS / MSR_MC29_STATUS 06_3EH.....	See Table 2-26
IA32_MC3_ADDR / MSR_MC3_ADDR 06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH .....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_57H.....	See Table 2-43
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
IA32_MC3_CTL / MSR_MC3_CTL 06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH .....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_57H.....	See Table 2-43
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
IA32_MC3_MISC / MSR_MC3_MISC 06_0FH, 06_17H .....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_0EH.....	See Table 2-48
IA32_MC3_STATUS / MSR_MC3_STATUS 06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH .....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
06_57H.....	See Table 2-43

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
IA32_MC30_ADDR / MSR_MC30_ADDR	
06_3EH.....	See Table 2-26
IA32_MC30_CTL / MSR_MC30_CTL	
06_3EH.....	See Table 2-26
IA32_MC30_MISC / MSR_MC30_MISC	
06_3EH.....	See Table 2-26
IA32_MC30_STATUS / MSR_MC30_STATUS	
06_3EH.....	See Table 2-26
IA32_MC31_ADDR / MSR_MC31_ADDR	
06_3EH.....	See Table 2-26
IA32_MC31_CTL / MSR_MC31_CTL	
06_3EH.....	See Table 2-26
IA32_MC31_MISC / MSR_MC31_MISC	
06_3EH.....	See Table 2-26
IA32_MC31_STATUS / MSR_MC31_STATUS	
06_3EH.....	See Table 2-26
IA32_MC4_ADDR / MSR_MC4_ADDR	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_57H.....	See Table 2-43
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
IA32_MC4_CTL / MSR_MC4_CTL	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_57H.....	See Table 2-43
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
IA32_MC4_CTL2 / MSR_MC4_CTL2	
06_2AH, 06_2DH.....	See Table 2-19
IA32_MC4_STATUS / MSR_MC4_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_57H.....	See Table 2-43

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
<b>MSR_MC5_ADDR / MSR_MC5_ADDR</b>	
06_0FH, 06_17H.....	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3FH.....	See Table 2-31
06_4FH.....	See Table 2-37
06_57H.....	See Table 2-43
06_0EH.....	See Table 2-48
<b>IA32_MC5_CTL / MSR_MC5_CTL</b>	
06_0FH, 06_17H.....	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3FH.....	See Table 2-31
06_4FH.....	See Table 2-37
06_57H.....	See Table 2-43
06_0EH.....	See Table 2-48
<b>IA32_MC5_MISC / MSR_MC5_MISC</b>	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3FH.....	See Table 2-31
06_4FH.....	See Table 2-37
06_0EH.....	See Table 2-48
<b>IA32_MC5_STATUS / MSR_MC5_STATUS</b>	
06_0FH, 06_17H.....	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3FH.....	See Table 2-31
06_4FH.....	See Table 2-37
06_57H.....	See Table 2-43
06_0EH.....	See Table 2-48
<b>IA32_MC6_ADDR / MSR_MC6_ADDR</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC6_CTL / MSR_MC6_CTL</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>MSR_MC6_DEMOTION_POLICY_CONFIG</b>	
06_37H.....	See Table 2-9
<b>IA32_MC6_MISC / MSR_MC6_MISC</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>MSR_MC6_RESIDENCY_COUNTER</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_37H.....	See Table 2-9
06_57H.....	See Table 2-43
<b>IA32_MC6_STATUS / MSR_MC6_STATUS</b>	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3FH.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC7_ADDR / MSR_MC7_ADDR</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC7_CTL / MSR_MC7_CTL</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC7_MISC / MSR_MC7_MISC</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC7_STATUS / MSR_MC7_STATUS</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC8_ADDR / MSR_MC8_ADDR</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC8_CTL / MSR_MC8_CTL</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC8_MISC / MSR_MC8_MISC</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_4FH.....	See Table 2-37
<b>IA32_MC8_STATUS / MSR_MC8_STATUS</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_4FH.....	See Table 2-37
<b>IA32_MC9_ADDR / MSR_MC9_ADDR</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC9_CTL / MSR_MC9_CTL</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC9_MISC / MSR_MC9_MISC</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>IA32_MC9_STATUS / MSR_MC9_STATUS</b>	
06_2EH.....	See Table 2-16
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25
06_3F.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-36
06_4FH.....	See Table 2-37
<b>MSR_MCG_MISC</b>	
0FH.....	See Table 2-45
<b>MSR_MCG_R10</b>	
0FH.....	See Table 2-45
<b>MSR_MCG_R11</b>	
0FH.....	See Table 2-45
<b>MSR_MCG_R12</b>	
0FH.....	See Table 2-45
<b>MSR_MCG_R13</b>	
0FH.....	See Table 2-45
<b>MSR_MCG_R14</b>	
0FH.....	See Table 2-45
<b>MSR_MCG_R15</b>	



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
OFH.....	See Table 2-45
MSR_MCG_R8	
OFH.....	See Table 2-45
MSR_MCG_R9	
OFH.....	See Table 2-45
MSR_MCG_RAX	
OFH.....	See Table 2-45
MSR_MCG_RBP	
OFH.....	See Table 2-45
MSR_MCG_RBX	
OFH.....	See Table 2-45
MSR_MCG_RCX	
OFH.....	See Table 2-45
MSR_MCG_RDI	
OFH.....	See Table 2-45
MSR_MCG_RDX	
OFH.....	See Table 2-45
MSR_MCG_RESERVED1 - MSR_MCG_RESERVED5	
OFH.....	See Table 2-45
MSR_MCG_RFLAGS	
OFH.....	See Table 2-45
MSR_MCG_RIP	
OFH.....	See Table 2-45
MSR_MCG_RSI	
OFH.....	See Table 2-45
MSR_MCG_RSP	
OFH.....	See Table 2-45
MSR_MISC_FEATURE_CONTROL	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
MSR_MISC_PWR_MGMT	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
MSR_MOB_ESCRO	
OFH.....	See Table 2-45
MSR_MOB_ESCR1	
OFH.....	See Table 2-45
MSR_MS_CCCRO	
OFH.....	See Table 2-45
MSR_MS_CCCR1	

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
0FH.....	See Table 2-45
MSR_MS_CCCR2	
0FH.....	See Table 2-45
MSR_MS_CCCR3	
0FH.....	See Table 2-45
MSR_MS_COUNTER0	
0FH.....	See Table 2-45
MSR_MS_COUNTER1	
0FH.....	See Table 2-45
MSR_MS_COUNTER2	
0FH.....	See Table 2-45
MSR_MS_COUNTER3	
0FH.....	See Table 2-45
MSR_MS_ESCRO	
0FH.....	See Table 2-45
MSR_MS_ESCR1	
0FH.....	See Table 2-45
MSR_MTRRCAP	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
MSR_OFFCORE_RSP_0	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
06_57H.....	See Table 2-43
MSR_OFFCORE_RSP_1	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_25H, 06_2CH.....	See Table 2-17
06_2FH.....	See Table 2-18
06_2AH, 06_2DH.....	See Table 2-19
06_57H.....	See Table 2-43
MSR_PCIE_PLL_RATIO	
06_3FH.....	See Table 2-31
MSR_PCU_PMON_BOX_CTL	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
MSR_PCU_PMON_BOX_FILTER	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
MSR_PCU_PMON_BOX_STATUS	
06_3EH.....	See Table 2-27
06_3FH.....	See Table 2-32
MSR_PCU_PMON_CTRL0	

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
<b>MSR_PCU_PMON_CTR1</b>	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
<b>MSR_PCU_PMON_CTR2</b>	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
<b>MSR_PCU_PMON_CTR3</b>	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
<b>MSR_PCU_PMON_EVNTSELO</b>	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
<b>MSR_PCU_PMON_EVNTSEL1</b>	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
<b>MSR_PCU_PMON_EVNTSEL2</b>	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
<b>MSR_PCU_PMON_EVNTSEL3</b>	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
<b>MSR_PEBS_ENABLE</b>	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH.....	See Table 2-12
06_7AH.....	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
06_3EH.....	See Table 2-26
06_57H.....	See Table 2-43
0FH.....	See Table 2-45
<b>MSR_PEBS_FRONTEND</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
<b>MSR_PEBS_LD_LAT</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
<b>MSR_PEBS_MATRIX_VERT</b>	
0FH.....	See Table 2-45
<b>MSR_PEBS_NUM_ALT</b>	

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH.....	See Table 2-22
MSR_PERF_CAPABILITIES	
06_0FH, 06_17H .....	See Table 2-3
MSR_PERF_FIXED_CTR_CTRL	
06_0FH, 06_17H .....	See Table 2-3
MSR_PERF_FIXED_CTR0	
06_0FH, 06_17H .....	See Table 2-3
MSR_PERF_FIXED_CTR1	
06_0FH, 06_17H .....	See Table 2-3
MSR_PERF_FIXED_CTR2	
06_0FH, 06_17H .....	See Table 2-3
MSR_PERF_GLOBAL_CTRL	
06_0FH, 06_17H .....	See Table 2-3
MSR_PERF_GLOBAL_OVF_CTRL	
06_0FH, 06_17H .....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
MSR_PERF_GLOBAL_STATUS	
06_0FH, 06_17H .....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH .....	See Table 2-14
MSR_PERF_STATUS	
06_0FH, 06_17H .....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H .....	See Table 2-4
06_2AH, 06_2DH .....	See Table 2-19
MSR_PKG_C10_RESIDENCY	
06_5CH, 06_7AH .....	See Table 2-12
06_45H.....	See Table 2-29 and Table 2-30
06_4FH.....	See Table 2-37
MSR_PKG_C2_RESIDENCY	
06_27H.....	See Table 2-5
06_5CH, 06_7AH .....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H .....	See Table 2-19
06_57H.....	See Table 2-43
MSR_PKG_C3_RESIDENCY	
06_5CH, 06_7AH .....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH .....	See Table 2-14
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H .....	See Table 2-19
06_66H.....	See Table 2-41
06_57H.....	See Table 2-43
MSR_PKG_C4_RESIDENCY	
06_27H.....	See Table 2-5
MSR_PKG_C6_RESIDENCY	

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_27H.....	See Table 2-5
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH.....	See Table 2-14
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-19
06_57H.....	See Table 2-43
<b>MSR_PKG_C7_RESIDENCY</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH.....	See Table 2-14
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-19
06_57H.....	See Table 2-43
<b>MSR_PKG_C8_RESIDENCY</b>	
06_45H.....	See Table 2-30
06_4FH.....	See Table 2-37
<b>MSR_PKG_C9_RESIDENCY</b>	
06_45H.....	See Table 2-30
06_4FH.....	See Table 2-37
<b>MSR_PKG_CST_CONFIG_CONTROL</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_4CH.....	See Table 2-11
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
06_3AH.....	See Table 2-24
06_3EH.....	See Table 2-25
06_3CH, 06_45H, 06_46H.....	See Table 2-29
06_45H.....	See Table 2-30
06_3F.....	See Table 2-31
06_3DH.....	See Table 2-34
06_56H, 06_4FH.....	See Table 2-35
06_57H.....	See Table 2-43
<b>MSR_PKG_ENERGY_STATUS</b>	
06_37H, 06_4AH, 06_5AH, 06_5DH.....	See Table 2-8
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-19
<b>MSR_PKG_HDC_CONFIG</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
<b>MSR_PKG_HDC_DEEP_RESIDENCY</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
<b>MSR_PKG_HDC_SHALLOW_RESIDENCY</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
<b>MSR_PKG_PERF_STATUS</b>	
06_5CH, 06_7AH.....	See Table 2-12

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2DH.....	See Table 2-22
06_3EH, 06_3FH.....	See Table 2-25
06_3CH, 06_45H, 06_46H.....	See Table 2-29
06_57H.....	See Table 2-43
<b>MSR_PKG_POWER_INFO</b>	
06_4DH.....	See Table 2-10
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-19
06_57H.....	See Table 2-43
<b>MSR_PKG_POWER_LIMIT</b>	
06_37H, 06_4AH, 06_5AH, 06_5DH.....	See Table 2-8
06_4DH.....	See Table 2-10
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-19
06_57H.....	See Table 2-43
<b>MSR_PKGC_IRTL1</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-28
<b>MSR_PKGC_IRTL2</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-28
<b>MSR_PKGC3_IRTL</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH.....	See Table 2-19
<b>MSR_PKGC6_IRTL</b>	
06_2AH, 06_2DH.....	See Table 2-19
<b>MSR_PKGC7_IRTL</b>	
06_2AH.....	See Table 2-20
<b>MSR_PLATFORM_BRV</b>	
0FH.....	See Table 2-45
<b>MSR_PLATFORM_ENERGY_COUNTER</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
<b>MSR_PLATFORM_ID</b>	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
<b>MSR_PLATFORM_INFO</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3AH.....	See Table 2-24
06_3EH.....	See Table 2-25
06_3CH, 06_45H, 06_46H.....	See Table 2-28 and Table 2-29
06_56H, 06_4FH.....	See Table 2-35
06_57H.....	See Table 2-43
<b>MSR_PLATFORM_POWER_LIMIT</b>	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
<b>MSR_PMG_IO_CAPTURE_BASE</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_4CH.....	See Table 2-11
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
06_3AH.....	See Table 2-24
06_3EH.....	See Table 2-25
06_57H.....	See Table 2-43
<b>MSR_PMH_ESCRO</b>	
0FH.....	See Table 2-45
<b>MSR_PMH_ESCR1</b>	
0FH.....	See Table 2-45
<b>MSR_PMON_GLOBAL_CONFIG</b>	
06_3EH.....	See Table 2-27
06_3FH.....	See Table 2-32
<b>MSR_PMON_GLOBAL_CTL</b>	
06_3EH.....	See Table 2-27
06_3FH.....	See Table 2-32
<b>MSR_PMON_GLOBAL_STATUS</b>	
06_3EH.....	See Table 2-27
06_3FH.....	See Table 2-32
<b>MSR_POWER_CTL</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
<b>MSR_PPO_ENERGY_STATUS</b>	
06_37H, 06_4AH, 06_5AH, 06_5DH.....	See Table 2-8
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-19
06_57H.....	See Table 2-43
<b>MSR_PPO_POLICY</b>	
06_2AH, 06_45H.....	See Table 2-20
<b>MSR_PPO_POWER_LIMIT</b>	
06_4CH.....	See Table 2-11

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-19
06_57H	See Table 2-43
MSR_PP1_ENERGY_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_45H	See Table 2-20
06_3CH, 06_45H, 06_46H	See Table 2-29
MSR_PP1_POLICY	
06_2AH, 06_45H	See Table 2-20
06_3CH, 06_45H, 06_46H	See Table 2-29
MSR_PP1_POWER_LIMIT	
06_2AH, 06_45H	See Table 2-20
06_3CH, 06_45H, 06_46H	See Table 2-29
MSR_PPERF	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_PPIN	
06_3EH	See Table 2-25
06_56H, 06_4FH	See Table 2-35
MSR_PPIN_CTL	
06_3EH	See Table 2-25
06_56H, 06_4FH	See Table 2-35
MSR_PRMRR_PHYS_BASE	
06_8EH, 06_9EH	See Table 2-40
MSR_PRMRR_PHYS_MASK	
06_8EH, 06_9EH	See Table 2-40
MSR_PRMRR_VALID_CONFIG	
06_8EH, 06_9EH	See Table 2-40
MSR_RING_RATIO_LIMIT	
06_8EH, 06_9EH	See Table 2-40
MSR_RO_PMON_BOX_CTRL	
06_2EH	See Table 2-16
MSR_RO_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-16
MSR_RO_PMON_BOX_STATUS	
06_2EH	See Table 2-16
MSR_RO_PMON_CTRL0	
06_2EH	See Table 2-16
MSR_RO_PMON_CTRL1	
06_2EH	See Table 2-16
MSR_RO_PMON_CTRL2	
06_2EH	See Table 2-16
MSR_RO_PMON_CTRL3	
06_2EH	See Table 2-16



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_RO_PMON_CTR4 06_2EH.....	See Table 2-16
MSR_RO_PMON_CTR5 06_2EH.....	See Table 2-16
MSR_RO_PMON_CTR6 06_2EH.....	See Table 2-16
MSR_RO_PMON_CTR7 06_2EH.....	See Table 2-16
MSR_RO_PMON_EVNT_SELO 06_2EH.....	See Table 2-16
MSR_RO_PMON_EVNT_SEL1 06_2EH.....	See Table 2-16
MSR_RO_PMON_EVNT_SEL2 06_2EH.....	See Table 2-16
MSR_RO_PMON_EVNT_SEL3 06_2EH.....	See Table 2-16
MSR_RO_PMON_EVNT_SEL4 06_2EH.....	See Table 2-16
MSR_RO_PMON_EVNT_SEL5 06_2EH.....	See Table 2-16
MSR_RO_PMON_EVNT_SEL6 06_2EH.....	See Table 2-16
MSR_RO_PMON_EVNT_SEL7 06_2EH.....	See Table 2-16
MSR_RO_PMON_IPERFO_P0 06_2EH.....	See Table 2-16
MSR_RO_PMON_IPERFO_P1 06_2EH.....	See Table 2-16
MSR_RO_PMON_IPERFO_P2 06_2EH.....	See Table 2-16
MSR_RO_PMON_IPERFO_P3 06_2EH.....	See Table 2-16
MSR_RO_PMON_IPERFO_P4 06_2EH.....	See Table 2-16
MSR_RO_PMON_IPERFO_P5 06_2EH.....	See Table 2-16
MSR_RO_PMON_IPERFO_P6 06_2EH.....	See Table 2-16
MSR_RO_PMON_IPERFO_P7 06_2EH.....	See Table 2-16
MSR_RO_PMON_QLX_P0 06_2EH.....	See Table 2-16

## MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_R0_PMON_QLX_P1 06_2EH.....	See Table 2-16
MSR_R0_PMON_QLX_P2 06_2EH.....	See Table 2-16
MSR_R0_PMON_QLX_P3 06_2EH.....	See Table 2-16
MSR_R1_PMON_BOX_CTRL 06_2EH.....	See Table 2-16
MSR_R1_PMON_BOX_OVF_CTRL 06_2EH.....	See Table 2-16
MSR_R1_PMON_BOX_STATUS 06_2EH.....	See Table 2-16
MSR_R1_PMON_CTR10 06_2EH.....	See Table 2-16
MSR_R1_PMON_CTR11 06_2EH.....	See Table 2-16
MSR_R1_PMON_CTR12 06_2EH.....	See Table 2-16
MSR_R1_PMON_CTR13 06_2EH.....	See Table 2-16
MSR_R1_PMON_CTR14 06_2EH.....	See Table 2-16
MSR_R1_PMON_CTR15 06_2EH.....	See Table 2-16
MSR_R1_PMON_CTR8 06_2EH.....	See Table 2-16
MSR_R1_PMON_CTR9 06_2EH.....	See Table 2-16
MSR_R1_PMON_EVNT_SEL10 06_2EH.....	See Table 2-16
MSR_R1_PMON_EVNT_SEL11 06_2EH.....	See Table 2-16
MSR_R1_PMON_EVNT_SEL12 06_2EH.....	See Table 2-16
MSR_R1_PMON_EVNT_SEL13 06_2EH.....	See Table 2-16
MSR_R1_PMON_EVNT_SEL14 06_2EH.....	See Table 2-16
MSR_R1_PMON_EVNT_SEL15 06_2EH.....	See Table 2-16
MSR_R1_PMON_EVNT_SEL8 06_2EH.....	See Table 2-16

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_R1_PMON_EVNT_SEL9 06_2EH.....	See Table 2-16
MSR_R1_PMON_IPERF1_P10 06_2EH.....	See Table 2-16
MSR_R1_PMON_IPERF1_P11 06_2EH.....	See Table 2-16
MSR_R1_PMON_IPERF1_P12 06_2EH.....	See Table 2-16
MSR_R1_PMON_IPERF1_P13 06_2EH.....	See Table 2-16
MSR_R1_PMON_IPERF1_P14 06_2EH.....	See Table 2-16
MSR_R1_PMON_IPERF1_P15 06_2EH.....	See Table 2-16
MSR_R1_PMON_IPERF1_P8 06_2EH.....	See Table 2-16
MSR_R1_PMON_IPERF1_P9 06_2EH.....	See Table 2-16
MSR_R1_PMON_QLX_P4 06_2EH.....	See Table 2-16
MSR_R1_PMON_QLX_P5 06_2EH.....	See Table 2-16
MSR_R1_PMON_QLX_P6 06_2EH.....	See Table 2-16
MSR_R1_PMON_QLX_P7 06_2EH.....	See Table 2-16
MSR_RAPL_POWER_UNIT 06_37H, 06_4AH, 06_5AH, 06_5DH.....	See Table 2-8
06_4DH.....	See Table 2-10
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-19
06_3FH.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-35
06_57H.....	See Table 2-43
MSR_RAT_ESCR0 0FH.....	See Table 2-45
MSR_RAT_ESCR1 0FH.....	See Table 2-45
MSR_RING_PERF_LIMIT_REASONS 06_3CH, 06_45H, 06_46H.....	See Table 2-29
MSR_S0_PMON_BOX_CTRL 06_2EH.....	See Table 2-16

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3FH.....	See Table 2-32
MSR_SO_PMON_BOX_FILTER	
06_3FH.....	See Table 2-32
MSR_SO_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-16
MSR_SO_PMON_BOX_STATUS	
06_2EH.....	See Table 2-16
MSR_SO_PMON_CTRL0	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_SO_PMON_CTRL1	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_SO_PMON_CTRL2	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_SO_PMON_CTRL3	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_SO_PMON_EVNT_SELO	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_SO_PMON_EVNT_SEL1	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_SO_PMON_EVNT_SEL2	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_SO_PMON_EVNT_SEL3	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_SO_PMON_MASK	
06_2EH.....	See Table 2-16
MSR_SO_PMON_MATCH	
06_2EH.....	See Table 2-16
MSR_S1_PMON_BOX_CTRL	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_BOX_FILTER	
06_3FH.....	See Table 2-32
MSR_S1_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-16

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_S1_PMON_BOX_STATUS	
06_2EH.....	See Table 2-16
MSR_S1_PMON_CTRL0	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_CTRL1	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_CTRL2	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_CTRL3	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_EVTN_SEL0	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_EVTN_SEL1	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_EVTN_SEL2	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_EVTN_SEL3	
06_2EH.....	See Table 2-16
06_3FH.....	See Table 2-32
MSR_S1_PMON_MASK	
06_2EH.....	See Table 2-16
MSR_S1_PMON_MATCH	
06_2EH.....	See Table 2-16
MSR_S2_PMON_BOX_CTL	
06_3FH.....	See Table 2-32
MSR_S2_PMON_BOX_FILTER	
06_3FH.....	See Table 2-32
MSR_S2_PMON_CTRL0	
06_3FH.....	See Table 2-32
MSR_S2_PMON_CTRL1	
06_3FH.....	See Table 2-32
MSR_S2_PMON_CTRL2	
06_3FH.....	See Table 2-32
MSR_S2_PMON_CTRL3	
06_3FH.....	See Table 2-32

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_S2_PMON_EVTNSELO 06_3FH.....	See Table 2-32
MSR_S2_PMON_EVTNSEL1 06_3FH.....	See Table 2-32
MSR_S2_PMON_EVTNSEL2 06_3FH.....	See Table 2-32
MSR_S2_PMON_EVTNSEL3 06_3FH.....	See Table 2-32
MSR_S3_PMON_BOX_CTL 06_3FH.....	See Table 2-32
MSR_S3_PMON_BOX_FILTER 06_3FH.....	See Table 2-32
MSR_S3_PMON_CTRL0 06_3FH.....	See Table 2-32
MSR_S3_PMON_CTRL1 06_3FH.....	See Table 2-32
MSR_S3_PMON_CTRL2 06_3FH.....	See Table 2-32
MSR_S3_PMON_CTRL3 06_3FH.....	See Table 2-32
MSR_S3_PMON_EVTNSELO 06_3FH.....	See Table 2-32
MSR_S3_PMON_EVTNSEL1 06_3FH.....	See Table 2-32
MSR_S3_PMON_EVTNSEL2 06_3FH.....	See Table 2-32
MSR_S3_PMON_EVTNSEL3 06_3FH.....	See Table 2-32
MSR_SAAT_ESCR0 0FH.....	See Table 2-45
MSR_SAAT_ESCR1 0FH.....	See Table 2-45
MSR_SGXOWNEREP0CH0 06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
MSR_SGXOWNEREP0CH1 06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H.....	See Table 2-38
MSR_SMI_COUNT 06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_57H.....	See Table 2-43
<b>MSR_SMM_BLOCKED</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-29
<b>MSR_SMM_DELAYED</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-29
<b>MSR_SMM_FEATURE_CONTROL</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-29
<b>MSR_SMM_MCA_CAP</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-29
06_3FH.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-35
06_57H.....	See Table 2-43
<b>MSR_SMRR_PHYSBASE</b>	
06_0FH, 06_17H.....	See Table 2-3
<b>MSR_SMRR_PHYSMASK</b>	
06_0FH, 06_17H.....	See Table 2-3
<b>MSR_SSU_ESCR0</b>	
0FH.....	See Table 2-45
<b>MSR_TBPU_ESCR0</b>	
0FH.....	See Table 2-45
<b>MSR_TBPU_ESCR1</b>	
0FH.....	See Table 2-45
<b>MSR_TC_ESCR0</b>	
0FH.....	See Table 2-45
<b>MSR_TC_ESCR1</b>	
0FH.....	See Table 2-45
<b>MSR_TC_PRECISE_EVENT</b>	
0FH.....	See Table 2-45
<b>MSR_TEMPERATURE_TARGET</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
06_2AH, 06_2DH.....	See Table 2-19
06_3EH.....	See Table 2-25
06_56H, 06_4FH.....	See Table 2-35
06_57H.....	See Table 2-43
<b>MSR_THERM2_CTL</b>	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
0FH.....	See Table 2-45
06_0EH.....	See Table 2-48
06_09H.....	See Table 2-49
<b>MSR_THREAD_ID_INFO</b>	
06_3FH.....	See Table 2-31
<b>MSR_TRACE_HUB_STH ACPIBAR_BASE</b>	
06_8EH, 06_9EH.....	See Table 2-40
<b>MSR_TURBO_ACTIVATION_RATIO</b>	
06_5CH, 06_7AH.....	See Table 2-12
06_3AH.....	See Table 2-24
06_3CH, 06_45H, 06_46H.....	See Table 2-28
06_57H.....	See Table 2-43
<b>MSR_TURBO_GROUP_CORECNT</b>	
06_5CH, 06_7AH.....	See Table 2-12
<b>MSR_TURBO_POWER_CURRENT_LIMIT</b>	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-14
<b>MSR_TURBO_RATIO_LIMIT</b>	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_4DH.....	See Table 2-10
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH.....	See Table 2-14
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH.....	See Table 2-15
06_2EH.....	See Table 2-16
06_25H, 06_2CH.....	See Table 2-17
06_2FH.....	See Table 2-18
06_2AH, 06_45H.....	See Table 2-20
06_2DH.....	See Table 2-22
06_3EH.....	See Table 2-25 and Table 2-26
06_3CH, 06_45H, 06_46H.....	See Table 2-29
06_3FH.....	See Table 2-31
06_3DH.....	See Table 2-34
06_56H, 06_4FH.....	See Table 2-35
06_55H.....	See Table 2-42
06_57H.....	See Table 2-43
<b>MSR_TURBO_RATIO_LIMIT1</b>	
06_3EH.....	See Table 2-25 and Table 2-26
06_3FH.....	See Table 2-31
06_56H, 06_4FH.....	See Table 2-35
<b>MSR_TURBO_RATIO_LIMIT2</b>	
06_3FH.....	See Table 2-31



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_TURBO_RATIO_LIMIT3	
06_56H.....	See Table 2-36
06_4FH.....	See Table 2-37
MSR_TURBO_RATIO_LIMIT_CORES	
06_55H.....	See Table 2-42
MSR_U_PMON_BOX_STATUS	
06_3EH.....	See Table 2-27
06_3FH.....	See Table 2-32
MSR_U_PMON_CTR	
06_2EH.....	See Table 2-16
MSR_U_PMON_CTR0	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
MSR_U_PMON_CTR1	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
MSR_U_PMON_EVNT_SEL	
06_2EH.....	See Table 2-16
MSR_U_PMON_EVNTSELO	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
MSR_U_PMON_EVNTSEL1	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
MSR_U_PMON_GLOBAL_CTRL	
06_2EH.....	See Table 2-16
MSR_U_PMON_GLOBAL_OVF_CTRL	
06_2EH.....	See Table 2-16
MSR_U_PMON_GLOBAL_STATUS	
06_2EH.....	See Table 2-16
MSR_U_PMON_UCLK_FIXED_CTL	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
MSR_U_PMON_UCLK_FIXED_CTR	
06_2DH.....	See Table 2-23
06_3FH.....	See Table 2-32
MSR_U2L_ESCR0	
0FH.....	See Table 2-45
MSR_U2L_ESCR1	
0FH.....	See Table 2-45
MSR_UNC_ARB_PERFCTRO	
06_2AH.....	See Table 2-21

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3CH, 06_45H, 06_46H	See Table 2-29
06_4EH, 06_5EH	See Table 2-39
MSR_UNC_ARB_PERFCTR1	
06_2AH	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-29
06_4EH, 06_5EH	See Table 2-39
MSR_UNC_ARB_PERFEVTSELO	
06_2AH	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-29
06_4EH, 06_5EH	See Table 2-39
MSR_UNC_ARB_PERFEVTSEL1	
06_2AH	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-29
06_4EH, 06_5EH	See Table 2-39
MSR_UNC_CBO_0_PERFCTR0	
06_2AH	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-29
06_4EH, 06_5EH	See Table 2-39
MSR_UNC_CBO_0_PERFCTR1	
06_2AH	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-29
06_4EH, 06_5EH	See Table 2-39
MSR_UNC_CBO_0_PERFCTR2	
06_2AH	See Table 2-21
MSR_UNC_CBO_0_PERFCTR3	
06_2AH	See Table 2-21
MSR_UNC_CBO_0_PERFEVTSELO	
06_2AH	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-29
06_4EH, 06_5EH	See Table 2-39
MSR_UNC_CBO_0_PERFEVTSEL1	
06_2AH	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-29
06_4EH, 06_5EH	See Table 2-39
MSR_UNC_CBO_0_PERFEVTSEL2	
06_2AH	See Table 2-21
MSR_UNC_CBO_0_PERFEVTSEL3	
06_2AH	See Table 2-21
MSR_UNC_CBO_0_UNIT_STATUS	
06_2AH	See Table 2-21
MSR_UNC_CBO_1_PERFCTR0	
06_2AH	See Table 2-21

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_1_PERFCTR1</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_1_PERFCTR2</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_1_PERFCTR3</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_1_PERFEVTSELO</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_1_PERFEVTSEL1</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_1_PERFEVTSEL2</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_1_PERFEVTSEL3</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_1_UNIT_STATUS</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_2_PERFCTR0</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_2_PERFCTR1</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_2_PERFCTR2</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_2_PERFCTR3</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_2_PERFEVTSELO</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_2_PERFEVTSEL1</b>	
06_2AH .....	See Table 2-21

## MODEL-SPECIFIC REGISTERS (MSRS)

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_2_PERFEVTSEL2</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_2_PERFEVTSEL3</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_2_UNIT_STATUS</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_3_PERFCTR0</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_3_PERFCTR1</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_3_PERFCTR2</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_3_PERFCTR3</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_3_PERFEVTSELO</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_3_PERFEVTSEL1</b>	
06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
<b>MSR_UNC_CBO_3_PERFEVTSEL2</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_3_PERFEVTSEL3</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_3_UNIT_STATUS</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_4_PERFCTR0</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_4_PERFCTR1</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_4_PERFCTR2</b>	
06_2AH .....	See Table 2-21
<b>MSR_UNC_CBO_4_PERFCTR3</b>	
06_2AH .....	See Table 2-21

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
MSR_UNC_CBO_4_PERFEVTSELO 06_2AH .....	See Table 2-21
MSR_UNC_CBO_4_PERFEVTSEL1 06_2AH .....	See Table 2-21
MSR_UNC_CBO_4_PERFEVTSEL2 06_2AH .....	See Table 2-21
MSR_UNC_CBO_4_PERFEVTSEL3 06_2AH .....	See Table 2-21
MSR_UNC_CBO_4_UNIT_STATUS 06_2AH .....	See Table 2-21
MSR_UNC_CBO_CONFIG 06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
MSR_UNC_PERF_FIXED_CTR 06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
MSR_UNC_PERF_FIXED_CTRL 06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
MSR_UNC_PERF_GLOBAL_CTRL 06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
MSR_UNC_PERF_GLOBAL_STATUS 06_2AH .....	See Table 2-21
06_3CH, 06_45H, 06_46H .....	See Table 2-29
06_4EH, 06_5EH .....	See Table 2-39
MSR_UNCORE_ADDR_OPCODE_MATCH 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH .....	See Table 2-15
MSR_UNCORE_FIXED_CTR_CTRL 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH .....	See Table 2-15
MSR_UNCORE_FIXED_CTRL0 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH .....	See Table 2-15
MSR_UNCORE_PERF_GLOBAL_CTRL 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH .....	See Table 2-15
MSR_UNCORE_PERF_GLOBAL_OVF_CTRL 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH .....	See Table 2-15
MSR_UNCORE_PERF_GLOBAL_STATUS 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH .....	See Table 2-15

## MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNCORE_PERFEVTSELO 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PERFEVTSEL1 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PERFEVTSEL2 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PERFEVTSEL3 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PERFEVTSEL4 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PERFEVTSEL5 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PERFEVTSEL6 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PERFEVTSEL7 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PMC0 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PMC1 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PMC2 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PMC3 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PMC4 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PMC5 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
06_2EH	See Table 2-16
MSR_UNCORE_PMC6 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PMC7 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-15
MSR_UNCORE_PRMRR_BASE 06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_UNCORE_PRMRR_MASK 06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H	See Table 2-38
MSR_UNCORE_PRMRR_PHYS_BASE 06_8EH, 06_9EH	See Table 2-40
MSR_UNCORE_PRMRR_PHYS_MASK 06_8EH, 06_9EH	See Table 2-40
MSR_W_PMON_BOX_CTRL	

<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
06_2EH .....	See Table 2-16
MSR_W_PMON_BOX_OVF_CTRL	
06_2EH .....	See Table 2-16
MSR_W_PMON_BOX_STATUS	
06_2EH .....	See Table 2-16
MSR_W_PMON_CTRL0	
06_2EH .....	See Table 2-16
MSR_W_PMON_CTRL1	
06_2EH .....	See Table 2-16
MSR_W_PMON_CTRL2	
06_2EH .....	See Table 2-16
MSR_W_PMON_CTRL3	
06_2EH .....	See Table 2-16
MSR_W_PMON_EVNT_SELO	
06_2EH .....	See Table 2-16
MSR_W_PMON_EVNT_SEL1	
06_2EH .....	See Table 2-16
MSR_W_PMON_EVNT_SEL2	
06_2EH .....	See Table 2-16
MSR_W_PMON_EVNT_SEL3	
06_2EH .....	See Table 2-16
MSR_W_PMON_FIXED_CTRL	
06_2EH .....	See Table 2-16
MSR_W_PMON_FIXED_CTRL_CTL	
06_2EH .....	See Table 2-16
MSR_WEIGHTED_CORE_CO	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H .....	See Table 2-38
MTRRfix16K_80000	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix16K_A0000	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix4K_C0000	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix4K_C8000	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix4K_D0000	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MTRRfix4K_D8000	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix4K_E0000	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix4K_E8000	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix4K_F0000	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix4K_F8000	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRfix64K_00000	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRphysBase0	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRphysBase1	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRphysBase2	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRphysBase3	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRphysBase4	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRphysBase5	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRphysBase6	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50
MTRRphysBase7	
06_OEH .....	See Table 2-48
P6 Family .....	See Table 2-50



<b>MSR Name and CPUID DisplayFamily_DisplayModel</b>	<b>Location</b>
<b>MTRRphysMask0</b>	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
<b>MTRRphysMask1</b>	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
<b>MTRRphysMask2</b>	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
<b>MTRRphysMask3</b>	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
<b>MTRRphysMask4</b>	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
<b>MTRRphysMask5</b>	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
<b>MTRRphysMask6</b>	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50
<b>MTRRphysMask7</b>	
06_0EH .....	See Table 2-48
P6 Family .....	See Table 2-50

