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

Documentation Changes

September 2013

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

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

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

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

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

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

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

Copyright © 1997-2013 Intel Corporation. All rights reserved.

Contents

Revision History

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

Revision	Description	Date
-024	• Removed Documentation Changes 1-21 • Added Documentation Changes 1-16	June 2009
-025	• Removed Documentation Changes 1-16 • Added Documentation Changes 1-18	September 2009
-026	• Removed Documentation Changes 1-18 • Added Documentation Changes 1-15	December 2009
-027	• Removed Documentation Changes 1-15 • Added Documentation Changes 1-24	March 2010
-028	• Removed Documentation Changes 1-24 • Added Documentation Changes 1-29	June 2010
-029	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	September 2010
-030	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	January 2011
-031	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	April 2011
-032	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-14	May 2011
-033	• Removed Documentation Changes 1-14 • Added Documentation Changes 1-38	October 2011
-034	• Removed Documentation Changes 1-38 • Added Documentation Changes 1-16	December 2011
-035	• Removed Documentation Changes 1-16 • Added Documentation Changes 1-18	March 2012
-036	• Removed Documentation Changes 1-18 • Added Documentation Changes 1-17	May 2012
-037	• Removed Documentation Changes 1-17 • Added Documentation Changes 1-28	August 2012
-038	• Removed Documentation Changes 1-28 • Add Documentation Changes 1-22	January 2013
-039	• Removed Documentation Changes 1-22 • Add Documentation Changes 1-17	June 2013
-040	• Removed Documentation Changes 1-17 • Add Documentation Changes 1-24	September 2013

§

Preface

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

Affected Documents

Document Title	Document Number/ Location
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture	253665
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M	253666
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z	253667
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference	326018
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1	253668
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2	253669
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3	326019

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 2, Volume 1
2	Updates to Chapter 3, Volume 1
3	Updates to Chapter 10, Volume 1
4	Updates to Chapter 11, Volume 1
5	New Chapter 13, Volume 1
6	Updates to Appendix C, Volume 1
7	Updates to Appendix E, Volume 1
8	Updates to Chapter 3, Volume 2A
9	Updates to Chapter 4, Volume 2B
10	Updates to Appendix A, Volume 2C
11	Updates to Chapter 2, Volume 3A
12	Updates to Chapter 5, Volume 3A
13	Updates to Chapter 6, Volume 3A
14	Updates to Chapter 7, Volume 3A
15	Updates to Chapter 10, Volume 3A
16	Updates to Chapter 13, Volume 3A
17	Updates to Chapter 14, Volume 3B
18	Updates to Chapter 15, Volume 3B
19	Updates to Chapter 16, Volume 3B
20	Updates to Chapter 17, Volume 3B
21	Updates to Chapter 18, Volume 3B
22	Updates to Chapter 31, Volume 3C
23	Updates to Chapter 34, Volume 3C
24	Updates to Chapter 35, Volume 3C

Documentation Changes

1. Updates to Chapter 2, Volume 1

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

...

2.2.10 Intel® 64 Architecture

Intel 64 architecture increases the linear address space for software to 64 bits and supports physical address space up to 46 bits. The technology also introduces a new operating mode referred to as IA-32e mode.

IA-32e mode operates in one of two sub-modes: (1) compatibility mode enables a 64-bit operating system to run most legacy 32-bit software unmodified, (2) 64-bit mode enables a 64-bit operating system to run applications written to access 64-bit address space.

In the 64-bit mode, applications may access:

- 64-bit flat linear addressing
- 8 additional general-purpose registers (GPRs)
- 8 additional registers for streaming SIMD extensions (SSE, SSE2, SSE3 and SSSE3)
- 64-bit-wide GPRs and instruction pointers
- uniform byte-register addressing
- fast interrupt-prioritization mechanism
- a new instruction-pointer relative-addressing mode

An Intel 64 architecture processor supports existing IA-32 software because it is able to run all non-64-bit legacy modes supported by IA-32 architecture. Most existing IA-32 applications also run in compatibility mode.

...

2. Updates to Chapter 3, Volume 1

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

...

3.2.1 64-Bit Mode Execution Environment

The execution environment for 64-bit mode is similar to that described in Section 3.2. The following paragraphs describe the differences that apply.

- **Address space** — A task or program running in 64-bit mode on an IA-32 processor can address linear address space of up to 2^{64} bytes (subject to the canonical addressing requirement described in Section 3.3.7.1) and physical address space of up to 2^{46} bytes. Software can query CPUID for the physical address size supported by a processor.

- **Basic program execution registers** — The number of general-purpose registers (GPRs) available is 16. GPRs are 64-bits wide and they support operations on byte, word, doubleword and quadword integers. Accessing byte registers is done uniformly to the lowest 8 bits. The instruction pointer register becomes 64 bits. The EFLAGS register is extended to 64 bits wide, and is referred to as the RFLAGS register. The upper 32 bits of RFLAGS is reserved. The lower 32 bits of RFLAGS is the same as EFLAGS. See Figure 3-2.
- **XMM registers** — There are 16 XMM data registers for SIMD operations. See Section 10.2, “SSE Programming Environment,” for more information about these registers.
- **Stack** — The stack pointer size is 64 bits. Stack size is not controlled by a bit in the SS descriptor (as it is in non-64-bit modes) nor can the pointer size be overridden by an instruction prefix.
- **Control registers** — Control registers expand to 64 bits. A new control register (the task priority register: CR8 or TPR) has been added. See Chapter 2, “Intel® 64 and IA-32 Architectures,” in this volume.
- **Debug registers** — Debug registers expand to 64 bits. See Chapter 17, “Debugging, Branch Profiles and Time-Stamp Counter,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.*

Descriptor table registers — The global descriptor table register (GDTR) and interrupt descriptor table register (IDTR) expand to 10 bytes so that they can hold a full 64-bit base address. The local descriptor table register (LDTR) and the task register (TR) also expand to hold a full 64-bit base address.

...

3.4.3.3 System Flags and IOPL Field

The system flags and IOPL field in the EFLAGS register control operating-system or executive operations. **They should not be modified by application programs**. The functions of the system flags are as follows:

TF (bit 8)	**Trap flag** — Set to enable single-step mode for debugging; clear to disable single-step mode.
IF (bit 9)	**Interrupt enable flag** — Controls the response of the processor to maskable interrupt requests. Set to respond to maskable interrupts; cleared to inhibit maskable interrupts.
IOPL (bits 12 and 13)	**I/O privilege level field** — Indicates the I/O privilege level of the currently running program or task. The current privilege level (CPL) of the currently running program or task must be less than or equal to the I/O privilege level to access the I/O address space. The POPF and IRET instructions can modify this field only when operating at a CPL of 0.
NT (bit 14)	**Nested task flag** — Controls the chaining of interrupted and called tasks. Set when the current task is linked to the previously executed task; cleared when the current task is not linked to another task.
RF (bit 16)	**Resume flag** — Controls the processor’s response to debug exceptions.
VM (bit 17)	**Virtual-8086 mode flag** — Set to enable virtual-8086 mode; clear to return to protected mode without virtual-8086 mode semantics.
AC (bit 18)	**Alignment check flag** — Set this flag and the AM bit in the CR0 register to enable alignment checking of memory references; clear the AC flag and/or the AM bit to disable alignment checking.
VIF (bit 19)	**Virtual interrupt flag** — Virtual image of the IF flag. Used in conjunction with the VIP flag. (To use this flag and the VIP flag the virtual mode extensions are enabled by setting the VME flag in control register CR4.)
VIP (bit 20)	**Virtual interrupt pending flag** — Set to indicate that an interrupt is pending; clear when no interrupt is pending. (Software sets and clears this flag; the processor only reads it.) Used in conjunction with the VIF flag.
ID (bit 21)	**Identification flag** — The ability of a program to set or clear this flag indicates support for the CPUID instruction.

For a detailed description of these flags: see Chapter 3, “Protected-Mode Memory Management,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.*

...

3. Updates to Chapter 10, Volume 1

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

...

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, multimedia 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 destination region is mapped as cacheable memory (write back [WB], write through [WT] or WC memory type), 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.[1]
- 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 corre-

---

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

sponding 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.

...

10.5 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 FPU register 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, along with the x87 FPU and MMX 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 support the saving and restoring of the x87 execution environment (**x87 state**) and the registers used by the streaming SIMD extensions (**SSE state**). They extend the instructions FSAVE/FNSAVE and FRSTOR, which can be used for the x87 state, to save and restore SSE state.

A processor enumerates support for the FXSAVE and FXRSTOR instructions using the CPUID instruction. Specifically, CPUID.1:EDX.FXSR[bit 24] enumerates support for FXSAVE and FXRSTOR. Software enables FXSAVE and FXRSTOR by setting CR4.OSFXSR[bit 9] to 1 (e.g., with the MOV to CR4 instruction). If this bit is 0, execution of either FXRSTOR or FXSAVE causes an invalid-opcode exception (#UD).

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

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

Table 10-2 Format of an FXSAVE Area (Contd.)

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

The x87 state component comprises bytes 23:0 and bytes 159:32. The SSE state component comprises bytes 31:24 and bytes 415:160. 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 ≤ *j* ≤ 7, FXSAVE saves a 0 into bit *j* of byte 4 if x87 FPU data register ST*j* has a empty tag; otherwise, FXSAVE saves a 1 into bit *j* of byte 4.
  - — For each *j*, 0 ≤ *j* ≤ 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[bit 13] = 0, bytes 13:12 are used for x87 FPU Instruction Pointer Selector (FPU CS). Otherwise, the processor deprecates the FPU CS value; XRSTOR ignores this field, and XSAVE and XSAVEOPT save 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[bit 13] = 0, bytes 21:20 are used for x87 FPU Data Pointer Selector (FPU DS). Otherwise, the processor deprecates the FPU DS value; XRSTOR ignores this field, and XSAVE and XSAVEOPT save 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 ST0–ST7 (MM0–MM7). Each of the 8 register is allocated a 128-bit region, with the low 80 bits used for the register and the upper 48 bits unused.

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 XMM0–XMM7.
- Bytes 415:288 are used for the registers XMM8–XMM15. 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 XMM8–XMM15.

FXSAVE and FXRSTOR can operate on SSE state only if CR4.OSFXSR = 1; 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 following conditions cause execution of the XSAVE instruction to generate a fault:

- If FXSAVE is not enabled (CR4.OSFXSR = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- If the address of the XSAVE area is not 16-byte aligned, a general-protection exception (#GP) occurs.

If none of these conditions cause a fault, 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. The following conditions cause execution of the FXRSTOR instruction to generate a fault:

- If FXRSTOR is not enabled (CR4.OSFXSR = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- If the address of the FXSAVE area is not 16-byte aligned, a general-protection exception (#GP) occurs.
- 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).

If none of these conditions cause a fault, the instruction loads x87 state and SSE state rom 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.

...

4. 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.

...

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.

...

11.5.2.3 Divide-By-Zero Exception (#Z)

The processor reports a divide-by-zero exception when a DIVPS, DIVSS, DIVPD or DIVSD instruction attempts to divide a finite non-zero operand by 0. The flag (ZE) and mask (ZM) bits for the divide-by-zero exception are bits 2 and 9, respectively, in the MXCSR register.

See Section 4.9.1.3, “Divide-By-Zero Exception (#Z),” for more information about the divide-by-zero exception. See Section 11.5.4, “Handling SIMD Floating-Point Exceptions in Software,” for information on handling unmasked exceptions.

The divide-by-zero exception is not affected by the flush-to-zero mode at a single-instruction boundary.

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 divide-by-zero exception - when observed for a given operation involving denormal inputs.

...

11.5.2.5 Numeric Underflow Exception (#U)

The processor reports a numeric underflow exception whenever the rounded result of an arithmetic instruction 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, CVTPD2PS, CVTSD2SS, 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 (results that trigger the underflow exception) 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, 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.

...

11.6.13 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. PREFETCH*h* 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.

PREFETCH*h* 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 PREFETCH*h* 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.

...

5. New Chapter 13, Volume 1

Chapter 13 was added to the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

CHAPTER 13
MANAGING STATE USING THE XSAVE FEATURE SET

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

The XSAVE feature set comprises five instructions. XGETBV and XSETBV allow software to read and write the extended control register XCR0, which controls the operation of the XSAVE feature set. XSAVE and XSAVEOPT are two instructions that save processor state to memory; XRSTOR is a corresponding instruction that loads processor state from memory.

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

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

Section 13.4 presents details of the XSAVE area and its organization. Section 13.5 describes in detail each of the XSAVE-supported state components.

Section 13.6, Section 13.7, and Section 13.8 describe the operation of XSAVE, XRSTOR, and XSAVEOPT, respectively.

13.1 XSAVE-MANAGED FEATURES AND STATE-COMPONENT BITMAPS

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

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

- Bit 0 corresponds to the state component used for the x87 FPU execution environment (**x87 state**). See Section 13.5.1

- Bit 1 corresponds to the state component used for registers used by the streaming SIMD extensions (**SSE state**). See Section 13.5.2.
- Bit 2 corresponds to the state component used for the additional register state used by the Intel® Advanced Vector Extensions (**AVX state**). See Section 13.5.3.

Bits 63:3 are not currently defined in state-component bitmaps and are reserved for future expansion.

The state component corresponding to bit *i* of state-component bitmaps is called **state component *i***. Thus, x87 state is state component 0; SSE state is state component 1; and AVX state is state component 2.

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

Extended control register XCR0 contains a state-component bitmap that specifies the state components that software has enabled the XSAVE feature set to manage. If the bit corresponding to a state component is clear in XCR0, no save or restore instruction in the XSAVE feature set will operate on that state component, regardless of the value of the instruction mask. Details of instruction operation are given in Section 13.6 through Section 13.8.

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

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

The state components for x87 state and for SSE state are XSAVE-managed but not XSAVE-enabled. The processors allows modification to this state, and it allows execution of the x87 FPU instructions and the SSE instructions, regardless of the value of CR4.OSXSAVE and XCR0.

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

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

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

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

      If EAX[*i*] = 1 (for *i* > 1), sub-function *i* enumerates details for state component *i* (see below).

    - ECX enumerates the size (in bytes) required for an XSAVE area containing all the state components supported by this processor (see Section 13.4).
    - EBX enumerates the size (in bytes) required for an XSAVE area containing all the state components corresponding to bits currently set in XCR0.
  - — CPUID function 0DH, sub-function 1.
    - EAX[0] enumerates support for the XSAVEOPT instruction. The instruction is supported if and only if this bit is 1. If EAX[0] = 0, execution of XSAVEOPT causes an invalid-opcode exception (#UD).
    - EAX[31:1], EBX, ECX, and EDX are reserved.
  - — CPUID function 0DH, sub-function *i* (*i* > 1). This sub-function enumerates details for state component *i*. If CPUID.(EAX=0DH,ECX=0):EAX[*i*] = 1, the following items provide specific details:
    - EAX enumerates the size (in bytes) required for state component *i*.
    - EBX enumerates the offset (in bytes, from the base of the XSAVE area) of the section used for state component *i*.
    - ECX and EDX are reserved.

    If the processor does not support state component *i* (CPUID.(EAX=0DH,ECX=0):EAX[*i*] = 0), sub-function *i* returns 0 in EAX, EBX, ECX, and EDX.

13.3 ENABLING THE XSAVE FEATURE SET AND XSAVE-SUPPORTED FEATURES

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

When CR4.OSXSAVE = 1 and CPL = 0, software can use the XSETBV instruction to write a value to XCR0. (Execution of the XSETBV instruction causes a general-protection fault — #GP — if CPL > 0.) The following items provide details regarding individual bits in XCR0:

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

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

  XCR0[2] is 0 coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[2] if and only if CPUID.(EAX=0DH,ECX=0):EAX[2] = 1. In addition, execution of the XSETBV instruction causes a general-protection fault (#GP) if bits 2:1 of its source operand (EAX[2:1]) has the value 10b; that is, software cannot enable the XSAVE feature set for AVX state but not for SSE state.
- XCR0[63:3] are reserved. Execution of the XSETBV instruction causes a general-protection fault (#GP) if any of bits 63:3 of its source operand (EDX and EAX[31:3]) is 0. Bits 63:3 of XCR0 are all 0 coming out of RESET.

If CPL> 3, execution of the MOV from CR4 instruction causes a general-protection fault (#GP). Other mechanisms allow software to discover the enabling of the XSAVE feature set regardless of CPL:

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

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

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

13.4 XSAVE AREA

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

Every XSAVE area has the following format:

- The **legacy region**. The legacy region of an XSAVE area comprises the 512 bytes starting at the area’s base address. It is used to manage the state components for x87 state and SSE state. The legacy region is described in more detail in Section 13.4.1.
- The **XSAVE header**. The XSAVE header of an XSAVE area comprises the 64 bytes starting at an offset of 512 bytes from the area’s base address. The first 8 bytes the XSAVE header is a state-component bitmap (see Section 13.1) that identifies the state components in the XSAVE area. The XSAVE header is described in more detail in Section 13.4.2.
- The **extended region**. The extended region of an XSAVE area starts at an offset of 576 bytes from the area’s base address. It is used to manage the state components other than those for x87 state and SSE state. The extended region is described in more detail in Section 13.4.3. The size of the extended region is determined by which state components the processor supports and which have been enabled in XCR0 (see Section 13.3).

13.4.1 Legacy Region of an XSAVE Area

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

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

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
Reserved	CS or FPU IP bits 63:32	FPU IP bits 31:0		FOP	Rsvd.	FTW	FSW	FCW	**0**

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

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

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

Section 13.6 through Section 13.8 provide details of how instructions in the XSAVE feature set use the legacy region of an XSAVE area.

13.4.2 XSAVE Header

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

The first 8 bytes the XSAVE header is a state-component bitmap (see Section 13.1) that is called XSTATE_BV and which identifies the state components in the XSAVE area. The remaining 56 bytes of the XSAVE header are reserved.

Section 13.6 through Section 13.8 provide details of how instructions in the XSAVE feature set use the XSAVE header of an XSAVE area.

13.4.3 Extended Region of an XSAVE Area

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

The XSAVE feature set uses the extended area for each state component *i*, where *i* > 1. (Currently, the extended region is used only for AVX state, which is state component 2.)

The processor locates each state component in the extended region at an offset from the base address of the XSAVE area. The processor enumerates the byte offset for state component *i* in CPUID.(EAX=0DH,ECX=*i*):EBX; it enumerates the number of bytes required for state component *i* in CPUID.(EAX=0DH,ECX=*i*):EAX.

13.5 XSAVE-MANAGED STATE

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

13.5.1 x87 State

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

- Bytes 1:0, 3:2, 7:6. These are used for the x87 FPU Control Word (FCW), the x87 FPU Status Word (FSW), and the x87 FPU Opcode (FOP), respectively.
- Byte 4 is used for an abridged version of the x87 FPU Tag Word (FTW). The following items describe its usage:
  - For each *j*, 0 ≤ *j* ≤ 7, XSAVE and XSAVEOPT save a 0 into bit *j* of byte 4 if x87 FPU data register ST*j* has a empty tag; otherwise, XSAVE and XSAVEOPT save a 1 into bit *j* of byte 4.
  - For each *j*, 0 ≤ *j* ≤ 7, XRSTOR 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[bit 13] = 0, bytes 13:12 are used for x87 FPU Instruction Pointer Selector (FPU CS). Otherwise, the processor deprecates the FPU CS value: XSAVE and XSAVEOPT save 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[bit 13] = 0, bytes 21:20 are used for x87 FPU Data Pointer Selector (FPU DS). Otherwise, the processor deprecates the FPU DS value: XSAVE and XSAVEOPT save 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 13.5.2).
- Bytes 159:32 are used for the registers ST0–ST7 (MM0–MM7). Each of the 8 register is allocated a 128-bit region, with the low 80 bits used for the register and the upper 48 bits unused.

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

13.5.2 SSE State

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

- Bytes 23:0 are used for x87 state (see Section 13.5.1).
- Bytes 27:24 are used for the MXCSR register. XRSTOR generates a general-protection fault (#GP) in response to an attempt to set any of the reserved bits of the MXCSR register.[2]
- Bytes 31:28 are used for the MXCSR_MASK value. XRSTOR ignores this field.
- Bytes 159:32 are used for x87 state.
- Bytes 287:160 are used for the registers XMM0–XMM7.
- Bytes 415:288 are used for the registers XMM8–XMM15. These fields are used only in 64-bit mode. Executions of XSAVE and XSAVEOPT outside 64-bit mode do not write to these bytes; executions of XRSTOR outside 64-bit mode do not read these bytes and do not update XMM8–XMM15.

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

13.5.3 AVX State

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

---

[1] The processor ensures that XCR0[0] is always 1.

[2] While MXCSR and MXCSR_MASK are part of SSE state, XSAVE and XSAVEOPT also save them (and XRSTOR restores MXCSR) when software has specified that AVX state should be saved (or restored). See Section 13.6 through Section 13.8.

As noted in Section 13.1, the XSAVE feature set manages AVX state as state component 2. Thus, these instructions organize all AVX state in the extended region of the XSAVE area (see Section 13.4.3).

As noted in Section 13.2, CPUID.(EAX=0DH,ECX=2):EBX enumerates the offset (in bytes, from the base of the XSAVE area) of the section of the extended region of the XSAVE area used for AVX state. CPUID returns this value as 576. CPUID.(EAX=0DH,ECX=2):EAX enumerates the size (in bytes) required for AVX state. CPUID returns this value as 256.

The XSAVE feature set partitions YMM0_H–YMM15_H in a manner similar to that used for the XMM registers (see Section 13.5.2. Bytes 127:0 of the AVX-state section are used YMM0_H–YMM7_H. Bytes 255:128 are used for YMM8_H–YMM15_H, but they are used only in 64-bit mode. (Executions of XSAVE and XSAVEOPT outside 64-bit mode do not write to bytes 255:128; executions of XRSTOR outside 64-bit mode do not read these bytes and do not update YMM8_H–YMM15_H.)

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

13.5.4 Processor Tracking of XSAVE-Managed State

The XSAVEOPT instruction uses two optimization to reduce the amount of data that it writes to memory. XSAVEOPT avoids writing data for any state component known to be in its initial configuration (the **init optimization**). In addition, if XSAVEOPT is using the same XSAVE area as that used by the most recent execution of XRSTOR, it avoids writing data for any state component whose configuration is known not to have been modified since that execution of XRSTOR (the **modified optimization**). The operation of XSAVEOPT is described in more detail in Section 13.8.

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

As detailed in Section 13.7, a processor that implements the modified optimization saves information about the most recent execution of XRSTOR in a quantity called **XRSTOR_INFO**. It contains the CPL, whether the logical processor was in VMX non-root operation, and the linear address of the XSAVE area. An execution of XSAVEOPT uses the modified optimization only if that execution corresponds to XRSTOR_INFO on these three parameters.

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

13.6 OPERATION OF XSAVE

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

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

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.

---

[1] The XSETBV instruction can set XCR0[2] to 1 only if it is also setting XCR0[1] to 1. XSETBV generates a general-protection exception (#GP) in response to attempts to set XCR0[2] while clearing XCR0[1].

- If the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

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

- If bit *i* of the requested-feature bitmap is 0, XSTATE_BV[*i*] is not changed. (This implies that XSAVE first reads the XSTATE_BV field.)
- If bit *i* of the requested-feature bitmap is 1, the value written to XSTATE_BV[*i*] depends on whether the state component corresponding to bit *i* is its initial configuration (see Section 13.5.4):
  - — If the state component is in its initial configuration, XSTATE_BV[*i*] may be written with either 0 or 1.
  - — If the state component is not in its initial configuration, XSTATE_BV[*i*] is written with 1.

  (In practice, the value stored into XSTATE_BV[*i*] depends on how the processor is tracking state component *i*; see Section 13.5.4. Limitations on the tracking ability may result in XSTATE_BV[*i*] being saved as 1 even though state component *i* is in its initial configuration.)

  The following items specify the initial configuration each state component (for the purposes of defining the values saved to XSTATE_BV):
  - — **x87 state.** x87 state is in its initial configuration if the following all hold: FCW is 037FH; FSW is 0000H; FTW is FFFFH; FPU CS and FPU DS are each 0000H; FPU IP and FPU DP are each 00000000_00000000H; each of ST0–ST7 is 0000_00000000_00000000H.
  - — **SSE state.** In 64-bit mode, SSE state is in its initial configuration if each of XMM0–XMM15 is 0. Outside 64-bit mode, SSE state is in its initial configuration if each of XMM0–XMM7 is 0. In neither case is the value of the MXCSR register considered.
  - — **AVX state.** In 64-bit mode, AVX state is in its initial configuration if each of YMM0_H–YMM15_H is 0. Outside 64-bit mode, AVX state is in its initial configuration if each of YMM0_H–YMM7_H is 0.

Execution of XSAVE saves into the XSAVE area those state components corresponding to bits that are set in the requested-feature bitmap. See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes.

The MXCSR register and MXCSR_MASK are part of SSE state (see Section 13.5.2) and is thus associated with bit 1 of the requested-feature bitmap. However, the XSAVE instruction also saves these values when bit 2 is set in the requested-feature bitmap (even if bit 1 is clear).

13.7 OPERATION OF XRSTOR

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

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

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- Any of the following conditions causes a general-protection exception (#GP):
  - — The address of the XSAVE area is not 64-byte aligned.[2]
  - — Bytes 23:8 of the XSAVE header (see Section 13.4.2) are not all 0.[3]

---

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

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

  - A bit is set in the XSTATE_BV field of the XSAVE header that is not set in XCR0.
  - The requested-feature bitmap sets either bit 1 (SSE) or bit 2 (AVX) and the value at bytes 27:24 of the legacy region is not a legal value for the MXCSR register (e.g., the value sets reserved bits).

If none of these conditions cause a fault, the processor updates each state component *i* if bit *i* is set in the requested-feature bitmap. XRSTOR updates state component *i* based on the value of bit *i* in the XSTATE_BV field of the XSAVE header (see Section 13.4.2):

- If XSTATE_BV[*i*] = 0, the state component is set to its initial configuration. The following items specify the initial configuration that XRSTOR establishes for each state component:
  - XRSTOR initializes x87 state by establishing the following: FCW is set to 037FH; FSW is set to 0000H; FTW is set to FFFFH; FPU CS and FPU DS are each set to 0000H; FPU IP and FPU DP are each set to 00000000_00000000H; each of ST0–ST7 is set to 0000_00000000_00000000H.
  - In 64-bit mode, XRSTOR initializes SSE state by setting each of XMM0–XMM15 to 0. Outside 64-bit mode, XRSTOR initializes SSE state by setting each of XMM0–XMM7 to 0. In either case, XRSTOR loads MXCSR from the XSAVE area whenever bit 1 is set in the requested-feature bitmap.
  - In 64-bit mode, XRSTOR initializes AVX state by setting each of YMM0_H–YMM15_H to 0. Outside 64-bit mode, XRSTOR initializes AVX state by setting each of YMM0_H–YMM7_H to 0. In either case, XRSTOR loads MXCSR from the XSAVE area whenever bit 2 is set in the requested-feature bitmap.
- If XSTATE_BV[*i*] = 1, the state component is loaded with data from the XSAVE area. See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes.

The MXCSR register is part of SSE state (see Section 13.5.2) and would thus normally be updated only if bit 1 is set in the requested-feature bitmap. However, the XRSTOR instruction loads the MXCSR register from memory whenever the request-feature bitmap sets either bit 1 (SSE) or bit 2 (AVX). The value of the XSTATE_BV field does not affect the loading of the MXCSR register; whenever XRSTOR modifies the value of MXCSR, it does so by loading it from memory.

Upon executing the XRSTOR instruction, the processor establishes modified tracking and records internally information about the XRSTOR execution for future interaction with the XSAVEOPT instruction (see Section 13.5.4 and Section 13.8):

- If bit *i* is 0 in the requested-feature bitmap, state component *i* is tracked as modified.
- If bit *i* is 1 in the requested-feature bitmap, state component *i* may be tracked as unmodified. (This tracking may change later if software uses state component *i*.)
- XRSTOR_INFO is set to the triple ‹*x*,*y*,*z*›, where *x* is the CPL; *y* is 1 if the logical processor is in VMX non-root operation and 0 otherwise; and *z* is the linear address of the XSAVE area.

13.8 OPERATION OF XSAVEOPT

The operation of XSAVEOPT is similar to that of XSAVE. XSAVEOPT includes optimizations by which it omits saving state components that are in their initial configuration or that have not been modified since the last corresponding execution of XRSTOR. See Section 13.5.4 for more details.

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

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

---

3. Bytes 63:24 of the XSAVE header are also reserved. Software should ensure that bytes 63:8 of the XSAVE header are all 0 in any XSAVE area.

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

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

- If bit *i* of the requested-feature bitmap is 0, XSTATE_BV[*i*] is not changed. (This implies that XSAVEOPT first reads the XSTATE_BV field.)
- If bit *i* of the requested-feature bitmap is 1, the value written to XSTATE_BV[*i*] depends on whether the state component corresponding to bit *i* is its initial configuration:
  - If the state component is in its initial configuration, XSTATE_BV[*i*] may be written with either 0 or 1.
  - If the state component is not in its initial configuration, XSTATE_BV[*i*] is written with 1.

  (In practice, the value stored into XSTATE_BV[*i*] depends on how the processor is tracking state component *i*; see Section 13.5.4. Limitations on the tracking ability may result in XSTATE_BV[*i*] being saved as 1 even though state component *i* is in its initial configuration.)

  See Section 13.6 for a specification of when each state component is considered to be in its initial configuration.

Execution of XSAVEOPT saves into the XSAVE area those state components corresponding to bits that are set in the requested-feature bitmap (and in XSTATE_BV; see below). See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes.

Execution of XSAVEOPT performs two optimizations that reduce the amount of data written to memory:

- **Init optimization.**
  If bit *i* is set in the requested-feature bitmap but XSAVEOPT is clearing XSTATE_BV[*i*] (see above), state component *i* is not saved to the XSAVE area.
- **Modified optimization.**
  As noted in Section 13.7, execution of XRSTOR established XRSTOR_INFO as a triple ‹*x*,*y*,*z*›. Execution of XSAVEOPT uses the modified optimization only if the following all hold:
  - CPL = *x*;
  - the logical processor is in VMX non-root operation if and only if *y* = 1; and
  - *z* is the linear address of the XSAVE area being used by XSAVEOPT.

  If XSAVEOPT uses the modified optimization and the processor is tracking state component *i* as unmodified (see Section 13.5.4), state component *i* is not saved to the XSAVE area.

  (In practice, the benefit of the modified optimization for state component *i* depends on how the processor is tracking state component *i*; see Section 13.5.4. Limitations on the tracking ability may result in state component *i* being saved even though is in the same configuration that was loaded by the previous execution of XRSTOR.)

  Depending on details of the operating system, an execution of XSAVEOPT by a user application might use the modified optimization when the most recent execution of XRSTOR was by a different application. Because of this, Intel recommends the application software not use the XSAVEOPT instruction.

The MXCSR register and MXCSR_MASK are part of SSE state (see Section 13.5.2) and is thus associated with bit 1 of the requested-feature bitmap. However, the XSAVEOPT instruction also saves these values when bit 2 is set in

---

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

the requested-feature bitmap (even if bit 1 is clear). The init and modified optimizations do not apply to the MXCSR register and MXCSR_MASK.

6. Updates to Appendix C, Volume 1

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

...

C.5 SSE3 INSTRUCTIONS

Table C-5 lists the SSE3 instructions that have at least one of the following characteristics:

- have floating-point operands
- generate floating-point results

For each instruction, the table summarizes the floating-point exceptions that the instruction can generate.

Table C-5 Exceptions Generated with SSE3 Instructions

Instruction	Description	#I	#D	#Z	#O	#U	#P
ADDSUBPD	Add /Sub packed DP FP numbers from XMM2/Mem to XMM1.	Y	Y		Y	Y	Y
ADDSUBPS	Add /Sub packed SP FP numbers from XMM2/Mem to XMM1.	Y	Y		Y	Y	Y
FISTTP	See Table C-2.	Y					Y
HADDPD	Add horizontally packed DP FP numbers XMM2/Mem to XMM1.	Y	Y		Y	Y	Y
HADDPS	Add horizontally packed SP FP numbers XMM2/Mem to XMM1	Y	Y		Y	Y	Y
HSUBPD	Sub horizontally packed DP FP numbers XMM2/Mem to XMM1	Y	Y		Y	Y	Y
HSUBPS	Sub horizontally packed SP FP numbers XMM2/Mem to XMM1	Y	Y		Y	Y	Y

Other SSE3 instructions do not generate floating-point exceptions.

...

7. Updates to Appendix E, Volume 1

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

...

Table E-15 #D - Denormal Operand

Instruction	Condition	Masked Response	Unmasked Response and Exception Code
ADDPS ADDPD ADDSUBPS ADDSUBPD HADDPS HADDPD SUBPS SUBPD HSUBPS HSUBPD MULPS MULPD DIVPS DIVPD SQRTPS SQRTPD MAXPS MAXPD MINPS MINPD ADDSS ADDSD SUBSS SUBSD MULSS MULSD DIVSS DIVSD SQRTSS SQRTSD MAXSS MAXSD MINSS MINSD CVTPS2PD CVTSS2SD CVTPD2PS CVTSD2SS	src1 = denormal[1] or src2 = denormal (and the DAZ bit in MXCSR is 0)	res = Result rounded to the destination precision and using the bounded exponent, but only if no unmasked post-computation exception occurs; #DE = 1.	src1, src2 unchanged; #DE = 1 Note that SQRT, CVTPS2PD, CVTSS2SD, CVTPD2PS, CVTSD2SS have only 1 src.
CMPPS CMPPD CMPSS CMPSD	src1 = denormal[1] or src2 = denormal (and the DAZ bit in MXCSR is 0)	Comparison result, stored in the destination register; #DE = 1	src1, src2 unchanged; #DE = 1
COMISS COMISD UCOMISS UCOMISD	src1 = denormal[1] or src2 = denormal (and the DAZ bit in MXCSR is 0)	Comparison result, stored in the EFLAGS register; #DE = 1	src1, src2 unchanged; #DE = 1

NOTE:

1. For denormal encodings, see Section 4.8.3.2, "Normalized and Denormalized Finite Numbers."

...

8. 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-M, Part 1.

...

BEXTR — Bit Field Extract

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.NDS[1].LZ.0F38.W0 F7 /r BEXTR r32a, r/m32, r32b	RMV	V/V	BMI1	Contiguous bitwise extract from r/m32 using r32b as control; store result in r32a.
VEX.NDS[1].LZ.0F38.W1 F7 /r BEXTR r64a, r/m64, r64b	RMV	V/N.E.	BMI1	Contiguous bitwise extract from r/m64 using r64b as control; store result in r64a

NOTES:

1. ModRM:r/m is used to encode the first source operand (second operand) and VEX.vvvv encodes the second source operand (third operand).

Instruction Operand Encoding

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

Description

Extracts contiguous bits from the first source operand (the second operand) using an index value and length value specified in the second source operand (the third operand). Bit 7:0 of the second source operand specifies the starting bit position of bit extraction. A START value exceeding the operand size will not extract any bits from the second source operand. Bit 15:8 of the second source operand specifies the maximum number of bits (LENGTH) beginning at the START position to extract. Only bit positions up to (OperandSize -1) of the first source operand are extracted. The extracted bits are written to the destination register, starting from the least significant bit. All higher order bits in the destination operand (starting at bit position LENGTH) are zeroed. The destination register is cleared if no bits are extracted.

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

Operation

```
START ← SRC2[7:0];
LEN ← SRC2[15:8];
TEMP ← ZERO_EXTEND_TO_512 (SRC1 );
DEST ← ZERO_EXTEND(TEMP[START+LEN -1: START]);
ZF ← (DEST = 0);
```

Flags Affected

ZF is updated based on the result. AF, SF, and PF are undefined. All other flags are cleared.

Intel C/C++ Compiler Intrinsic Equivalent

BEXTR:	unsigned __int32 _bextr_u32(unsigned __int32 src, unsigned __int32 start. unsigned __int32 len);
BEXTR:	unsigned __int64 _bextr_u64(unsigned __int64 src, unsigned __int32 start. unsigned __int32 len);

SIMD Floating-Point Exceptions

None

Other Exceptions

See Section 2.5.1, “Exception Conditions for VEX-Encoded GPR Instructions”, Table 2-29; additionally

#UD If VEX.W = 1.

...

CMPXCHG—Compare and Exchange

Opcode/ Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F B0/*r* CMPXCHG *r/m8, r8*	MR	Valid	Valid*	Compare AL with *r/m8*. If equal, ZF is set and *r8* is loaded into *r/m8*. Else, clear ZF and load *r/m8* into AL.
REX + 0F B0/*r* CMPXCHG *r/m8**,r8*	MR	Valid	N.E.	Compare AL with *r/m8*. If equal, ZF is set and *r8* is loaded into *r/m8*. Else, clear ZF and load *r/m8* into AL.
0F B1/*r* CMPXCHG *r/m16, r16*	MR	Valid	Valid*	Compare AX with *r/m16*. If equal, ZF is set and *r16* is loaded into *r/m16*. Else, clear ZF and load *r/m16* into AX.
0F B1/*r* CMPXCHG *r/m32, r32*	MR	Valid	Valid*	Compare EAX with *r/m32*. If equal, ZF is set and *r32* is loaded into *r/m32*. Else, clear ZF and load *r/m32* into EAX.
REX.W + 0F B1/*r* CMPXCHG *r/m64, r64*	MR	Valid	N.E.	Compare RAX with *r/m64*. If equal, ZF is set and *r64* is loaded into *r/m64*. Else, clear ZF and load *r/m64* into RAX.

NOTES:

* See the IA-32 Architecture Compatibility section below.

** 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 (r, w)	ModRM:reg (r)	NA	NA

Description

Compares the value in the AL, AX, EAX, or RAX register with the first operand (destination operand). If the two values are equal, the second operand (source operand) is loaded into the destination operand. Otherwise, the destination operand is loaded into the AL, AX, EAX or RAX register. RAX register is available only in 64-bit mode.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically. To simplify the interface to the processor's bus, the destination operand receives a write cycle without regard to the result of the comparison. The destination operand is written back if the comparison fails; otherwise, the source operand is written into the destination. (The processor never produces a locked read without also producing a locked write.)

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.

IA-32 Architecture Compatibility

This instruction is not supported on Intel processors earlier than the Intel486 processors.

Operation

```
(* Accumulator = AL, AX, EAX, or RAX depending on whether a byte, word, doubleword, or quadword comparison is being performed
*)
TEMP ← DEST
IF accumulator = TEMP
    THEN
        ZF ← 1;
        DEST ← SRC;
    ELSE
        ZF ← 0;
        accumulator ← TEMP;
        DEST ← TEMP;
FI;
```

Flags Affected

The ZF flag is set if the values in the destination operand and register AL, AX, or EAX are equal; otherwise it is cleared. The CF, PF, AF, SF, and OF flags are set according to the results of the comparison operation.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Real-Address Mode Exceptions

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

#UD	If the LOCK prefix is used but the destination is not a memory operand.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

...

CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes

Opcode/ Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F C7 /*1 m64* CMPXCHG8B *m64*	M	Valid	Valid*	Compare EDX:EAX with *m64*. If equal, set ZF and load ECX:EBX into *m64*. Else, clear ZF and load *m64* into EDX:EAX.
REX.W + 0F C7 /*1 m128* CMPXCHG16B *m128*	M	Valid	N.E.	Compare RDX:RAX with *m128*. If equal, set ZF and load RCX:RBX into *m128*. Else, clear ZF and load *m128* into RDX:RAX.

NOTES:

*See IA-32 Architecture Compatibility section below.

Instruction Operand Encoding

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

Description

Compares the 64-bit value in EDX:EAX (or 128-bit value in RDX:RAX if operand size is 128 bits) with the operand (destination operand). If the values are equal, the 64-bit value in ECX:EBX (or 128-bit value in RCX:RBX) is stored in the destination operand. Otherwise, the value in the destination operand is loaded into EDX:EAX (or RDX:RAX). The destination operand is an 8-byte memory location (or 16-byte memory location if operand size is 128 bits). For the EDX:EAX and ECX:EBX register pairs, EDX and ECX contain the high-order 32 bits and EAX and EBX contain the low-order 32 bits of a 64-bit value. For the RDX:RAX and RCX:RBX register pairs, RDX and RCX contain the high-order 64 bits and RAX and RBX contain the low-order 64bits of a 128-bit value.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically. To simplify the interface to the processor's bus, the destination operand receives a write cycle without regard to the result of the comparison. The destination operand is written back if the comparison fails; otherwise, the source operand is written into the destination. (The processor never produces a locked read without also producing a locked write.)

In 64-bit mode, default operation size is 64 bits. Use of the REX.W prefix promotes operation to 128 bits. Note that CMPXCHG16B requires that the destination (memory) operand be 16-byte aligned. See the summary chart at the beginning of this section for encoding data and limits. For information on the CPUID flag that indicates CMPXCHG16B, see page 3-168.

IA-32 Architecture Compatibility

This instruction encoding is not supported on Intel processors earlier than the Pentium processors.

Operation

```
IF (64-Bit Mode and OperandSize = 64)
    THEN
        TEMP128 ← DEST
        IF (RDX:RAX = TEMP128)
            THEN
                ZF ← 1;
                DEST ← RCX:RBX;
            ELSE
                ZF ← 0;
                RDX:RAX ← TEMP128;
                DEST ← TEMP128;
                FI;
        FI
    ELSE
        TEMP64 ← DEST;
        IF (EDX:EAX = TEMP64)
            THEN
                ZF ← 1;
                DEST ← ECX:EBX;
            ELSE
                ZF ← 0;
                EDX:EAX ← TEMP64;
                DEST ← TEMP64;
                FI;
        FI;
FI;
```

Flags Affected

The ZF flag is set if the destination operand and EDX:EAX are equal; otherwise it is cleared. The CF, PF, AF, SF, and OF flags are unaffected.

Protected Mode Exceptions

#UD	If the destination is not a memory operand.
#GP(0)	If the destination is located in a non-writable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

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

Real-Address Mode Exceptions

#UD	If the destination operand is not a memory location.
#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.

Virtual-8086 Mode Exceptions

#UD	If the destination operand is not a memory location.
#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.

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.
	If memory operand for CMPXCHG16B is not aligned on a 16-byte boundary.
	If CPUID.01H:ECX.CMPXCHG16B[bit 13] = 0.
#UD	If the destination operand is not a memory location.
#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.

...

Table 3-17 Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0H	EAX	Maximum Input Value for Basic CPUID Information (see Table 3-18)
	EBX	“Genu”
	ECX	“ntel”
	EDX	“inel”

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

Initial EAX Value	Information Provided about the Processor	
01H	EAX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-5)
	EBX	Bits 07-00: Brand Index Bits 15-08: CLFLUSH line size (Value * 8 = cache line size in bytes) Bits 23-16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31-24: Initial APIC ID
	ECX	Feature Information (see Figure 3-6 and Table 3-20)
	EDX	Feature Information (see Figure 3-7 and Table 3-21)
		NOTES: * The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. This field is only valid if CPUID.1.EDX.HTT[bit 28]= 1.
02H	EAX	Cache and TLB Information (see Table 3-22)
	EBX	Cache and TLB Information
	ECX	Cache and TLB Information
	EDX	Cache and TLB Information
03H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
	EDX	Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
		NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature. See AP-485, *Intel Processor Identification and the CPUID Instruction* (Order Number 241618) for more information on PSN.
	CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLE.BOOT_NT4[bit 22] = 0 (default).	
	Deterministic Cache Parameters Leaf	
04H		**NOTES:** Leaf 04H output depends on the initial value in ECX.* See also: “INPUT EAX = 4: Returns Deterministic Cache Parameters for each level on page 3-176.
	EAX	Bits 04-00: Cache Type Field 0 = Null - No more caches 1 = Data Cache 2 = Instruction Cache 3 = Unified Cache 4-31 = Reserved Bits 07-05: Cache Level (starts at 1) Bit 08: Self Initializing cache level (does not need SW initialization) Bit 09: Fully Associative cache

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

Initial EAX Value	Information Provided about the Processor	
		Bits 13-10: Reserved Bits 25-14: Maximum number of addressable IDs for logical processors sharing this cache**, *** Bits 31-26: Maximum number of addressable IDs for processor cores in the physical package**, ****, *****
	EBX	Bits 11-00: L = System Coherency Line Size** Bits 21-12: P = Physical Line partitions** Bits 31-22: W = Ways of associativity**
	ECX	Bits 31-00: S = Number of Sets**
	EDX	Bit 0: Write-Back Invalidate/Invalidate 0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache. Bit 1: Cache Inclusiveness 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 2: Complex Cache Indexing 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31-03: Reserved = 0 **NOTES:** * If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0. Invalid sub-leaves of EAX = 04H: ECX = n, n > 3. ** Add one to the return value to get the result. ***The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache **** The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. ***** The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.
	MONITOR/MWAIT Leaf	
05H	EAX	Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	EBX	Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	ECX	Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled Bits 31 - 02: Reserved

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

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 03 - 00: Number of C0* sub C-states supported using MWAIT Bits 07 - 04: Number of C1* sub C-states supported using MWAIT Bits 11 - 08: Number of C2* sub C-states supported using MWAIT Bits 15 - 12: Number of C3* sub C-states supported using MWAIT Bits 19 - 16: Number of C4* sub C-states supported using MWAIT Bits 23 - 20: Number of C5* sub C-states supported using MWAIT Bits 27 - 24: Number of C6* sub C-states supported using MWAIT Bits 31 - 28: Number of C7* sub C-states supported using MWAIT **NOTE:** * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.
	Thermal and Power Management Leaf	
06H	EAX	Bit 00: Digital temperature sensor is supported if set Bit 01: Intel Turbo Boost Technology Available (see description of IA32_MISC_ENABLE[38]). Bit 02: ARAT. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved Bit 04: PLN. Power limit notification controls are supported if set. Bit 05: ECMD. Clock modulation duty cycle extension is supported if set. Bit 06: PTM. Package thermal management is supported if set. Bits 31 - 07: Reserved
	EBX	Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor Bits 31 - 04: Reserved
	ECX	Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of expected processor performance at frequency specified in CPUID Brand String Bits 02 - 01: Reserved = 0 Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H) Bits 31 - 04: Reserved = 0
	EDX	Reserved = 0
	Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)	
07H		Sub-leaf 0 (Input ECX = 0). *
	EAX	Bits 31-00: Reports the maximum input value for supported leaf 7 sub-leaves.

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

Initial EAX Value	Information Provided about the Processor	
	EBX	Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 02: Reserved Bit 03: BMI1 Bit 04: HLE Bit 05: AVX2 Bit 06: Reserved Bit 07: SMEP. Supports Supervisor-Mode Execution Prevention if 1. Bit 08: BMI2 Bit 09: Supports Enhanced REP MOVSB/STOSB if 1. Bit 10: INVPCID. If 1, supports INVPCID instruction for system software that manages process-context identifiers. Bit 11: RTM Bit 12: Supports Quality of Service Monitoring (QM) capability if 1. Bit 13: Deprecates FPU CS and FPU DS values if 1. Bits 31:14: Reserved
	ECX	Reserved
	EDX	Reserved **NOTE:** * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Invalid sub-leaves of EAX = 07H: ECX = n, n > 0.
	Direct Cache Access Information Leaf	
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H)
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
	Architectural Performance Monitoring Leaf	
0AH	EAX	Bits 07 - 00: Version ID of architectural performance monitoring Bits 15- 08: Number of general-purpose performance monitoring counter per logical processor Bits 23 - 16: Bit width of general-purpose, performance monitoring counter Bits 31 - 24: Length of EBX bit vector to enumerate architectural performance monitoring events
	EBX	Bit 00: Core cycle event not available if 1 Bit 01: Instruction retired event not available if 1 Bit 02: Reference cycles event not available if 1 Bit 03: Last-level cache reference event not available if 1 Bit 04: Last-level cache misses event not available if 1 Bit 05: Branch instruction retired event not available if 1 Bit 06: Branch mispredict retired event not available if 1 Bits 31- 07: Reserved = 0
	ECX	Reserved = 0
	EDX	Bits 04 - 00: Number of fixed-function performance counters (if Version ID > 1) Bits 12- 05: Bit width of fixed-function performance counters (if Version ID > 1) Reserved = 0
	Extended Topology Enumeration Leaf	

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

Initial EAX Value	Information Provided about the Processor	
0BH		**NOTES:** Most of Leaf 0BH output depends on the initial value in ECX. The EDX output of leaf 0BH is always valid and does not vary with input value in ECX. Output value in ECX[7:0] always equals input value in ECX[7:0]. For sub-leaves that return an invalid level-type of 0 in ECX[15:8]; EAX and EBX will return 0. If an input value n in ECX returns the invalid level-type of 0 in ECX[15:8], other input values with ECX > n also return 0 in ECX[15:8].
	EAX	Bits 04-00: Number of bits to shift right on x2APIC ID to get a unique topology ID of the next level type*. All logical processors with the same next level ID share current level. Bits 31-05: Reserved.
	EBX	Bits 15 - 00: Number of logical processors at this level type. The number reflects configuration as shipped by Intel**. Bits 31- 16: Reserved.
	ECX	Bits 07 - 00: Level number. Same value in ECX input Bits 15 - 08: Level type***. Bits 31 - 16:: Reserved.
	EDX	Bits 31- 00: x2APIC ID the current logical processor.
		NOTES: * Software should use this field (EAX[4:0]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0 : invalid 1 : SMT 2 : Core 3-255 : Reserved
	Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)	
0DH		**NOTES:** Leaf 0DH main leaf (ECX = 0).
	EAX	Bits 31-00: Reports the valid bit fields of the lower 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved. Bit 00: legacy x87 Bit 01: 128-bit SSE Bit 02: 256-bit AVX Bits 31- 03: Reserved
	EBX	Bits 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.

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

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor, i.e all the valid bit fields in XCR0.
	EDX	Bit 31-00: Reports the valid bit fields of the upper 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved.
	Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)	
0DH	EAX	Bits 31-01: Reserved Bit 00: XSAVEOPT is available;
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
	Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)	
0DH		**NOTES:** Leaf 0DH output depends on the initial value in ECX. Each valid sub-leaf index maps to a valid bit in the XCR0 register starting at bit position 2 * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Invalid sub-leaves of EAX = 0DH: ECX = n, n > 2.
	EAX	Bits 31-0: The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index, *n*. This field reports 0 if the sub-leaf index, *n*, is invalid*.
	EBX	Bits 31-0: The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, *n*, is invalid*.
	ECX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
	EDX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
	Quality of Service Resource Type Enumeration Sub-leaf (EAX = 0FH, ECX = 0)	
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX
	EAX	Reserved.
	EBX	Bits 31-0: Maximum range (zero-based) of RMID within this physical processor of all types.
	ECX	Reserved.
	EDX	Bit 00: Reserved. Bit 01: Supports L3 Cache QoS if 1. Bits 31:02: Reserved
	L3 Cache QoS Capability Enumeration Sub-leaf (EAX = 0FH, ECX = 1)	
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX.
	EAX	Reserved.
	EBX	Bits 31-0: Conversion factor from reported IA32_QM_CTR value to occupancy metric (bytes).
	ECX	Maximum range (zero-based) of RMID of this resource type.

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

Initial EAX Value	Information Provided about the Processor	
	EDX	Bit 00: Supports L3 occupancy monitoring if 1. Bits 31:01: Reserved
	Unimplemented CPUID Leaf Functions	
40000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
	Extended Function CPUID Information	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (see Table 3-18).
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved
	ECX	Bit 00: LAHF/SAHF available in 64-bit mode Bits 04-01 Reserved Bit 05: LZCNT Bits 07-06 Reserved Bit 08: PREFETCHW Bits 31-09 Reserved
	EDX	Bits 10-00: Reserved Bit 11: SYSCALL/SYSRET available in 64-bit mode Bits 19-12: Reserved = 0 Bit 20: Execute Disable Bit available Bits 25-21: Reserved = 0 Bit 26: 1-GByte pages are available if 1 Bit 27: RDTSCP and IA32_TSC_AUX are available if 1 Bits 28: Reserved = 0 Bit 29: Intel® 64 Architecture available if 1 Bits 31-30: Reserved = 0
80000002H	EAX	Processor Brand String
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
80000003H	EAX	Processor Brand String Continued
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
80000004H	EAX	Processor Brand String Continued
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued

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

Initial EAX Value	Information Provided about the Processor	
80000005H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0
80000006H	EAX EBX	Reserved = 0 Reserved = 0
	ECX	Bits 07-00: Cache Line size in bytes Bits 11-08: Reserved Bits 15-12: L2 Associativity field * Bits 31-16: Cache size in 1K units
	EDX	Reserved = 0
		NOTES: * L2 associativity field encodings: 00H - Disabled 01H - Direct mapped 02H - 2-way 04H - 4-way 06H - 8-way 08H - 16-way 0FH - Fully associative
80000007H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Bits 07-00: Reserved = 0 Bit 08: Invariant TSC available if 1 Bits 31-09: Reserved = 0
80000008H	EAX	Linear/Physical Address size Bits 07-00: #Physical Address Bits* Bits 15-8: #Linear Address Bits Bits 31-16: Reserved = 0
	EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0
		NOTES: * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

...

Table 3-22 Encoding of CPUID Leaf 2 Descriptors

Value	Type	Description
00H	General	Null descriptor, this byte contains no information
01H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 32 entries
02H	TLB	Instruction TLB: 4 MByte pages, fully associative, 2 entries

Table 3-22 Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
03H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 64 entries
04H	TLB	Data TLB: 4 MByte pages, 4-way set associative, 8 entries
05H	TLB	Data TLB1: 4 MByte pages, 4-way set associative, 32 entries
06H	Cache	1st-level instruction cache: 8 KBytes, 4-way set associative, 32 byte line size
08H	Cache	1st-level instruction cache: 16 KBytes, 4-way set associative, 32 byte line size
09H	Cache	1st-level instruction cache: 32KBytes, 4-way set associative, 64 byte line size
0AH	Cache	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size
0BH	TLB	Instruction TLB: 4 MByte pages, 4-way set associative, 4 entries
0CH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 32 byte line size
0DH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 64 byte line size
0EH	Cache	1st-level data cache: 24 KBytes, 6-way set associative, 64 byte line size
21H	Cache	2nd-level cache: 256 KBytes, 8-way set associative, 64 byte line size
22H	Cache	3rd-level cache: 512 KBytes, 4-way set associative, 64 byte line size, 2 lines per sector
23H	Cache	3rd-level cache: 1 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
24H	Cache	2nd-level cache: 1 MBytes, 16-way set associative, 64 byte line size
25H	Cache	3rd-level cache: 2 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
29H	Cache	3rd-level cache: 4 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
2CH	Cache	1st-level data cache: 32 KBytes, 8-way set associative, 64 byte line size
30H	Cache	1st-level instruction cache: 32 KBytes, 8-way set associative, 64 byte line size
40H	Cache	No 2nd-level cache or, if processor contains a valid 2nd-level cache, no 3rd-level cache
41H	Cache	2nd-level cache: 128 KBytes, 4-way set associative, 32 byte line size
42H	Cache	2nd-level cache: 256 KBytes, 4-way set associative, 32 byte line size
43H	Cache	2nd-level cache: 512 KBytes, 4-way set associative, 32 byte line size
44H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 32 byte line size
45H	Cache	2nd-level cache: 2 MByte, 4-way set associative, 32 byte line size
46H	Cache	3rd-level cache: 4 MByte, 4-way set associative, 64 byte line size
47H	Cache	3rd-level cache: 8 MByte, 8-way set associative, 64 byte line size
48H	Cache	2nd-level cache: 3MByte, 12-way set associative, 64 byte line size
49H	Cache	3rd-level cache: 4MB, 16-way set associative, 64-byte line size (Intel Xeon processor MP, Family 0FH, Model 06H); 2nd-level cache: 4 MByte, 16-way set associative, 64 byte line size
4AH	Cache	3rd-level cache: 6MByte, 12-way set associative, 64 byte line size
4BH	Cache	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size
4CH	Cache	3rd-level cache: 12MByte, 12-way set associative, 64 byte line size
4DH	Cache	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size
4EH	Cache	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size
4FH	TLB	Instruction TLB: 4 KByte pages, 32 entries
50H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries

Table 3-22 Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
51H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 128 entries
52H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 256 entries
55H	TLB	Instruction TLB: 2-MByte or 4-MByte pages, fully associative, 7 entries
56H	TLB	Data TLB0: 4 MByte pages, 4-way set associative, 16 entries
57H	TLB	Data TLB0: 4 KByte pages, 4-way associative, 16 entries
59H	TLB	Data TLB0: 4 KByte pages, fully associative, 16 entries
5AH	TLB	Data TLB0: 2-MByte or 4 MByte pages, 4-way set associative, 32 entries
5BH	TLB	Data TLB: 4 KByte and 4 MByte pages, 64 entries
5CH	TLB	Data TLB: 4 KByte and 4 MByte pages,128 entries
5DH	TLB	Data TLB: 4 KByte and 4 MByte pages,256 entries
60H	Cache	1st-level data cache: 16 KByte, 8-way set associative, 64 byte line size
61H	TLB	Instruction TLB: 4 KByte pages, fully associative, 48 entries
63H	TLB	Data TLB: 1 GByte pages, 4-way set associative, 4 entries
66H	Cache	1st-level data cache: 8 KByte, 4-way set associative, 64 byte line size
67H	Cache	1st-level data cache: 16 KByte, 4-way set associative, 64 byte line size
68H	Cache	1st-level data cache: 32 KByte, 4-way set associative, 64 byte line size
70H	Cache	Trace cache: 12 K-μop, 8-way set associative
71H	Cache	Trace cache: 16 K-μop, 8-way set associative
72H	Cache	Trace cache: 32 K-μop, 8-way set associative
76H	TLB	Instruction TLB: 2M/4M pages, fully associative, 8 entries
78H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 64byte line size
79H	Cache	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7AH	Cache	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7BH	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7CH	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector
7DH	Cache	2nd-level cache: 2 MByte, 8-way set associative, 64byte line size
7FH	Cache	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size
80H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64-byte line size
82H	Cache	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size
83H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size
B0H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries
B1H	TLB	Instruction TLB: 2M pages, 4-way, 8 entries or 4M pages, 4-way, 4 entries
B2H	TLB	Instruction TLB: 4KByte pages, 4-way set associative, 64 entries
B3H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 128 entries

Table 3-22 Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
B4H	TLB	Data TLB1: 4 KByte pages, 4-way associative, 256 entries
B5H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 64 entries
B6H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 128 entries
BAH	TLB	Data TLB1: 4 KByte pages, 4-way associative, 64 entries
C0H	TLB	Data TLB: 4 KByte and 4 MByte pages, 4-way associative, 8 entries
C1H	STLB	Shared 2nd-Level TLB: 4 KByte/2MByte pages, 8-way associative, 1024 entries
C2H	DTLB	DTLB: 2 MByte/$MByte pages, 4-way associative, 16 entries
CAH	STLB	Shared 2nd-Level TLB: 4 KByte pages, 4-way associative, 512 entries
D0H	Cache	3rd-level cache: 512 KByte, 4-way set associative, 64 byte line size
D1H	Cache	3rd-level cache: 1 MByte, 4-way set associative, 64 byte line size
D2H	Cache	3rd-level cache: 2 MByte, 4-way set associative, 64 byte line size
D6H	Cache	3rd-level cache: 1 MByte, 8-way set associative, 64 byte line size
D7H	Cache	3rd-level cache: 2 MByte, 8-way set associative, 64 byte line size
D8H	Cache	3rd-level cache: 4 MByte, 8-way set associative, 64 byte line size
DCH	Cache	3rd-level cache: 1.5 MByte, 12-way set associative, 64 byte line size
DDH	Cache	3rd-level cache: 3 MByte, 12-way set associative, 64 byte line size
DEH	Cache	3rd-level cache: 6 MByte, 12-way set associative, 64 byte line size
E2H	Cache	3rd-level cache: 2 MByte, 16-way set associative, 64 byte line size
E3H	Cache	3rd-level cache: 4 MByte, 16-way set associative, 64 byte line size
E4H	Cache	3rd-level cache: 8 MByte, 16-way set associative, 64 byte line size
EAH	Cache	3rd-level cache: 12MByte, 24-way set associative, 64 byte line size
EBH	Cache	3rd-level cache: 18MByte, 24-way set associative, 64 byte line size
ECH	Cache	3rd-level cache: 24MByte, 24-way set associative, 64 byte line size
F0H	Prefetch	64-Byte prefetching
F1H	Prefetch	128-Byte prefetching
FFH	General	CPUID leaf 2 does not report cache descriptor information, use CPUID leaf 4 to query cache parameters

...

IMUL—Signed Multiply

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F6 /5	IMUL *r/m8**	M	Valid	Valid	AX← AL ∗ *r/m* byte.
F7 /5	IMUL *r/m16*	M	Valid	Valid	DX:AX ← AX ∗ *r/m* word.
F7 /5	IMUL *r/m32*	M	Valid	Valid	EDX:EAX ← EAX ∗ *r/m*32.
REX.W + F7 /5	IMUL *r/m64*	M	Valid	N.E.	RDX:RAX ← RAX ∗ *r/m*64.
0F AF /*r*	IMUL *r16, r/m16*	RM	Valid	Valid	word register ← word register ∗ *r/m*16.
0F AF /*r*	IMUL *r32, r/m32*	RM	Valid	Valid	doubleword register ← doubleword register ∗ *r/m32.*
REX.W + 0F AF /*r*	IMUL *r64, r/m64*	RM	Valid	N.E.	Quadword register ← Quadword register ∗ *r/m64*.
6B /*r ib*	IMUL *r16, r/m16, imm8*	RMI	Valid	Valid	word register ← *r/m16* ∗ sign-extended immediate byte.
6B /*r ib*	IMUL *r32, r/m32, imm8*	RMI	Valid	Valid	doubleword register ← *r/m32* ∗ sign-extended immediate byte.
REX.W + 6B /*r ib*	IMUL *r64, r/m64, imm8*	RMI	Valid	N.E.	Quadword register ← *r/m64* ∗ sign-extended immediate byte.
69 /*r iw*	IMUL *r16, r/m16, imm16*	RMI	Valid	Valid	word register ← *r/m16* ∗ immediate word.
69 /*r id*	IMUL *r32, r/m32, imm32*	RMI	Valid	Valid	doubleword register ← *r/m32* ∗ immediate doubleword.
REX.W + 69 /*r id*	IMUL *r64, r/m64, imm32*	RMI	Valid	N.E.	Quadword register ← *r/m64* ∗ immediate doubleword.

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, w)	NA	NA	NA
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
RMI	ModRM:reg (r, w)	ModRM:r/m (r)	imm8/16/32	NA

Description

Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands.

- **One-operand form** — This form is identical to that used by the MUL instruction. Here, the source operand (in a general-purpose register or memory location) is multiplied by the value in the AL, AX, EAX, or RAX register (depending on the operand size) and the product is stored in the AX, DX:AX, EDX:EAX, or RDX:RAX registers, respectively.
- **Two-operand form** — With this form the destination operand (the first operand) is multiplied by the source operand (second operand). The destination operand is a general-purpose register and the source operand is an immediate value, a general-purpose register, or a memory location. The product is then stored in the destination operand location.

- **Three-operand form** — This form requires a destination operand (the first operand) and two source operands (the second and the third operands). Here, the first source operand (which can be a general-purpose register or a memory location) is multiplied by the second source operand (an immediate value). The product is then stored in the destination operand (a general-purpose register).

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

The CF and OF flags are set when significant bit (including the sign bit) are carried into the upper half of the result. The CF and OF flags are cleared when the result (including the sign bit) fits exactly in the lower half of the result.

The three forms of the IMUL instruction are similar in that the length of the product is calculated to twice the length of the operands. With the one-operand form, the product is stored exactly in the destination. With the two- and three- operand forms, however, the result is truncated to the length of the destination before it is stored in the destination register. Because of this truncation, the CF or OF flag should be tested to ensure that no significant bits are lost.

The two- and three-operand forms may also be used with unsigned operands because the lower half of the product is the same regardless if the operands are signed or unsigned. The CF and OF flags, however, cannot be used to determine if the upper half of the result is non-zero.

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. Use of REX.W modifies the three forms of the instruction as follows.

- One-operand form —The source operand (in a 64-bit general-purpose register or memory location) is multiplied by the value in the RAX register and the product is stored in the RDX:RAX registers.
- **Two-operand form** — The source operand is promoted to 64 bits if it is a register or a memory location. The destination operand is promoted to 64 bits.
- **Three-operand form** — The first source operand (either a register or a memory location) and destination operand are promoted to 64 bits. If the source operand is an immediate, it is sign extended to 64 bits.

Operation

```
IF (NumberOfOperands = 1)
    THEN IF (OperandSize = 8)
        THEN
            AX ← AL ∗ SRC (* Signed multiplication *)
            IF AL = AX
                THEN CF ← 0; OF ← 0;
                ELSE CF ← 1; OF ← 1; FI;
        ELSE IF OperandSize = 16
            THEN
                DX:AX ← AX ∗ SRC (* Signed multiplication *)
                IF sign_extend_to_32 (AX) = DX:AX
                    THEN CF ← 0; OF ← 0;
                    ELSE CF ← 1; OF ← 1; FI;
            ELSE IF OperandSize = 32
                THEN
                    EDX:EAX ← EAX ∗ SRC (* Signed multiplication *)
                    IF EAX = EDX:EAX
                        THEN CF ← 0; OF ← 0;
                        ELSE CF ← 1; OF ← 1; FI;
                ELSE (* OperandSize = 64 *)
                    RDX:RAX ← RAX ∗ SRC (* Signed multiplication *)
```

```
                    IF RAX = RDX:RAX
                        THEN CF ← 0; OF ← 0;
                        ELSE CF ← 1; OF ← 1; FI;
                FI;
        FI;
    ELSE IF (NumberOfOperands = 2)
        THEN
            temp ← DEST ∗ SRC (* Signed multiplication; temp is double DEST size *)
            DEST ← DEST ∗ SRC (* Signed multiplication *)
            IF temp ≠ DEST
                THEN CF ← 1; OF ← 1;
                ELSE CF ← 0; OF ← 0; FI;
        ELSE (* NumberOfOperands = 3 *)
            DEST ← SRC1 ∗ SRC2 (* Signed multiplication *)
            temp ← SRC1 ∗ SRC2 (* Signed multiplication; temp is double SRC1 size *)
            IF temp ≠ DEST
                THEN CF ← 1; OF ← 1;
                ELSE CF ← 0; OF ← 0; FI;
    FI;
FI;
```

Flags Affected

For the one operand form of the instruction, the CF and OF flags are set when significant bits are carried into the upper half of the result and cleared when the result fits exactly in the lower half of the result. For the two- and three-operand forms of the instruction, the CF and OF flags are set when the result must be truncated to fit in the destination operand size and cleared when the result fits exactly in the destination operand size. The SF, ZF, AF, and PF flags are undefined.

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 NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

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

...

LOOP/LOOP*cc*—Loop According to ECX Counter

Opcode	**Instruction**	**Op/ En**	**64-Bit Mode**	**Compat/ Leg Mode**	**Description**
E2 *cb*	LOOP *rel8*	D	Valid	Valid	Decrement count; jump short if count ≠ 0.
E1 *cb*	LOOPE *rel8*	D	Valid	Valid	Decrement count; jump short if count ≠ 0 and ZF = 1.
E0 *cb*	LOOPNE *rel8*	D	Valid	Valid	Decrement count; jump short if count ≠ 0 and ZF = 0.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
D	Offset	NA	NA	NA

Description

Performs a loop operation using the RCX, ECX or CX register as a counter (depending on whether address size is 64 bits, 32 bits, or 16 bits). Note that the LOOP instruction ignores REX.W; but 64-bit address size can be overridden using a 67H prefix.

Each time the LOOP instruction is executed, the count register is decremented, then checked for 0. If the count is 0, the loop is terminated and program execution continues with the instruction following the LOOP instruction. If the count is not zero, a near jump is performed to the destination (target) operand, which is presumably the instruction at the beginning of the loop.

The target instruction is specified with a relative offset (a signed offset relative to the current value of the instruction pointer in the IP/EIP/RIP register). This offset is generally specified as a label in assembly code, but at the machine code level, it is encoded as a signed, 8-bit immediate value, which is added to the instruction pointer. Offsets of –128 to +127 are allowed with this instruction.

Some forms of the loop instruction (LOOP*cc*) also accept the ZF flag as a condition for terminating the loop before the count reaches zero. With these forms of the instruction, a condition code (*cc*) is associated with each instruction to indicate the condition being tested for. Here, the LOOP*cc* instruction itself does not affect the state of the ZF flag; the ZF flag is changed by other instructions in the loop.

Operation

```
IF (AddressSize = 32)
    THEN Count is ECX;
ELSE IF (AddressSize = 64)
    Count is RCX;
ELSE Count is CX;
FI;

Count ← Count – 1;

IF Instruction is not LOOP
    THEN
        IF (Instruction ← LOOPE) or (Instruction ← LOOPZ)
            THEN IF (ZF = 1) and (Count ≠ 0)
                    THEN BranchCond ← 1;
                    ELSE BranchCond ← 0;
                FI;
            ELSE (Instruction = LOOPNE) or (Instruction = LOOPNZ)
                IF (ZF = 0 ) and (Count ≠ 0)
                    THEN BranchCond ← 1;
                    ELSE BranchCond ← 0;
                FI;
        FI;
    ELSE (* Instruction = LOOP *)
        IF (Count ≠ 0)
            THEN BranchCond ← 1;
            ELSE BranchCond ← 0;
        FI;
FI;

IF BranchCond = 1
    THEN
        IF OperandSize = 32
            THEN EIP ← EIP + SignExtend(DEST);
            ELSE IF OperandSize = 64
                THEN RIP ← RIP + SignExtend(DEST);
                FI;
            ELSE IF OperandSize = 16
                THEN EIP ← EIP AND 0000FFFFH;
                FI;
        FI;
        IF OperandSize = (32 or 64)
            THEN IF (R/E)IP < CS.Base or (R/E)IP > CS.Limit
                #GP; FI;
                FI;
        FI;
    ELSE
        Terminate loop and continue program execution at (R/E)IP;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the offset being jumped to is beyond the limits of the CS segment.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If the offset being jumped to is beyond the limits of the CS segment or is outside of the effective address space from 0 to FFFFH. This condition can occur if a 32-bit address size override prefix is used.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the offset being jumped to is in a non-canonical form.
#UD	If the LOCK prefix is used.

...

9. 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, N-Z, Part 2.

...

PBLENDVB — Variable Blend Packed Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 38 10 /r PBLENDVB *xmm1, xmm2/m128, <XMM0>*	RM	V/V	SSE4_1	Select byte values from *xmm1* and *xmm2/ m128* from mask specified in the high bit of each byte in *XMM0* and store the values into *xmm1.*
VEX.NDS.128.66.0F3A.W0 4C /r /is4 VPBLENDVB *xmm1, xmm2, xmm3/m128, xmm4*	RVMR	V/V	AVX	Select byte values from *xmm2* and *xmm3/ m128* using mask bits in the specified mask register, *xmm4*, and store the values into *xmm1*.
VEX.NDS.256.66.0F3A.W0 4C /r /is4 VPBLENDVB *ymm1, ymm2, ymm3/m256, ymm4*	RVMR	V/V	AVX2	Select byte values from *ymm2* and *ymm3/ m256* from mask specified in the high bit of each byte in *ymm4* and store the values into *ymm1*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	<XMM0>	NA
RVMR	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	ModRM:reg (r)

Description

Conditionally copies byte elements from the source operand (second operand) to the destination operand (first operand) depending on mask bits defined in the implicit third register argument, XMM0. The mask bits are the most significant bit in each byte element of the XMM0 register.

If a mask bit is “1", then the corresponding byte element in the source operand is copied to the destination, else the byte element in the destination operand is left unchanged.

The register assignment of the implicit third operand is defined to be the architectural register XMM0.

128-bit Legacy SSE version: The first source operand and the destination operand is the same. Bits (VLMAX-1:128) of the corresponding YMM destination register remain unchanged. The mask register operand is implicitly defined to be the architectural register XMM0. An attempt to execute PBLENDVB with a VEX prefix will cause #UD.

VEX.128 encoded version: The first source operand and the destination operand are XMM registers. The second source operand is an XMM register or 128-bit memory location. The mask operand is the third source register, and encoded in bits[7:4] of the immediate byte(imm8). The bits[3:0] of imm8 are ignored. In 32-bit mode, imm8[7] is ignored. The upper bits (VLMAX-1:128) of the corresponding YMM register (destination register) are zeroed. VEX.L must be 0, otherwise the instruction will #UD. VEX.W must be 0, otherwise, the instruction will #UD.

VEX.256 encoded version: The first source operand and the destination operand are YMM registers. The second source operand is an YMM register or 256-bit memory location. The third source register is an YMM register and encoded in bits[7:4] of the immediate byte(imm8). The bits[3:0] of imm8 are ignored. In 32-bit mode, imm8[7] is ignored.

VPBLENDVB permits the mask to be any XMM or YMM register. In contrast, PBLENDVB treats XMM0 implicitly as the mask and do not support non-destructive destination operation. An attempt to execute PBLENDVB encoded with a VEX prefix will cause a #UD exception.

Operation

PBLENDVB (128-bit Legacy SSE version)

```
MASK ← XMM0
IF (MASK[7] = 1) THEN DEST[7:0] ← SRC[7:0];
ELSE DEST[7:0] ← DEST[7:0];
IF (MASK[15] = 1) THEN DEST[15:8] ← SRC[15:8];
ELSE DEST[15:8] ← DEST[15:8];
IF (MASK[23] = 1) THEN DEST[23:16] ← SRC[23:16]
ELSE DEST[23:16] ← DEST[23:16];
IF (MASK[31] = 1) THEN DEST[31:24] ← SRC[31:24]
ELSE DEST[31:24] ← DEST[31:24];
IF (MASK[39] = 1) THEN DEST[39:32] ← SRC[39:32]
ELSE DEST[39:32] ← DEST[39:32];
IF (MASK[47] = 1) THEN DEST[47:40] ← SRC[47:40]
ELSE DEST[47:40] ← DEST[47:40];
IF (MASK[55] = 1) THEN DEST[55:48] ← SRC[55:48]
ELSE DEST[55:48] ← DEST[55:48];
IF (MASK[63] = 1) THEN DEST[63:56] ← SRC[63:56]
ELSE DEST[63:56] ← DEST[63:56];
IF (MASK[71] = 1) THEN DEST[71:64] ← SRC[71:64]
ELSE DEST[71:64] ← DEST[71:64];
IF (MASK[79] = 1) THEN DEST[79:72] ← SRC[79:72]
ELSE DEST[79:72] ← DEST[79:72];
IF (MASK[87] = 1) THEN DEST[87:80] ← SRC[87:80]
ELSE DEST[87:80] ← DEST[87:80];
IF (MASK[95] = 1) THEN DEST[95:88] ← SRC[95:88]
ELSE DEST[95:88] ←   DEST[95:88];
IF (MASK[103] = 1) THEN DEST[103:96] ← SRC[103:96]
ELSE DEST[103:96] ←   DEST[103:96];
IF (MASK[111] = 1) THEN DEST[111:104] ← SRC[111:104]
ELSE DEST[111:104] ← DEST[111:104];
IF (MASK[119] = 1) THEN DEST[119:112] ← SRC[119:112]
ELSE DEST[119:112] ← DEST[119:112];
IF (MASK[127] = 1) THEN DEST[127:120] ← SRC[127:120]
ELSE DEST[127:120] ← DEST[127:120])
DEST[VLMAX-1:128] (Unmodified)
```

VPBLENDVB (VEX.128 encoded version)

```
MASK ← SRC3
IF (MASK[7] = 1) THEN DEST[7:0] ← SRC2[7:0];
ELSE DEST[7:0] ← SRC1[7:0];
IF (MASK[15] = 1) THEN DEST[15:8] ← SRC2[15:8];
ELSE DEST[15:8] ← SRC1[15:8];
IF (MASK[23] = 1) THEN DEST[23:16] ← SRC2[23:16]
```

```
ELSE DEST[23:16] ← SRC1[23:16];
IF (MASK[31] = 1) THEN DEST[31:24] ← SRC2[31:24]
ELSE DEST[31:24] ← SRC1[31:24];
IF (MASK[39] = 1) THEN DEST[39:32] ← SRC2[39:32]
ELSE DEST[39:32] ← SRC1[39:32];
IF (MASK[47] = 1) THEN DEST[47:40] ← SRC2[47:40]
ELSE DEST[47:40] ← SRC1[47:40];
IF (MASK[55] = 1) THEN DEST[55:48] ← SRC2[55:48]
ELSE DEST[55:48] ← SRC1[55:48];
IF (MASK[63] = 1) THEN DEST[63:56] ← SRC2[63:56]
ELSE DEST[63:56] ← SRC1[63:56];
IF (MASK[71] = 1) THEN DEST[71:64] ← SRC2[71:64]
ELSE DEST[71:64] ← SRC1[71:64];
IF (MASK[79] = 1) THEN DEST[79:72] ← SRC2[79:72]
ELSE DEST[79:72] ← SRC1[79:72];
IF (MASK[87] = 1) THEN DEST[87:80] ← SRC2[87:80]
ELSE DEST[87:80] ← SRC1[87:80];
IF (MASK[95] = 1) THEN DEST[95:88] ← SRC2[95:88]
ELSE DEST[95:88] ←    SRC1[95:88];
IF (MASK[103] = 1) THEN DEST[103:96] ← SRC2[103:96]
ELSE DEST[103:96] ←    SRC1[103:96];
IF (MASK[111] = 1) THEN DEST[111:104] ← SRC2[111:104]
ELSE DEST[111:104] ← SRC1[111:104];
IF (MASK[119] = 1) THEN DEST[119:112] ← SRC2[119:112]
ELSE DEST[119:112] ← SRC1[119:112];
IF (MASK[127] = 1) THEN DEST[127:120] ← SRC2[127:120]
ELSE DEST[127:120] ← SRC1[127:120])
DEST[VLMAX-1:128] ← 0
```

VPBLENDVB (VEX.256 encoded version)

```
MASK ← SRC3
IF (MASK[7] == 1) THEN DEST[7:0] ← SRC2[7:0];
ELSE DEST[7:0] ← SRC1[7:0];
IF (MASK[15] == 1) THEN DEST[15:8] ←SRC2[15:8];
ELSE DEST[15:8] ← SRC1[15:8];
IF (MASK[23] == 1) THEN DEST[23:16] ←SRC2[23:16]
ELSE DEST[23:16] ← SRC1[23:16];
IF (MASK[31] == 1) THEN DEST[31:24] ← SRC2[31:24]
ELSE DEST[31:24] ← SRC1[31:24];
IF (MASK[39] == 1) THEN DEST[39:32] ← SRC2[39:32]
ELSE DEST[39:32] ← SRC1[39:32];
IF (MASK[47] == 1) THEN DEST[47:40] ← SRC2[47:40]
ELSE DEST[47:40] ← SRC1[47:40];
IF (MASK[55] == 1) THEN DEST[55:48] ← SRC2[55:48]
ELSE DEST[55:48] ← SRC1[55:48];
IF (MASK[63] == 1) THEN DEST[63:56] ←SRC2[63:56]
ELSE DEST[63:56] ← SRC1[63:56];
IF (MASK[71] == 1) THEN DEST[71:64] ←SRC2[71:64]
ELSE DEST[71:64] ← SRC1[71:64];
IF (MASK[79] == 1) THEN DEST[79:72] ← SRC2[79:72]
```

```
ELSE DEST[79:72] ← SRC1[79:72];
IF (MASK[87] == 1) THEN DEST[87:80] ← SRC2[87:80]
ELSE DEST[87:80] ← SRC1[87:80];
IF (MASK[95] == 1) THEN DEST[95:88] ← SRC2[95:88]
ELSE DEST[95:88] ← SRC1[95:88];
IF (MASK[103] == 1) THEN DEST[103:96] ← SRC2[103:96]
ELSE DEST[103:96] ← SRC1[103:96];
IF (MASK[111] == 1) THEN DEST[111:104] ← SRC2[111:104]
ELSE DEST[111:104] ← SRC1[111:104];
IF (MASK[119] == 1) THEN DEST[119:112] ← SRC2[119:112]
ELSE DEST[119:112] ← SRC1[119:112];
IF (MASK[127] == 1) THEN DEST[127:120] ← SRC2[127:120]
ELSE DEST[127:120] ← SRC1[127:120])
IF (MASK[135] == 1) THEN DEST[135:128] ← SRC2[135:128];
ELSE DEST[135:128] ← SRC1[135:128];
IF (MASK[143] == 1) THEN DEST[143:136] ← SRC2[143:136];
ELSE DEST[[143:136] ← SRC1[143:136];
IF (MASK[151] == 1) THEN DEST[151:144] ← SRC2[151:144]
ELSE DEST[151:144] ← SRC1[151:144];
IF (MASK[159] == 1) THEN DEST[159:152] ← SRC2[159:152]
ELSE DEST[159:152] ← SRC1[159:152];
IF (MASK[167] == 1) THEN DEST[167:160] ← SRC2[167:160]
ELSE DEST[167:160] ← SRC1[167:160];
IF (MASK[175] == 1) THEN DEST[175:168] ← SRC2[175:168]
ELSE DEST[175:168] ← SRC1[175:168];
IF (MASK[183] == 1) THEN DEST[183:176] ← SRC2[183:176]
ELSE DEST[183:176] ← SRC1[183:176];
IF (MASK[191] == 1) THEN DEST[191:184] ← SRC2[191:184]
ELSE DEST[191:184] ← SRC1[191:184];
IF (MASK[199] == 1) THEN DEST[199:192] ← SRC2[199:192]
ELSE DEST[199:192] ← SRC1[199:192];
IF (MASK[207] == 1) THEN DEST[207:200] ← SRC2[207:200]
ELSE DEST[207:200] ← SRC1[207:200]
IF (MASK[215] == 1) THEN DEST[215:208] ← SRC2[215:208]
ELSE DEST[215:208] ← SRC1[215:208];
IF (MASK[223] == 1) THEN DEST[223:216] ← SRC2[223:216]
ELSE DEST[223:216] ← SRC1[223:216];
IF (MASK[231] == 1) THEN DEST[231:224] ← SRC2[231:224]
ELSE DEST[231:224] ← SRC1[231:224];
IF (MASK[239] == 1) THEN DEST[239:232] ← SRC2[239:232]
ELSE DEST[239:232] ← SRC1[239:232];
IF (MASK[247] == 1) THEN DEST[247:240] ← SRC2[247:240]
ELSE DEST[247:240] ← SRC1[247:240];
IF (MASK[255] == 1) THEN DEST[255:248] ← SRC2[255:248]
ELSE DEST[255:248] ← SRC1[255:248]
```

Intel C/C++ Compiler Intrinsic Equivalent

(V)PBLENDVB: __m128i _mm_blendv_epi8 (__m128i v1, __m128i v2, __m128i mask);

VPBLENDVB: __m256i _mm256_blendv_epi8 (__m256i v1, __m256i v2, __m256i mask);

Flags Affected

None.

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4; additionally

#UD	If VEX.L = 1.
	If VEX.W = 1.

...

PREFETCH*h*—Prefetch Data Into Caches

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 18 /1	PREFETCHT0 *m8*	M	Valid	Valid	Move data from *m8* closer to the processor using T0 hint.
0F 18 /2	PREFETCHT1 *m8*	M	Valid	Valid	Move data from *m8* closer to the processor using T1 hint.
0F 18 /3	PREFETCHT2 *m8*	M	Valid	Valid	Move data from *m8* closer to the processor using T2 hint.
0F 18 /0	PREFETCHNTA *m8*	M	Valid	Valid	Move data from *m8* closer to the processor using NTA hint.

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 a locality hint:

- T0 (temporal data)—prefetch data into all levels of the cache hierarchy.
  - Pentium III processor—1st- or 2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T1 (temporal data with respect to first level cache)—prefetch data into level 2 cache and higher.
  - Pentium III processor—2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T2 (temporal data with respect to second level cache)—prefetch data into level 2 cache and higher.
  - Pentium III processor—2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure and into a location close to the processor, minimizing cache pollution.

— Pentium III processor—1st-level cache

— Pentium 4 and Intel Xeon processors—2nd-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.)

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 PREFETCH*h* 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 PREFETCH*h* 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 PREFETCH*h* instruction is not ordered with respect to the fence instructions (MFENCE, SFENCE, and LFENCE) or locked memory references. A PREFETCH*h* instruction is also unordered with respect to CLFLUSH instructions, other PREFETCH*h* 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

```
FETCH (m8);
```

Intel C/C++ Compiler Intrinsic Equivalent

```
void _mm_prefetch(char *p, int i)
```

The argument "*p" gives the address of the byte (and corresponding cache line) to be prefetched. The value "i" gives a constant (_MM_HINT_T0, _MM_HINT_T1, _MM_HINT_T2, or _MM_HINT_NTA) that specifies the type of prefetch operation to be performed.

Numeric Exceptions

None.

Exceptions (All Operating Modes)

#UD	If the LOCK prefix is used.

...

PREFETCHW—Prefetch Data into Caches in Anticipation of a Write

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 0D /1 PREFETCHW m8	A	V/V	PRFCHW	Move data from m8 closer to the processor in anticipation of a 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 cache line of data from memory that contains the byte specified with the source operand to a location in the 1st or 2nd level cache and invalidates all other cached instances of the line.

The source operand is a byte memory location. If the line selected is already present in the lowest level cache and is already in an exclusively owned state, no data movement occurs. Prefetches from non-writeback memory are ignored.

The PREFETCHW instruction is merely a hint and does not affect program behavior. If executed, this instruction moves data closer to the processor and invalidates any other cached copy in anticipation of the line being written to in the future.

The characteristic 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 with exclusive ownership from system memory regions that permit such accesses (that is, the WB memory type). A PREFETCHW 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 PREFETCHW instruction is not ordered with respect to the fence instructions (MFENCE, SFENCE, and LFENCE) or locked memory references. A PREFETCHW instruction is also unordered with respect to CLFLUSH instructions, other PREFETCHW 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

```
FETCH_WITH_EXCLUSIVE_OWNERSHIP (m8);
```

Flags Affected

All flags are affected

C/C++ Compiler Intrinsic Equivalent

`void _m_prefetchw( void * );`

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.

...

RDMSR—Read from Model Specific Register

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 32	RDMSR	NP	Valid	Valid	Read MSR specified by ECX into EDX:EAX.

NOTES:

* See IA-32 Architecture Compatibility section below.

Instruction Operand Encoding

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

Description

Reads the contents of a 64-bit model specific register (MSR) specified in the ECX register into registers EDX:EAX. (On processors that support the Intel 64 architecture, the high-order 32 bits of RCX are ignored.) The EDX register is loaded with the high-order 32 bits of the MSR and the EAX register is loaded with the low-order 32 bits. (On processors that support the Intel 64 architecture, the high-order 32 bits of each of RAX and RDX are cleared.) If fewer than 64 bits are implemented in the MSR being read, the values returned to EDX:EAX in unimplemented bit locations are undefined.

This instruction must be executed at privilege level 0 or in real-address mode; otherwise, a general protection exception #GP(0) will be generated. Specifying a reserved or unimplemented MSR address in ECX will also cause a general protection exception.

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

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

IA-32 Architecture Compatibility

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

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

```
EDX:EAX ← MSR[ECX];
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the value in ECX specifies a reserved or unimplemented MSR address.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If the value in ECX specifies a reserved or unimplemented MSR address.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

RDRAND—Read Random Number

Opcode*/ Instruction	**Op/ En**	**64/32 bit Mode Support**	**CPUID Feature Flag**	**Description**
0F C7 /6 RDRAND r16	M	V/V	RDRAND	Read a 16-bit random number and store in the destination register.
0F C7 /6 RDRAND r32	M	V/V	RDRAND	Read a 32-bit random number and store in the destination register.
REX.W + 0F C7 /6 RDRAND r64	M	V/I	RDRAND	Read a 64-bit random number and store in the destination register.

Instruction Operand Encoding

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

Description

Loads a hardware generated random value and store it in the destination register. The size of the random value is determined by the destination register size and operating mode. The Carry Flag indicates whether a random value is available at the time the instruction is executed. CF=1 indicates that the data in the destination is valid. Otherwise CF=0 and the data in the destination operand will be returned as zeros for the specified width. All other flags are forced to 0 in either situation. Software must check the state of CF=1 for determining if a valid random value has been returned, otherwise it is expected to loop and retry execution of RDRAND (see *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, Section 7.3.17, “Random Number Generator Instruction”).

This instruction is available at all privilege levels.

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

Operation

```
IF HW_RND_GEN.ready = 1
    THEN
        CASE of
            osize is 64: DEST[63:0] ← HW_RND_GEN.data;
            osize is 32: DEST[31:0] ← HW_RND_GEN.data;
            osize is 16: DEST[15:0] ← HW_RND_GEN.data;
        ESAC
        CF ← 1;
    ELSE
        CASE of
            osize is 64: DEST[63:0] ← 0;
            osize is 32: DEST[31:0] ← 0;
            osize is 16: DEST[15:0] ← 0;
        ESAC
        CF ← 0;
FI
OF, SF, ZF, AF, PF ← 0;
```

Flags Affected

The CF flag is set according to the result (see the "Operation" section above). The OF, SF, ZF, AF, and PF flags are set to 0.

Intel C/C++ Compiler Intrinsic Equivalent

RDRAND:	int _rdrand16_step(unsigned short *);
RDRAND:	int _rdrand32_step(unsigned int *);
RDRAND:	int _rdrand64_step(unsigned __int64 *);

Protected Mode Exceptions

#UD	If the LOCK prefix is used. If the F2H or F3H prefix is used. If CPUID.01H:ECX.RDRAND[bit 30] = 0.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

VBROADCAST—Broadcast Floating-Point Data

Opcode/ Instruction	**Op/ En**	**64/32-bit Mode**	**CPUID Feature Flag**	**Description**
VEX.128.66.0F38.W0 18 /r VBROADCASTSS *xmm1, m32*	RM	V/V	AVX	Broadcast single-precision floating-point element in mem to four locations in *xmm1.*
VEX.256.66.0F38.W0 18 /r VBROADCASTSS *ymm1, m32*	RM	V/V	AVX	Broadcast single-precision floating-point element in mem to eight locations in *ymm1.*
VEX.256.66.0F38.W0 19 /r VBROADCASTSD *ymm1, m64*	RM	V/V	AVX	Broadcast double-precision floating-point element in mem to four locations in *ymm1.*
VEX.256.66.0F38.W0 1A /r VBROADCASTF128 *ymm1, m128*	RM	V/V	AVX	Broadcast 128 bits of floating-point data in mem to low and high 128-bits in *ymm1.*
VEX.128.66.0F38.W0 18/r VBROADCASTSS *xmm1, xmm2*	RM	V/V	AVX2	Broadcast the low single-precision floating-point element in the source operand to four locations in *xmm1.*
VEX.256.66.0F38.W0 18 /r VBROADCASTSS *ymm1, xmm2*	RM	V/V	AVX2	Broadcast low single-precision floating-point element in the source operand to eight locations in *ymm1.*
VEX.256.66.0F38.W0 19 /r VBROADCASTSD *ymm1, xmm2*	RM	V/V	AVX2	Broadcast low double-precision floating-point element in the source operand to four locations in *ymm1.*

Instruction Operand Encoding

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

Description

Load floating point values from the source operand (second operand) and broadcast to all elements of the destination operand (first operand).

VBROADCASTSD and VBROADCASTF128 are only supported as 256-bit wide versions. VBROADCASTSS is supported in both 128-bit and 256-bit wide versions.

Memory and register source operand syntax support of 256-bit instructions depend on the processor's enumeration of the following conditions with respect to CPUID.1:ECX.AVX[bit 28] and CPUID.(EAX=07H, ECX=0H):EBX.AVX2[bit 5]:

- If CPUID.1:ECX.AVX = 1 and CPUID.(EAX=07H, ECX=0H):EBX.AVX2 = 0: the destination operand is a YMM register. The source operand support can be either a 32-bit, 64-bit, or 128-bit memory location. Register source encodings are reserved and will #UD.
- If CPUID.1:ECX.AVX = 1 and CPUID.(EAX=07H, ECX=0H):EBX.AVX2 = 1: the destination operand is a YMM register. The source operand support can be a register or memory location.

Note: In VEX-encoded versions, VEX.vvvv is reserved and must be 1111b otherwise instructions will #UD. An attempt to execute VBROADCASTSD or VBROADCASTF128 encoded with VEX.L= 0 will cause an #UD exception. Attempts to execute any VBROADCAST* instruction with VEX.W = 1 will cause #UD.

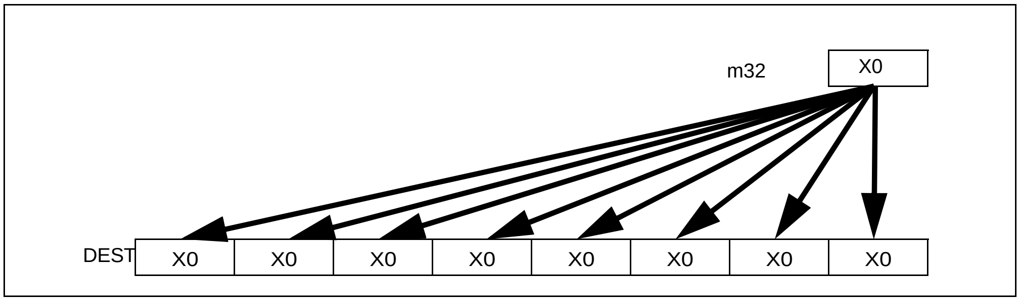


Figure 4-27 VBROADCASTSS Operation (VEX.256 encoded version)

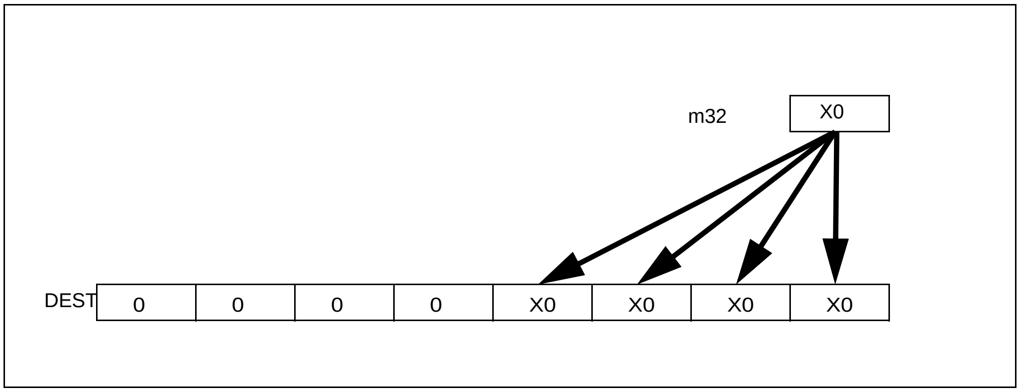


Figure 4-28 VBROADCASTSS Operation (128-bit version)

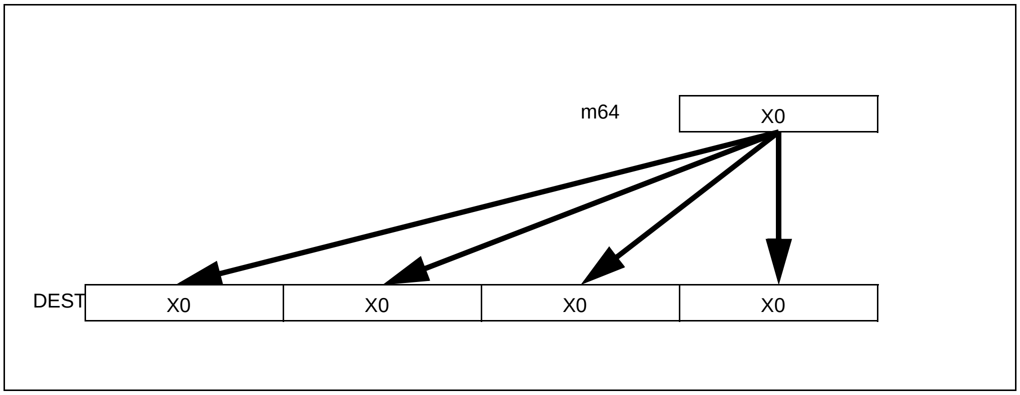


Figure 4-29 VBROADCASTSD Operation

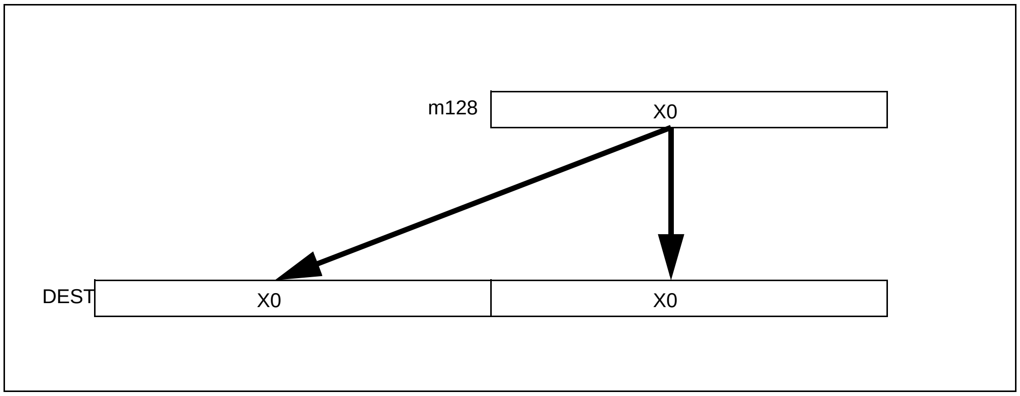


Figure 4-30 VBROADCASTF128 Operation

Operation

VBROADCASTSS (128 bit version)

```
temp ← SRC[31:0]
DEST[31:0] ← temp
DEST[63:32] ← temp
DEST[95:64] ← temp
DEST[127:96] ← temp
DEST[VLMAX-1:128] ← 0
```

VBROADCASTSS (VEX.256 encoded version)

```
temp ← SRC[31:0]
DEST[31:0] ← temp
DEST[63:32] ← temp
DEST[95:64] ← temp
DEST[127:96] ← temp
DEST[159:128] ← temp
DEST[191:160] ← temp
DEST[223:192] ← temp
DEST[255:224] ← temp
```

VBROADCASTSD (VEX.256 encoded version)

```
temp ← SRC[63:0]
DEST[63:0] ← temp
DEST[127:64] ← temp
DEST[191:128] ← temp
DEST[255:192] ← temp
```

VBROADCASTF128

```
temp ← SRC[127:0]
DEST[127:0] ← temp
DEST[VLMAX-1:128] ← temp
```

Intel C/C++ Compiler Intrinsic Equivalent

VBROADCASTSS:	__m128 _mm_broadcast_ss(float *a);
VBROADCASTSS:	__m256 _mm256_broadcast_ss(float *a);
VBROADCASTSD:	__m256d _mm256_broadcast_sd(double *a);
VBROADCASTF128:	__m256 _mm256_broadcast_ps(__m128 * a);
VBROADCASTF128:	__m256d _mm256_broadcast_pd(__m128d * a);

Flags Affected

None.

Other Exceptions

See Exceptions Type 6; additionally

#UD	If VEX.L = 0 for VBROADCASTSD, If VEX.L = 0 for VBROADCASTF128, If VEX.W = 1.

...

XGETBV—Get Value of Extended Control Register

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 01 D0	XGETBV	NP	Valid	Valid	Reads an XCR specified by ECX into EDX:EAX.

Instruction Operand Encoding

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

Description

Reads the contents of the extended control register (XCR) specified in the ECX register into registers EDX:EAX. (On processors that support the Intel 64 architecture, the high-order 32 bits of RCX are ignored.) The EDX register is loaded with the high-order 32 bits of the XCR and the EAX register is loaded with the low-order 32 bits. (On processors that support the Intel 64 architecture, the high-order 32 bits of each of RAX and RDX are cleared.) If fewer than 64 bits are implemented in the XCR being read, the values returned to EDX:EAX in unimplemented bit locations are undefined.

Specifying a reserved or unimplemented XCR in ECX causes a general protection exception.

Currently, only XCR0 (the XFEATURE_ENABLED_MASK register) is supported. Thus, all other values of ECX are reserved and will cause a #GP(0).

Operation

```
EDX:EAX ← XCR[ECX];
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XGETBV: unsigned __int64 _xgetbv(unsigned int);

Protected Mode Exceptions

#GP(0)	If an invalid XCR is specified in ECX.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0. If CR4.OSXSAVE[bit 18] = 0. If the LOCK prefix is used. If 66H, F3H or F2H prefix is used.

Real-Address Mode Exceptions

#GP	If an invalid XCR is specified in ECX.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0. If CR4.OSXSAVE[bit 18] = 0. If the LOCK prefix is used. If 66H, F3H or F2H prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

XRSTOR—Restore Processor Extended States

Opcode	**Instruction**	**Op/ En**	**64-Bit Mode**	**Compat/ Leg Mode**	**Description**
0F AE /5	XRSTOR *mem*	M	Valid	Valid	Restore processor extended states from *memory*. The states are specified by EDX:EAX
REX.W+ 0F AE /5	XRSTOR64 *mem*	M	Valid	N.E.	Restore processor extended states from *memory*. The states are specified by EDX:EAX

Instruction Operand Encoding

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

Description

Performs a full or partial restore of the enabled processor states using the state information stored in the memory address specified by the source operand. The implicit EDX:EAX register pair specifies a 64-bit restore mask.

The format of the XSAVE/XRSTOR area is shown in Table 4-17. The memory layout of the XSAVE/XRSTOR area may have holes between save areas written by the processor as a result of the processor not supporting certain processor extended states or system software not supporting certain processor extended states. There is no relationship between the order of XCR0 bits and the order of the state layout. States corresponding to higher and lower XCR0 bits may be intermingled in the layout.

Table 4-17 General Layout of XSAVE/XRSTOR Save Area

Save Areas	Offset (Byte)	Size (Bytes)
FPU/SSE SaveArea[1]	0	512
Header	512	64
Reserved (Ext_Save_Area_2)	CPUID.(EAX=0DH, ECX=2):EBX	CPUID.(EAX=0DH, ECX=2):EAX
Reserved(Ext_Save_Area_4)[2]	CPUID.(EAX=0DH, ECX=4):EBX	CPUID.(EAX=0DH, ECX=4):EAX
Reserved(Ext_Save_Area_3)	CPUID.(EAX=0DH, ECX=3):EBX	CPUID.(EAX=0DH, ECX=3):EAX
Reserved(...)	...	...

NOTES:

1. Bytes 464:511 are available for software use. XRSTOR ignores the value contained in bytes 464:511 of an XSAVE SAVE image.
2. State corresponding to higher and lower XCR0 bits may be intermingled in layout.

XRSTOR operates on each subset of the processor state or a processor extended state in one of three ways (depending on the corresponding bit in XCR0 (XFEATURE_ENABLED_MASK register), the restore mask EDX:EAX, and the save mask XSAVE.HEADER.XSTATE_BV in memory):

- Updates the processor state component using the state information stored in the respective save area (see Table 4-17) of the source operand, if the corresponding bit in XCR0, EDX:EAX, and XSAVE.HEADER.XSTATE_BV are all 1.
- Writes certain registers in the processor state component using processor-supplied values (see Table 4-19) without using state information stored in respective save area of the memory region, if the corresponding bit in XCR0 and EDX:EAX are both 1, but the corresponding bit in XSAVE.HEADER.XSTATE_BV is 0.
- The processor state component is unchanged, if the corresponding bit in XCR0 or EDX:EAX is 0.

The format of the header section (XSAVE.HEADER) of the XSAVE/XRSTOR area is shown in Table 4-18.

Table 4-18 XSAVE.HEADER Layout

15 8	7 0	Byte Offset from Header	Byte Offset from XSAVE/XRSTOR Area
Rsrvd (Must be 0)	XSTATE_BV	**0**	**512**
Reserved	Rsrvd (Must be 0)	**16**	**528**
Reserved	Reserved	**32**	**544**
Reserved	Reserved	**48**	**560**

If a processor state component is not enabled in XCR0 but the corresponding save mask bit in XSAVE.HEADER.XSTATE_BV is 1, an attempt to execute XRSTOR will cause a #GP(0) exception. Software may specify all 1's in the implicit restore mask EDX:EAX, so that all the enabled processors states in XCR0 are restored from state information stored in memory or from processor supplied values. When using all 1's as the restore mask, software is required to determine the total size of the XSAVE/XRSTOR save area (specified as source operand) to fit all enabled processor states by using the value enumerated in CPUID.(EAX=0D, ECX=0):EBX. While it's legal to set any bit in the EDX:EAX mask to 1, it is strongly recommended to set only the bits that are required to save/restore specific states.

An attempt to restore processor states with writing 1s to reserved bits in certain registers (see Table 4-20) will cause a #GP(0) exception.

Because bit 63 of XCR0 is reserved for future bit vector expansion, it will not be used for any future processor state feature, and XRSTOR will ignore bit 63 of EDX:EAX (EDX[31]).

Table 4-19 Processor Supplied Init Values XRSTOR May Use

Processor State Component	Processor Supplied Register Values
x87 FPU State	FCW ← 037FH; FTW ← 0FFFFH; FSW ← 0H; FPU CS ← 0H; FPU DS ← 0H; FPU IP ← 0H; FPU DP ← 0; ST0-ST7 ← 0;
SSE State[1]	If 64-bit Mode: XMM0-XMM15 ← 0H; Else XMM0-XMM7 ← 0H

NOTES:

1. MXCSR state is not updated by processor supplied values. MXCSR state can only be updated by XRSTOR from state information stored in XSAVE/XRSTOR area.

Table 4-20 Reserved Bit Checking and XRSTOR

Processor State Component	Reserved Bit Checking
X87 FPU State	None
SSE State	Reserved bits of MXCSR

A source operand not aligned to 64-byte boundary (for 64-bit and 32-bit modes) will result in a general-protection (#GP) exception. In 64-bit mode, the upper 32 bits of RDX and RAX are ignored.

Operation

```
/* The alignment of the x87 and SSE fields in the XSAVE area is the same as in FXSAVE area*/

RS_TMP_MASK[62:0] ← (EDX[30:0] << 32 ) OR EAX[31:0];
ST_TMP_MASK[62:0] ← SRCMEM.HEADER.XSTATE_BV[62:0];
IF ( ( (XCR0[62:0] XOR 7FFFFFFF_FFFFFFFFH ) AND ST_TMP_MASK[62:0] ) )
    THEN
        #GP(0)
ELSE
    FOR i = 0, 62 STEP 1
        IF ( RS_TMP_MASK[i] and XCR0[i] )
            THEN
                IF ( ST_TMP_MASK[i] )
                    CASE ( i ) OF
                    0:   Processor state[x87 FPU] ← SRCMEM. FPUSSESave_Area[FPU];
                    1:   Processor state[SSE] ← SRCMEM. FPUSSESave_Area[SSE];
                         // MXCSR is loaded as part of the SSE state
                    DEFAULT:  // i corresponds to a valid sub-leaf index of CPUID leaf 0DH
                         Processor state[i] ← SRCMEM. Ext_Save_Area[ i ];
                    ESAC;
                ELSE
                    Processor extended state[i] ← Processor supplied values; (see Table 4-19)
```

```
                    CASE ( i ) OF
                    1:    MXCSR ← SRCMEM. FPUSSESave_Area[SSE];
                    ESAC;
                FI;
        FI;
    NEXT;
FI;
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XRSTOR: void _xrstor(void * , unsigned __int64);

XRSTOR: void _xrstor64(void * , unsigned __int64);

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
	If a bit in XCR0 is 0 and the corresponding bit in HEADER.XSTATE_BV field of the source operand is 1.
	If bytes 23:8 of HEADER is not zero.
	If attempting to write any reserved bits of the MXCSR register with 1.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
	If a bit in XCR0 is 0 and the corresponding bit in HEADER.XSTATE_BV field of the source operand is 1.
	If bytes 23:8 of HEADER is not zero.
	If attempting to write any reserved bits of the MXCSR register with 1.

#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Protected Mode

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
	If a bit in XCR0 is 0 and the corresponding bit in XSAVE.HEADER.XSTATE_BV is 1.
	If bytes 23:8 of HEADER is not zero.
	If attempting to write any reserved bits of the MXCSR register with 1.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

...

XSAVE—Save Processor Extended States

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F AE /4	XSAVE *mem*	M	Valid	Valid	Save processor extended states to *memory*. The states are specified by EDX:EAX
REX.W+ 0F AE /4	XSAVE64 *mem*	M	Valid	N.E.	Save processor extended states to *memory*. The states are specified by EDX:EAX

Instruction Operand Encoding

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

Description

Performs a full or partial save of the enabled processor state components to a memory address specified in the destination operand. A full or partial save of the processor states is specified by an implicit mask operand via the register pair, EDX:EAX. The destination operand is a memory location that must be 64-byte aligned.

The implicit 64-bit mask operand in EDX:EAX specifies the subset of enabled processor state components to save into the XSAVE/XRSTOR save area. The XSAVE/XRSTOR save area comprises of individual save area for each processor state components and a header section, see Table 4-17. Each component save area is written if both the corresponding bits in the save mask operand and in XCR0 (the XFEATURE_ENABLED_MASK register) are 1. A processor state component save area is not updated if either one of the corresponding bits in the mask operand or in XCR0 is 0. If the mask operand (EDX:EAX) contains all 1's, all enabled processor state components in XCR0 are written to the respective component save area.

The bit assignment used for the EDX:EAX register pair matches XCR0 (see chapter 2 of Vol. 3B). For the XSAVE instruction, software can specify "1" in any bit position of EDX:EAX, irrespective of whether the corresponding bit position in XCR0 is valid for the processor. The bit vector in EDX:EAX is "anded" with XCR0 to determine which save area will be written. While it's legal to set any bit in the EDX:EAX mask to 1, it is strongly recommended to set only the bits that are required to save/restore specific states. When specifying 1 in any bit position of EDX:EAX mask, software is required to determine the total size of the XSAVE/XRSTOR save area (specified as destination operand) to fit all enabled processor states by using the value enumerated in CPUID.(EAX=0D, ECX=0):EBX.

The content layout of the XSAVE/XRSTOR save area is architecturally defined to be extendable and enumerated via the sub-leaves of CPUID.0DH leaf. The extendable framework of the XSAVE/XRSTOR layout is depicted by Table 4-17. The layout of the XSAVE/XRSTOR save area is fixed and may contain non-contiguous individual save areas. The XSAVE/XRSTOR save area is not compacted if some features are not saved or are not supported by the processor and/or by system software.

The layout of the register fields of first 512 bytes of the XSAVE/XRSTOR is the same as the FXSAVE/FXRSTOR area (refer to “FXSAVE—Save x87 FPU, MMX Technology, and SSE State” on page 357). But XSAVE/XRSTOR organizes the 512 byte area as x87 FPU states (including FPU operation states, x87/MMX data registers), MXCSR (including MXCSR_MASK), and XMM registers.

Bytes 464:511 are available for software use. The processor does not write to bytes 464:511 when executing XSAVE.

The processor writes 1 or 0 to each HEADER.XSTATE_BV[i] bit field of an enabled processor state component in a manner that is consistent to XRSTOR's interaction with HEADER.XSTATE_BV (see the operation section of XRSTOR instruction). If a processor implementation discern that a processor state component is in its initialized state (according to Table 4-19) it may modify the corresponding bit in the HEADER.XSTATE_BV as ‘0’.

A destination operand not aligned to 64-byte boundary (in either 64-bit or 32-bit modes) will result in a general-protection (#GP) exception being generated. In 64-bit mode, the upper 32 bits of RDX and RAX are ignored.

Operation

```
TMP_MASK[62:0] ← ( (EDX[30:0] << 32 ) OR EAX[31:0] ) AND XCR0[62:0];
FOR i = 0, 62 STEP 1
    IF ( TMP_MASK[i] = 1) THEN
        THEN
            CASE ( i ) of
                0: DEST.FPUSSESAVE_Area[x87 FPU] ← processor state[x87 FPU];
                1: DEST.FPUSSESAVE_Area[SSE] ← processor state[SSE];
```

```
                    // SSE state include MXCSR
                DEFAULT: // i corresponds to a valid sub-leaf index of CPUID leaf 0DH
                    DEST.Ext_Save_Area[ i ] ← processor state[i] ;
            ESAC:
            DEST.HEADER.XSTATE_BV[i] ← INIT_FUNCTION[i];
    FI;
NEXT;
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XSAVE:	void _xsave(void * , unsigned __int64);
XSAVE:	void _xsave64(void * , unsigned __int64);

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#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.
	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

...

XSAVEOPT—Save Processor Extended States Optimized

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F AE /6 XSAVEOPT *mem*	M	V/V	XSAVEOPT	Save processor extended states specified in EDX:EAX to *memory,* optimizing the state save operation if possible.
REX.W + 0F AE /6 XSAVEOPT64 *mem*	M	V/V	XSAVEOPT	Save processor extended states specified in EDX:EAX to *memory,* optimizing the state save operation if possible.

Instruction Operand Encoding

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

Description

XSAVEOPT performs a full or partial save of the enabled processor state components to a memory address specified in the destination operand. A full or partial save of the processor states is specified by an implicit mask operand via the register pair, EDX:EAX. The destination operand is a memory location that must be 64-byte aligned. The hardware may optimize the manner in which data is saved. The performance of this instruction will be equal or better than using the XSAVE instruction.

The implicit 64-bit mask operand in EDX:EAX specifies the subset of enabled processor state components to save into the XSAVE/XRSTOR save area. The XSAVE/XRSTOR save area comprises of individual save area for each processor state components and a header section, see Table 4-17.

The bit assignment used for the EDX:EAX register pair matches XCR0 (the XFEATURE_ENABLED_MASK register). For the XSAVEOPT instruction, software can specify "1" in any bit position of EDX:EAX, irrespective of whether the corresponding bit position in XCR0 is valid for the processor. The bit vector in EDX:EAX is "anded" with XCR0 to determine which save area will be written. While it's legal to set any bit in the EDX:EAX mask to 1, it is strongly recommended to set only the bits that are required to save/restore specific states. When specifying 1 in any bit position of EDX:EAX mask, software is required to determine the total size of the XSAVE/XRSTOR save area (specified as destination operand) to fit all enabled processor states by using the value enumerated in CPUID.(EAX=0D, ECX=0):EBX.

The content layout of the XSAVE/XRSTOR save area is architecturally defined to be extendable and enumerated via the sub-leaves of CPUID.0DH leaf. The extendable framework of the XSAVE/XRSTOR layout is depicted by Table 4-17. The layout of the XSAVE/XRSTOR save area is fixed and may contain non-contiguous individual save areas. The XSAVE/XRSTOR save area is not compacted if some features are not saved or are not supported by the processor and/or by system software.

The layout of the register fields of first 512 bytes of the XSAVE/XRSTOR is the same as the FXSAVE/FXRSTOR area. But XSAVE/XRSTOR organizes the 512 byte area as x87 FPU states (including FPU operation states, x87/MMX data registers), MXCSR (including MXCSR_MASK), and XMM registers.

The processor writes 1 or 0 to each.HEADER.XSTATE_BV[i] bit field of an enabled processor state component in a manner that is consistent to XRSTOR's interaction with HEADER.XSTATE_BV.

The state updated to the XSAVE/XRSTOR area may be optimized as follows:

- If the state is in its initialized form, the corresponding XSTATE_BV bit may be set to 0, and the corresponding processor state component that is indicated as initialized will not be saved to memory.

A processor state component save area is not updated if either one of the corresponding bits in the mask operand or in XCR0 is 0. The processor state component that is updated to the save area is computed by bit-wise AND of the mask operand (EDX:EAX) with XCR0.

HEADER.XSTATE_BV is updated to reflect the data that is actually written to the save area. A "1" bit in the header indicates the contents of the save area corresponding to that bit are valid. A "0" bit in the header indicates that the state corresponding to that bit is in its initialized form. The memory image corresponding to a "0" bit may or may not contain the correct (initialized) value since only the header bit (and not the save area contents) is updated when the header bit value is 0. XRSTOR will ensure the correct value is placed in the register state regardless of the value of the save area when the header bit is zero.

XSAVEOPT Usage Guidelines

When using the XSAVEOPT facility, software must be aware of the following guidelines:

1. The processor uses a tracking mechanism to determine which state components will be written to memory by the XSAVEOPT instruction. The mechanism includes three sub-conditions that are recorded internally each time XRSTOR is executed and evaluated on the invocation of the next XSAVEOPT. If a change is detected in any one of these sub-conditions, XSAVEOPT will behave exactly as XSAVE. The three sub-conditions are:
   - current CPL of the logical processor
   - indication whether or not the logical processor is in VMX non-root operation
   - linear address of the XSAVE/XRSTOR area
2. Upon allocation of a new XSAVE/XRSTOR area and before an XSAVE or XSAVEOPT instruction is used, the save area header (HEADER.XSTATE) must be initialized to zeroes for proper operation.
3. XSAVEOPT is designed primarily for use in context switch operations. The values stored by the XSAVEOPT instruction depend on the values previously stored in a given XSAVE area.

4. Manual modifications to the XSAVE area between an XRSTOR instruction and the matching XSAVEOPT may result in data corruption.
5. For optimization to be performed properly, the XRSTOR XSAVEOPT pair must use the same segment when referencing the XSAVE area and the base of that segment must be unchanged between the two operations.
6. Software should avoid executing XSAVEOPT into a buffer from which it hadn't previously executed a XRSTOR. For newly allocated buffers, software can execute XRSTOR with the linear address of the buffer and a restore mask of EDX:EAX = 0. Executing XRSTOR(0:0) doesn't restore any state, but ensures expected operation of the XSAVEOPT instruction.
7. The XSAVE area can be moved or even paged, but the contents at the linear address of the save area at an XSAVEOPT must be the same as that when the previous XRSTOR was performed.

A destination operand not aligned to 64-byte boundary (in either 64-bit or 32-bit modes) will result in a general-protection (#GP) exception being generated. In 64-bit mode, the upper 32 bits of RDX and RAX are ignored.

Operation

```
TMP_MASK[62:0]    (EDX[30:0] << 32 ) OR EAX[31:0] ) AND XCR0[62:0];
FOR i = 0, 62 STEP 1
    IF (TMP_MASK[i] = 1)
    THEN
        If not HW_CAN_OPTIMIZE_SAVE
        THEN
            CASE ( i ) of
                0: DEST.FPUSSESAVE_Area[x87 FPU]   processor state[x87 FPU];
                1: DEST.FPUSSESAVE_Area[SSE]   processor state[SSE];
                    // SSE state include MXCSR
                2: DEST.EXT_SAVE_Area2[YMM]   processor state[YMM];
                DEFAULT: // i corresponds to a valid sub-leaf index of CPUID leaf 0DH
                    DEST.Ext_Save_Area[ i ]   processor state[i] ;
            ESAC:
        FI;
        DEST.HEADER.XSTATE_BV[i]   INIT_FUNCTION[i];
    FI;
NEXT;
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XSAVEOPT: void _xsaveopt(void * , unsigned __int64);

XSAVEOPT: void _xsaveopt64(void * , unsigned __int64);

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.

	If CPUID.(EAX=0DH, ECX=01H):EAX.XSAVEOPT[bit 0] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CPUID.(EAX=0DH, ECX=01H):EAX.XSAVEOPT[bit 0] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#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.
	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CPUID.(EAX=0DH, ECX=01H):EAX.XSAVEOPT[bit 0] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.

...

XSETBV—Set Extended Control Register

Opcode	**Instruction**	**Op/ En**	**64-Bit Mode**	**Compat/ Leg Mode**	**Description**
0F 01 D1	XSETBV	NP	Valid	Valid	Write the value in EDX:EAX to the XCR specified by ECX.

Instruction Operand Encoding

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

Description

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

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

Currently, only XCR0 (the XFEATURE_ENABLED_MASK register) is supported. Thus, all other values of ECX are reserved and will cause a #GP(0). Note that bit 0 of XCR0 (corresponding to x87 state) must be set to 1; the instruction will cause a #GP(0) if an attempt is made to clear this bit. Additionally, bit 1 of XCR0 (corresponding to AVX state) and bit 2 of XCR0 (corresponding to SSE state) must be set to 1 when using AVX registers; the instruction will cause a #GP(0) if an attempt is made to set XCR0[2:1] = 10.

Operation

```
XCR[ECX] ← EDX:EAX;
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XSETBV:	`void _xsetbv( unsigned int, unsigned __int64);`

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If an invalid XCR is specified in ECX.
	If the value in EDX:EAX sets bits that are reserved in the XCR specified by ECX.
	If an attempt is made to clear bit 0 of XCR0.
	If an attempt is made to set XCR0[2:1] = 10.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.
	If 66H, F3H or F2H prefix is used.

Real-Address Mode Exceptions

#GP	If an invalid XCR is specified in ECX.
	If the value in EDX:EAX sets bits that are reserved in the XCR specified by ECX.
	If an attempt is made to clear bit 0 of XCR0.
	If an attempt is made to set XCR0[2:1] = 10.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0.
	If CR4.OSXSAVE[bit 18] = 0.
	If the LOCK prefix is used.

If 66H, F3H or F2H prefix is used.

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

10. Updates to Appendix A, Volume 2C

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

...

Table A-2 One-byte Opcode Map: (00H — F7H) *

	0	1	2	3	4	5	6	7
0	ADD						PUSH ES^{i64}	POP ES^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
1	ADC						PUSH SS^{i64}	POP SS^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
2	AND						SEG=ES (Prefix)	DAA^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
3	XOR						SEG=SS (Prefix)	AAA^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
4	INC^{i64} general register / REX^{o64} Prefixes							
	eAX REX	eCX REX.B	eDX REX.X	eBX REX.XB	eSP REX.R	eBP REX.RB	eSI REX.RX	eDI REX.RXB
5	PUSH^{d64} general register							
	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSHA^{i64}/ PUSHAD^{i64}	POPA^{i64}/ POPAD^{i64}	BOUND^{i64} Gv, Ma	ARPL^{i64} Ew, Gw MOVSXD^{o64} Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)
7	Jcc^{f64}, Jb - Short-displacement jump on condition							
	O	NO	B/NAE/C	NB/AE/NC	Z/E	NZ/NE	BE/NA	NBE/A
8	Immediate Grp 1^{1A}				TEST		XCHG	
	Eb, Ib	Ev, Iz	Eb, Ib^{i64}	Ev, Ib	Eb, Gb	Ev, Gv	Eb, Gb	Ev, Gv
9	NOP PAUSE(F3) XCHG r8, rAX	XCHG word, double-word or quad-word register with rAX						
		rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
A	MOV				MOVS/B Yb, Xb	MOVS/W/D/Q Yv, Xv	CMPS/B Xb, Yb	CMPS/W/D Xv, Yv
	AL, Ob	rAX, Ov	Ob, AL	Ov, rAX				

	0	1	2	3	4	5	6	7
B	MOV immediate byte into byte register							
	AL/R8L, Ib	CL/R9L, Ib	DL/R10L, Ib	BL/R11L, Ib	AH/R12L, Ib	CH/R13L, Ib	DH/R14L, Ib	BH/R15L, Ib
C	Shift Grp 2^{1A}		near RET^{f64} Iw	near RET^{f64}	LES^{i64} Gz, Mp VEX+2byte	LDS^{i64} Gz, Mp VEX+1byte	Grp 11^{1A} - MOV	
	Eb, Ib	Ev, Ib					Eb, Ib	Ev, Iz
D	Shift Grp 2^{1A}				AAM^{i64} Ib	AAD^{i64} Ib		XLAT/ XLATB
	Eb, 1	Ev, 1	Eb, CL	Ev, CL				
E	LOOPNE^{f64}/ LOOPNZ^{f64} Jb	LOOPE^{f64}/ LOOPZ^{f64} Jb	LOOP^{f64} Jb	JrCXZ^{f64}/ Jb	IN		OUT	
					AL, Ib	eAX, Ib	Ib, AL	Ib, eAX
F	LOCK (Prefix)		REPNE XACQUIRE (Prefix)	REP/REPE XRELEASE (Prefix)	HLT	CMC	Unary Grp 3^{1A}	
							Eb	Ev

Table A-2 One-byte Opcode Map: (08H — FFH) *

	8	9	A	B	C	D	E	F
0	OR						PUSH CS^{i64}	2-byte escape (Table A-3)
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
1	SBB						PUSH DS^{i64}	POP DS^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
2	SUB						SEG=CS (Prefix)	DAS^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
3	CMP						SEG=DS (Prefix)	AAS^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
4	DEC^{i64} general register / REX^{o64} Prefixes							
	eAX REX.W	eCX REX.WB	eDX REX.WX	eBX REX.WXB	eSP REX.WR	eBP REX.WRB	eSI REX.WRX	eDI REX.WRXB
5	POP^{d64} into general register							
	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSH^{d64} Iz	IMUL Gv, Ev, Iz	PUSH^{d64} Ib	IMUL Gv, Ev, Ib	INS/ INSB Yb, DX	INS/ INSW/ INSD Yz, DX	OUTS/ OUTSB DX, Xb	OUTS/ OUTSW/ OUTSD DX, Xz
7	Jcc^{f64}, Jb- Short displacement jump on condition							
	S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
8	MOV				MOV Ev, Sw	LEA Gv, M	MOV Sw, Ew	Grp 1A^{1A} POP^{d64} Ev
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev				
9	CBW/ CWDE/ CDQE	CWD/ CDQ/ CQO	far CALL^{i64} Ap	FWAIT/ WAIT	PUSHF/D/Q ^{d64}/ Fv	POPF/D/Q ^{d64}/ Fv	SAHF	LAHF
A	TEST		STOS/B Yb, AL	STOS/W/D/Q Yv, rAX	LODS/B AL, Xb	LODS/W/D/Q rAX, Xv	SCAS/B AL, Yb	SCAS/W/D/Q rAX, Xv
	AL, Ib	rAX, Iz						
B	MOV immediate word or double into word, double, or quad register							
	rAX/r8, Iv	rCX/r9, Iv	rDX/r10, Iv	rBX/r11, Iv	rSP/r12, Iv	rBP/r13, Iv	rSI/r14, Iv	rDI/r15 , Iv
C	ENTER	LEAVE^{d64}	far RET	far RET	INT 3	INT	INTO^{i64}	IRET/D/Q
	Iw, Ib		Iw			Ib		
D	ESC (Escape to coprocessor instruction set)							
E	near CALL^{f64}	JMP			IN		OUT	
	Jz	near^{f64} Jz	far^{i64} Ap	short^{f64} Jb	AL, DX	eAX, DX	DX, AL	DX, eAX
F	CLC	STC	CLI	STI	CLD	STD	INC/DEC Grp 4^{1A}	INC/DEC Grp 5^{1A}

NOTES:

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

…

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

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
80-83	1	mem, 11B		ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
8F	1A	mem, 11B		POP							
C0,C1 reg, imm D0, D1 reg, 1 D2, D3 reg, CL	2	mem, 11B		ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
F6, F7	3	mem, 11B		TEST Ib/Iz		NOT	NEG	MUL AL/rAX	IMUL AL/rAX	DIV AL/rAX	IDIV AL/rAX
FE	4	mem, 11B		INC Eb	DEC Eb						
FF	5	mem, 11B		INC Ev	DEC Ev	near CALL^{f64} Ev	far CALL Ep	near JMP^{f64} Ev	far JMP Mp	PUSH^{d64} Ev	
0F 00	6	mem, 11B		SLDT Rv/Mw	STR Rv/Mw	LLDT Ew	LTR Ew	VERR Ew	VERW Ew		
0F 01	7	mem		SGDT Ms	SIDT Ms	LGDT Ms	LIDT Ms	SMSW Mw/Rv		LMSW Ew	INVLPG Mb
		11B		VMCALL (001) VMLAUNCH (010) VMRESUME (011) VMXOFF (100)	MONITOR (000) MWAIT (001) CLAC (010) STAC (011)	XGETBV (000) XSETBV (001) VMFUNC (100) XEND (101) XTEST (110)					SWAPGS ^{o64}(000) RDTSCP (001)
0F BA	8	mem, 11B						BT	BTS	BTR	BTC
0F C7	9	mem			CMPXCH8B Mq CMPXCHG16B Mdq					VMPTRLD Mq	VMPTRST Mq
			66							VMCLEAR Mq	
			F3							VMXON Mq	VMPTRST Mq
		11B								RDRAND Rv	RDSEED Rv
0F B9	10	mem									
		11B									
C6	11	mem		MOV Eb, Ib							
		11B									XABORT (000) Ib
C7		mem		MOV Ev, Iz							
		11B									XBEGIN (000) Jz
0F 71	12	mem									
		11B				psrlw Nq, Ib		psraw Nq, Ib		psllw Nq, Ib	
			66			vpsrlw Hx,Ux,Ib		vpsraw Hx,Ux,Ib		vpsllw Hx,Ux,Ib	
0F 72	13	mem									
		11B				psrld Nq, Ib		psrad Nq, Ib		pslld Nq, Ib	
			66			vpsrld Hx,Ux,Ib		vpsrad Hx,Ux,Ib		vpslld Hx,Ux,Ib	
0F 73	14	mem									
		11B				psrlq Nq, Ib				psllq Nq, Ib	
			66			vpsrlq Hx,Ux,Ib	vpsrldq Hx,Ux,Ib			vpsllq Hx,Ux,Ib	vpslldq Hx,Ux,Ib

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

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

NOTES:

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

...

11. Updates to Chapter 2, Volume 3A

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

...

2.3 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.

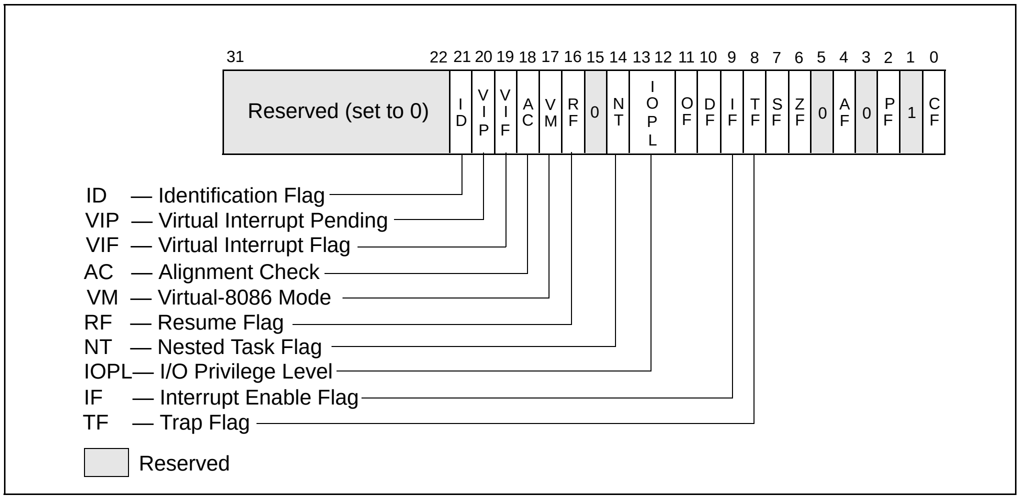


Figure 2-5 System Flags in the EFLAGS Register

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 16, “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 (bit 18)** — Set this flag and the AM flag in control register CR0 to enable alignment checking of memory references; clear the AC flag and/or the AM flag to disable alignment checking. 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.

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.

...

12. Updates to Chapter 5, Volume 3A

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

...

5.8.5.1 Stack Switching in 64-bit Mode

Although protection-check rules for call gates are unchanged from 32-bit mode, stack-switch changes in 64-bit mode are different.

When stacks are switched as part of a 64-bit mode privilege-level change through a call gate, a new SS (stack segment) descriptor is not loaded; 64-bit mode only loads an inner-level RSP from the TSS. The new SS is forced to NULL and the SS selector’s RPL field is forced to the new CPL. The new SS is set to NULL in order to handle nested far transfers (far CALL, INTn, interrupts and exceptions). The old SS and RSP are saved on the new stack.

On a subsequent far RET, the old SS is popped from the stack and loaded into the SS register. See Table 5-2.

Table 5-2 64-Bit-Mode Stack Layout After Far CALL with CPL Change

32-bit Mode			IA-32e mode		
Old SS Selector	+12			+24	Old SS Selector
Old ESP	+8			+16	Old RSP
CS Selector	+4			+8	Old CS Selector
EIP	0	**ESP**	**RSP**	0	RIP
< 4 Bytes >					< 8 Bytes >

In 64-bit mode, stack operations resulting from a privilege-level-changing far call or far return are eight-bytes wide and change the RSP by eight. The mode does not support the automatic parameter-copy feature found in 32-bit mode. The call-gate count field is ignored. Software can access the old stack, if necessary, by referencing the old stack-segment selector and stack pointer saved on the new process stack.

In 64-bit mode, far RET is allowed to load a NULL SS under certain conditions. If the target mode is 64-bit mode and the target CPL< >3, IRET allows SS to be loaded with a NULL selector. If the called procedure itself is interrupted, the NULL SS is pushed on the stack frame. On the subsequent far RET, the NULL SS on the stack acts as a flag to tell the processor not to load a new SS descriptor.

...

13. Updates to Chapter 6, Volume 3A

Change bars show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

6.14.4 Stack Switching in IA-32e Mode

The IA-32 architecture provides a mechanism to automatically switch stack frames in response to an interrupt. The 64-bit extensions of Intel 64 architecture implement a modified version of the legacy stack-switching mechanism and an alternative stack-switching mechanism called the interrupt stack table (IST).

In IA-32 modes, the legacy IA-32 stack-switch mechanism is unchanged. In IA-32e mode, the legacy stack-switch mechanism is modified. When stacks are switched as part of a 64-bit mode privilege-level change (resulting from an interrupt), a new SS descriptor is not loaded. IA-32e mode loads only an inner-level RSP from the TSS. The new SS selector is forced to NULL and the SS selector's RPL field is set to the new CPL. The new SS is set to NULL in order to handle nested far transfers (far CALL, INT, interrupts and exceptions). The old SS and RSP are saved on the new stack (Figure 6-8). On the subsequent IRET, the old SS is popped from the stack and loaded into the SS register.

In summary, a stack switch in IA-32e mode works like the legacy stack switch, except that a new SS selector is not loaded from the TSS. Instead, the new SS is forced to NULL.

...

14. Updates to Chapter 7, Volume 3A

Change bars show changes to Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

7.3 TASK SWITCHING

The processor transfers execution to another task in one of four cases:

- The current program, task, or procedure executes a JMP or CALL instruction to a TSS descriptor in the GDT.
- The current program, task, or procedure executes a JMP or CALL instruction to a task-gate descriptor in the GDT or the current LDT.
- An interrupt or exception vector points to a task-gate descriptor in the IDT.
- The current task executes an IRET when the NT flag in the EFLAGS register is set.

JMP, CALL, and IRET instructions, as well as interrupts and exceptions, are all mechanisms for redirecting a program. The referencing of a TSS descriptor or a task gate (when calling or jumping to a task) or the state of the NT flag (when executing an IRET instruction) determines whether a task switch occurs.

The processor performs the following operations when switching to a new task:

1. Obtains the TSS segment selector for the new task as the operand of the JMP or CALL instruction, from a task gate, or from the previous task link field (for a task switch initiated with an IRET instruction).
2. Checks that the current (old) task is allowed to switch to the new task. Data-access privilege rules apply to JMP and CALL instructions. The CPL of the current (old) task and the RPL of the segment selector for the new task must be less than or equal to the DPL of the TSS descriptor or task gate being referenced. Exceptions, interrupts (except for interrupts generated by the INT *n* instruction), and the IRET instruction are permitted to switch tasks regardless of the DPL of the destination task-gate or TSS descriptor. For interrupts generated by the INT *n* instruction, the DPL is checked.
3. Checks that the TSS descriptor of the new task is marked present and has a valid limit (greater than or equal to 67H).
4. Checks that the new task is available (call, jump, exception, or interrupt) or busy (IRET return).
5. Checks that the current (old) TSS, new TSS, and all segment descriptors used in the task switch are paged into system memory.
6. If the task switch was initiated with a JMP or IRET instruction, the processor clears the busy (B) flag in the current (old) task's TSS descriptor; if initiated with a CALL instruction, an exception, or an interrupt: the busy (B) flag is left set. (See Table 7-2.)
7. If the task switch was initiated with an IRET instruction, the processor clears the NT flag in a temporarily saved image of the EFLAGS register; if initiated with a CALL or JMP instruction, an exception, or an interrupt, the NT flag is left unchanged in the saved EFLAGS image.
8. Saves the state of the current (old) task in the current task's TSS. The processor finds the base address of the current TSS in the task register and then copies the states of the following registers into the current TSS: all the general-purpose registers, segment selectors from the segment registers, the temporarily saved image of the EFLAGS register, and the instruction pointer register (EIP).
9. If the task switch was initiated with a CALL instruction, an exception, or an interrupt, the processor will set the NT flag in the EFLAGS loaded from the new task. If initiated with an IRET instruction or JMP instruction, the NT flag will reflect the state of NT in the EFLAGS loaded from the new task (see Table 7-2).

10. If the task switch was initiated with a CALL instruction, JMP instruction, an exception, or an interrupt, the processor sets the busy (B) flag in the new task’s TSS descriptor; if initiated with an IRET instruction, the busy (B) flag is left set.
11. Loads the task register with the segment selector and descriptor for the new task's TSS.
12. The TSS state is loaded into the processor. This includes the LDTR register, the PDBR (control register CR3), the EFLAGS register, the EIP register, the general-purpose registers, and the segment selectors. A fault during the load of this state may corrupt architectural state. (If paging is not enabled, a PDBR value is read from the new task's TSS, but it is not loaded into CR3.)
13. The descriptors associated with the segment selectors are loaded and qualified. Any errors associated with this loading and qualification occur in the context of the new task and may corrupt architectural state.

NOTES

If all checks and saves have been carried out successfully, the processor commits to the task switch. If an unrecoverable error occurs in steps 1 through 11, the processor does not complete the task switch and insures that the processor is returned to its state prior to the execution of the instruction that initiated the task switch.

If an unrecoverable error occurs in step 12, architectural state may be corrupted, but an attempt will be made to handle the error in the prior execution environment. If an unrecoverable error occurs after the commit point (in step 13), the processor completes the task switch (without performing additional access and segment availability checks) and generates the appropriate exception prior to beginning execution of the new task.

If exceptions occur after the commit point, the exception handler must finish the task switch itself before allowing the processor to begin executing the new task. See Chapter 6, “Interrupt 10—Invalid TSS Exception (#TS),” for more information about the affect of exceptions on a task when they occur after the commit point of a task switch.

14. Begins executing the new task. (To an exception handler, the first instruction of the new task appears not to have been executed.)

The state of the currently executing task is always saved when a successful task switch occurs. If the task is resumed, execution starts with the instruction pointed to by the saved EIP value, and the registers are restored to the values they held when the task was suspended.

When switching tasks, the privilege level of the new task does not inherit its privilege level from the suspended task. The new task begins executing at the privilege level specified in the CPL field of the CS register, which is loaded from the TSS. Because tasks are isolated by their separate address spaces and TSSs and because privilege rules control access to a TSS, software does not need to perform explicit privilege checks on a task switch.

Table 7-1 shows the exception conditions that the processor checks for when switching tasks. It also shows the exception that is generated for each check if an error is detected and the segment that the error code references. (The order of the checks in the table is the order used in the P6 family processors. The exact order is model specific and may be different for other IA-32 processors.) Exception handlers designed to handle these exceptions may be subject to recursive calls if they attempt to reload the segment selector that generated the exception. The cause of the exception (or the first of multiple causes) should be fixed before reloading the selector.

...

15. 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.

...

10.5.1 Local Vector Table

The local vector table (LVT) allows software to specify the manner in which the local interrupts are delivered to the processor core. It consists of the following 32-bit APIC registers (see Figure 10-8), one for each local interrupt:

- **LVT CMCI Register (FEE0 02F0H)** — Specifies interrupt delivery when an overflow condition of corrected machine check error count reaching a threshold value occurred in a machine check bank supporting CMCI (see Section 15.5.1, "CMCI Local APIC Interface").
- **LVT Timer Register (FEE0 0320H)** — Specifies interrupt delivery when the APIC timer signals an interrupt (see Section 10.5.4, "APIC Timer").
- **LVT Thermal Monitor Register (FEE0 0330H)** — Specifies interrupt delivery when the thermal sensor generates an interrupt (see Section 14.5.2, "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 18.12.5.8, "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 Figure 10-8, some of these fields and flags are not available (and reserved) for some entries.

The setup information that can be specified in the registers of the LVT table is as follows:

Vector	Interrupt vector number.
Delivery Mode	Specifies the type of interrupt to be sent to the processor. Some delivery modes will only operate as intended when used in conjunction with a specific trigger mode. The allowable delivery modes are as follows:

000 (Fixed)	Delivers the interrupt specified in the vector field.
010 (SMI)	Delivers an SMI interrupt to the processor core through the processor's local SMI signal path. When using this delivery mode, the vector field should be set to 00H for future compatibility.
100 (NMI)	Delivers an NMI interrupt to the processor. The vector information is ignored.

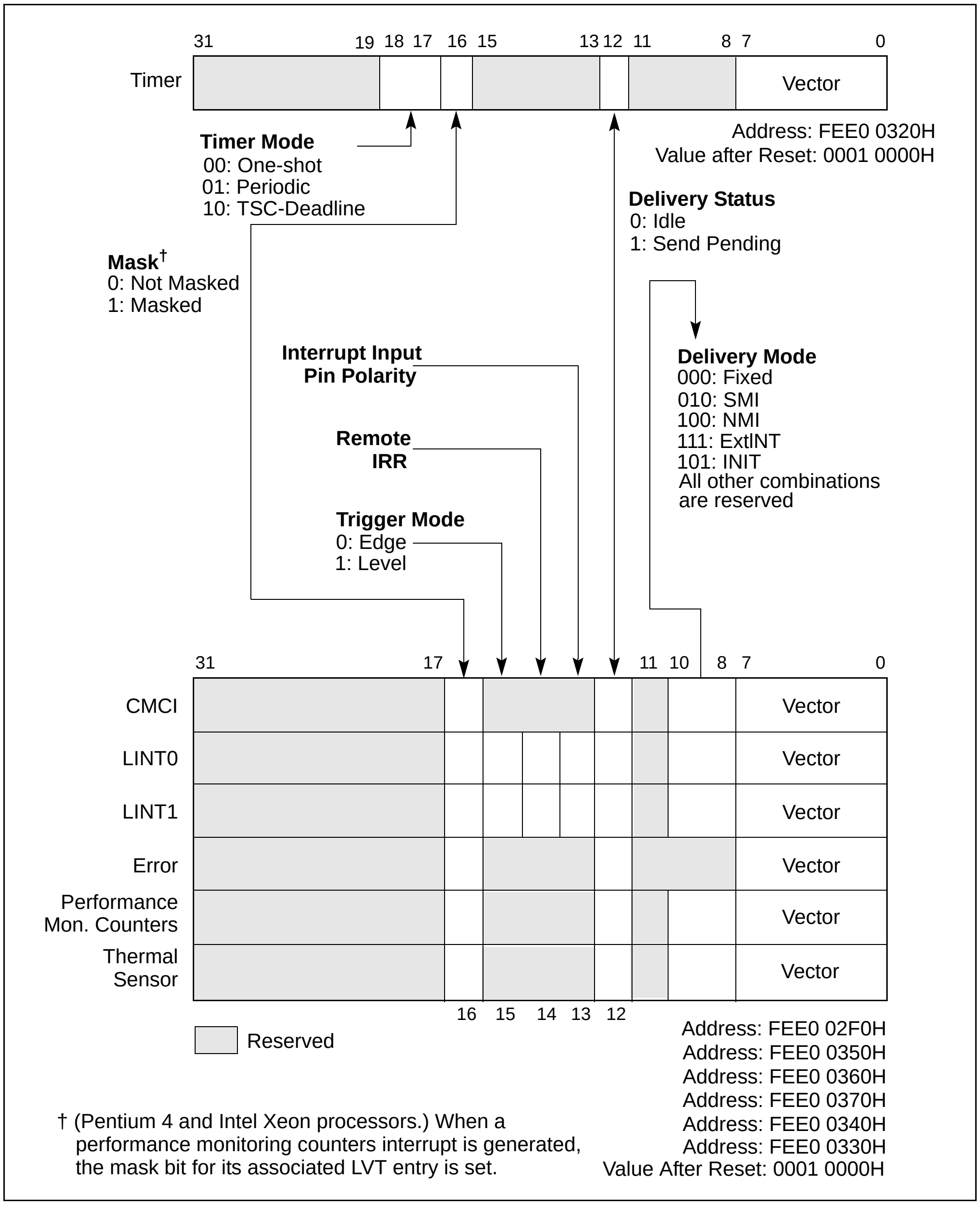


Figure 10-8 Local Vector Table (LVT)

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 control-

ler. 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,

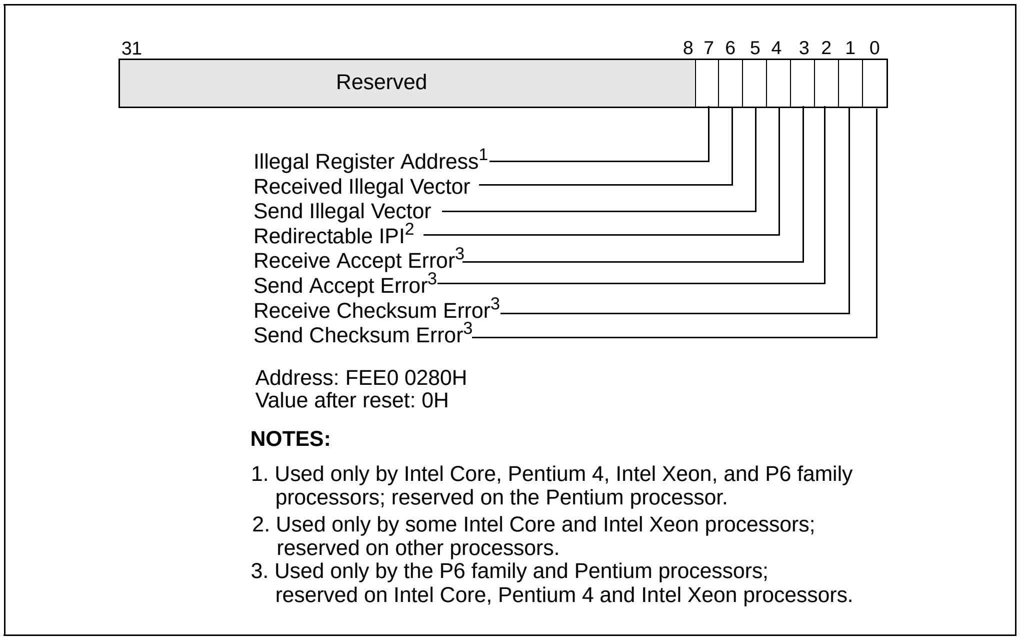
(10b) TSC-Deadline mode using absolute target value in IA32_TSC_DEADLINE MSR (see Section 10.5.4.1),

(11b) is reserved.

...

10.5.3 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:



Address: FEE0 0280H
Value after reset: 0H

NOTES:

1. Used only by Intel Core, Pentium 4, Intel Xeon, and P6 family processors; reserved on the Pentium processor.
2. Used only by some Intel Core and Intel Xeon processors; reserved on other processors.
3. Used only by the P6 family and Pentium processors; reserved on Intel Core, Pentium 4 and Intel Xeon processors.

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 “redirectible 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 be 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.

...

16. Updates to Chapter 13, Volume 3A

Change bars show changes to Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A:* System Programming Guide, Part 1.

...

CHAPTER 13
SYSTEM PROGRAMMING FOR INSTRUCTION SET EXTENSIONS AND PROCESSOR EXTENDED STATES

This chapter describes system programming features for instruction set extensions operating on the processor state extension known as the SSE state (XMM registers, MXCSR) and for other processor extended states. Instruction set extensions operating on the SSE state include the streaming SIMD extensions (SSE), streaming SIMD extensions 2 (SSE2), streaming SIMD extensions 3 (SSE3), Supplemental SSE3 (SSSE3), and SSE4. Collectively, these are called **SSE extensions** and the corresponding instructions **SSE instructions**.

Sections 13.1 through 13.5 cover system programming requirements to enable the SSE extensions, providing operating system or executive support for the SSE extensions, SIMD floating-point exceptions, exception handling, and task (context) switching.

Processor extended states refer to extensions to the Intel 64 architecture that will allow system executives to implement support for multiple processor state extensions that may be introduced over time without requiring the system executive to be modified each time a new processor state extension is introduced. System programming for managing processor extended states is described in the sections starting 13.6.

13.1 PROVIDING OPERATING SYSTEM SUPPORT FOR SSE EXTENSIONS

To use SSE extensions, the operating system or executive must provide support for initializing the processor to use these extensions, for handling SIMD floating-point exceptions, and for using either FXSAVE and FXRSTOR (Section 10.5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*) or the XSAVE feature set (Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*) to manage context. The following sections provide system programming guidelines for this support. Because SSE extensions share the same state, experience the same sets of non-numerical and numerical exception behavior, these guidelines that apply to SSE also apply to other sets of SIMD extensions that operate on the same processor state and subject to the same sets of non-numerical and numerical exception behavior.

Chapter 11, "Programming with Streaming SIMD Extensions 2 (SSE2)," and Chapter 12, "Programming with SSE3, SSSE3 and SSE4," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, discuss support for SSE/SSE2/SSE3/SSSE3/SSE4 from an applications point of view program.

13.1.1 Adding Support to an Operating System for SSE Extensions

The following guidelines describe functions that an operating system or executive must perform to support SSE extensions:

1. Check that the processor supports the SSE extensions.
2. Check that the processor supports the FXSAVE and FXRSTOR instructions or the XSAVE feature set.
3. Provide an initialization for the SSE states.
4. Provide support for the FXSAVE and FXRSTOR instructions or the XSAVE feature set.
5. Provide support (if necessary) in non-numeric exception handlers for exceptions generated by the SSE instructions.
6. Provide an exception handler for the SIMD floating-point exception (#XM).

The following sections describe how to implement each of these guidelines.

13.1.2 Checking for CPU Support

If the processor attempts to execute an unsupported SSE instruction, the processor generates an invalid-opcode exception (#UD). Before an operating system or executive attempts to use SSE extensions, it should check that support is present by confirming the following bit values returned by the CPUID instruction:

- CPUID.1:EDX.SSE[bit 25] = 1
- CPUID.1:EDX.SSE2[bit 26] = 1
- CPUID.1:ECX.SSE3[bit 0] = 1
- CPUID.1:ECX.SSSE3[bit 9] = 1
- CPUID.1:ECX.SSE4_1[bit 19] = 1
- CPUID.1:ECX.SSE4_2[bit 20] = 1

(To use POPCNT instruction, software must check CPUID.1:ECX.POPCNT[bit 23] = 1.)

Separate checks must be made to ensure that the processor supports either FXSAVE and FXRSTOR or the XSAVE feature set. See Section 10.5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* and Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, respectively.

13.1.3 Initialization of the SSE Extensions

The operating system or executive should carry out the following steps to set up SSE extensions for use by application programs:

1. Set CR4.OSFXSR[bit 9] = 1. Setting this flag implies that the operating system provides facilities for saving and restoring SSE state using FXSAVE and FXRSTOR instructions or the XSAVE feature set. These instructions may be used to save the SSE state during task switches and when invoking the SIMD floating-point exception (#XM) handler (see Section 13.4, “Saving the SSE State on Task or Context Switches,” and Section 13.1.5, “Providing an Handler for the SIMD Floating-Point Exception (#XM),” respectively).

   If the processor does not support the FXSAVE and FXRSTOR instructions, attempting to set the OSFXSR flag causes a general-protection exception (#GP) to be generated.

2. Set CR4.OSXMMEXCPT[bit 10] = 1. Setting this flag implies that the operating system provides an SIMD floating-point exception (#XM) handler (see Section 13.1.5, “Providing an Handler for the SIMD Floating-Point Exception (#XM)”).

NOTE

> The OSFXSR and OSXMMEXCPT bits in control register CR4 must be set by the operating system. The processor has no other way of detecting operating-system support for the FXSAVE and FXRSTOR instructions or for handling SIMD floating-point exceptions.

3. Clear CR0.EM[bit 2] = 0. This action disables emulation of the x87 FPU, which is required when executing SSE instructions (see Section 2.5, “Control Registers”).
4. Set CR0.MP[bit 1] = 1. This setting is the required setting for Intel 64 and IA-32 processors that support the SSE extensions (see Section 9.2.1, “Configuring the x87 FPU Environment”).

Table 13-1 and Table 13-2 show the actions of the processor when an SSE instruction is executed, depending on the following:

- OSFXSR and OSXMMEXCPT flags in control register CR4
- SSE/SSE2/SSE3/SSSE3/SSE4 feature flags returned by CPUID
- EM, MP, and TS flags in control register CR0

Table 13-1 Action Taken for Combinations of OSFXSR, OSXMMEXCPT, SSE, SSE2, SSE3, EM, MP, and TS[1]

CR4		CPUID	CR0 Flags			Action
OSFXSR	OSXMMEXCPT	SSE, SSE2, SSE3[2], SSE4_1[3]	EM	MP[4]	TS	
0	X[5]	X	X	1	X	#UD exception.
1	X	0	X	1	X	#UD exception.
1	X	1	1	1	X	#UD exception.
1	0	1	0	1	0	Execute instruction; #UD exception if unmasked SIMD floating-point exception is detected.
1	1	1	0	1	0	Execute instruction; #XM exception if unmasked SIMD floating-point exception is detected.
1	X	1	0	1	1	#NM exception.

NOTES:

1. For execution of any SSE instruction except the PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, and CLFLUSH instructions.
2. Exception conditions due to CR4.OSFXSR or CR4.OSXMMEXCPT do not apply to FISTTP.

3. Only applies to DPPS, DPPD, ROUNDPS, ROUNDPD, ROUNDSS, ROUNDSD.
4. For processors that support the MMX instructions, the MP flag should be set.
5. X = Don't care.

Table 13-2 Action Taken for Combinations of OSFXSR, SSSE3, SSE4, EM, and TS

CR4	CPUID	CR0 Flags		Action
OSFXSR	SSSE3 SSE4_1[1] SSE4_2[2]	EM	TS	
0	X[3]	X	X	#UD exception.
1	0	X	X	#UD exception.
1	1	1	X	#UD exception.
1	1	0	1	#NM exception.

NOTES:

1. Applies to SSE4_1 instructions except DPPS, DPPD, ROUNDPS, ROUNDPD, ROUNDSS, ROUNDSD.
2. Applies to SSE4_2 instructions except CRC32 and POPCNT.
3. X = Don't care.

The SIMD floating-point exception mask bits (bits 7 through 12), the flush-to-zero flag (bit 15), the denormals-are-zero flag (bit 6), and the rounding control field (bits 13 and 14) in the MXCSR register should be left in their default values of 0. This permits the application to determine how these features are to be used.

13.1.4 Providing Non-Numeric Exception Handlers for Exceptions Generated by the SSE Instructions

SSE instructions can generate the same type of memory-access exceptions (such as page faults and limit violations) and other non-numeric exceptions as other Intel 64 and IA-32 architecture instructions generate.

Ordinarily, existing exception handlers can handle these and other non-numeric exceptions without code modification. However, depending on the mechanisms used in existing exception handlers, some modifications might need to be made.

The SSE extensions can generate the non-numeric exceptions listed below:

- Memory Access Exceptions:
  — Stack-segment fault (#SS).
  — General protection exception (#GP). Executing most SSE instructions with an unaligned 128-bit memory reference generates a general-protection exception. (The MOVUPS and MOVUPD instructions allow unaligned a loads or stores of 128-bit memory locations, without generating a general-protection exception.) A 128-bit reference within the stack segment that is not aligned to a 16-byte boundary will also generate a general-protection exception, instead a stack-segment fault exception (#SS).
  — Page fault (#PF).
  — Alignment check (#AC). When enabled, this type of alignment check operates on operands that are less than 128-bits in size: 16-bit, 32-bit, and 64-bit. To enable the generation of alignment check exceptions, do the following:
    - Set the AM flag (bit 18 of control register CR0)
    - Set the AC flag (bit 18 of the EFLAGS register)

    - CPL must be 3

  If alignment check exceptions are enabled, 16-bit, 32-bit, and 64-bit misalignment will be detected for the MOVUPD and MOVUPS instructions; detection of 128-bit misalignment is not guaranteed and may vary with implementation.

- System Exceptions:
  - — Invalid-opcode exception (#UD). This exception is generated when executing SSE instructions under the following conditions:
    - SSE/SSE2/SSE3/SSSE3/SSE4_1/SSE4_2 feature flags returned by CPUID are set to 0. This condition does not affect the CLFLUSH instruction, nor POPCNT.
    - The CLFSH feature flag returned by the CPUID instruction is set to 0. This exception condition only pertains to the execution of the CLFLUSH instruction.
    - The POPCNT feature flag returned by the CPUID instruction is set to 0. This exception condition only pertains to the execution of the POPCNT instruction.
    - The EM flag (bit 2) in control register CR0 is set to 1, regardless of the value of TS flag (bit 3) of CR0. This condition does not affect the PAUSE, PREFETCH*h*, MOVNTI, SFENCE, LFENCE, MFENCE, CLFLUSH, CRC32 and POPCNT instructions.
    - The OSFXSR flag (bit 9) in control register CR4 is set to 0. This condition does not affect the PSHUFW, MOVNTQ, MOVNTI, PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, CLFLUSH, CRC32 and POPCNT instructions.
    - Executing a instruction that causes a SIMD floating-point exception when the OSXMMEXCPT flag (bit 10) in control register CR4 is set to 0. See Section 13.5.1, “Using the TS Flag to Control the Saving of the x87 FPU and SSE State.”
  - — Device not available (#NM). This exception is generated by executing a SSE instruction when the TS flag (bit 3) of CR0 is set to 1.

Other exceptions can occur during delivery of the above exceptions.

13.1.5 Providing an Handler for the SIMD Floating-Point Exception (#XM)

SSE instructions do not generate numeric exceptions on packed integer operations. They can generate the following numeric (SIMD floating-point) exceptions on packed and scalar single-precision and double-precision floating-point operations.

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

These SIMD floating-point exceptions (with the exception of the denormal operand exception) are defined in the IEEE Standard 754 for Binary Floating-Point Arithmetic and represent the same conditions that cause x87 FPU floating-point error exceptions (#MF) to be generated for x87 FPU instructions.

Each of these exceptions can be masked, in which case the processor returns a reasonable result to the destination operand without invoking an exception handler. However, if any of these exceptions are left unmasked, detection of the exception condition results in a SIMD floating-point exception (#XM) being generated. See Chapter 6, “Interrupt 19—SIMD Floating-Point Exception (#XM).”

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),” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*, 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 OSXMMEXCPT flag (bit 10) must be set in control register CR4.

13.1.5.1 Numeric Error flag and IGNNE#

SSE extensions ignore the NE flag in control register CR0 (that is, they treat it as if it were always set) and the IGNNE# pin. When an unmasked SIMD floating-point exception is detected, it is always reported by generating a SIMD floating-point exception (#XM).

13.2 EMULATION OF SSE EXTENSIONS

The Intel 64 and IA-32 architectures do not support emulation of the SSE instructions, as they do for x87 FPU instructions.

The EM flag in control register CR0 (provided to invoke emulation of x87 FPU instructions) cannot be used to invoke emulation of SSE instructions. If an SSE instruction is executed when CR0.EM = 1, an invalid opcode exception (#UD) is generated. See Table 13-1.

13.3 SAVING AND RESTORING SSE STATE

The SSE state consists of the state of the XMM and MXCSR registers. Intel recommends the following method for saving and restoring this state:

- Execute the FXSAVE, XSAVE, or XSAVEOPT instruction to save the state of the XMM and MXCSR registers to memory.
- Execute the FXRSTOR or XRSTOR instruction to restore the state of the XMM and MXCSR registers from the image saved in memory earlier.

This save and restore method is required for all operating systems. See Section 13.5, “Designing OS Facilities for Saving x87 FPU and SSE State Automatically on Task or Context Switches.” See Section 10.5 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* for information about FXSAVE and FXRSTOR; see Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* for the XSAVE feature set.

In some cases, applications may choose to save only the XMM and MXCSR registers in the following manner:

- Execute MOVDQ instructions to save the contents of the XMM registers to memory.
- Execute a STMXCSR instruction to save the state of the MXCSR register to memory.

Such applications must restore the XMM and MXCSR registers as follows:

- Execute MOVDQ instructions to load the saved contents of the XMM registers from memory into the XMM registers.
- Execute a LDMXCSR instruction to restore the state of the MXCSR register from memory.

13.4 SAVING THE SSE STATE ON TASK OR CONTEXT SWITCHES

When switching from one task or context to another, it is often necessary to save the SSE state. FXSAVE and FXRSTOR instructions provide a simple method for saving and restoring this state, as does the XSAVE feature set. See Section 10.5 and Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*. Guidelines for writing such procedures are in Section 13.5, "Designing OS Facilities for Saving x87 FPU and SSE State Automatically on Task or Context Switches."

13.5 DESIGNING OS FACILITIES FOR SAVING X87 FPU AND SSE STATE AUTOMATICALLY ON TASK OR CONTEXT SWITCHES

The x87 FPU, SSE, and AVX state consist of the state of the x87 FPU, XMM, and MXCSR registers. The FXSAVE and FXRSTOR instructions provide a fast method for saving ad restoring this state, as does the XSAVE feature set.

Older operating systems may use FSAVE/FNSAVE and FRSTOR to save the x87 FPU state. These facilities can be extended to save and restore SSE state by substituting FXSAVE and FXRSTOR or the XSAVE feature set in place of FSAVE/FNSAVE and FRSTOR.

If task or content switching facilities are written from scratch, any of several approaches may be taken for using the FXSAVE and FXRSTOR instructions of the XSAVE feature set to save and restore x87 FPU and SSE state:

- The operating system can require applications that are intended be run as tasks take responsibility for saving the state of the x87 FPU, XMM, and MXCSR registers prior to a task suspension during a task switch and for restoring the registers when the task is resumed. This approach is appropriate for cooperative multitasking operating systems, where the application has control over (or is able to determine) when a task switch is about to occur and can save state prior to the task switch.
- The operating system can take the responsibility for automatically saving the x87 FPU, and MXCSR registers as part of the task switch process (using the FXSAVE, XSAVE, or XSAVEOPT instructions) and automatically restoring the state of the registers when a suspended task is resumed (using the FXRSTOR or XRSTOR instructions). Here, the x87 FPU and SSE states must be saved as part of the task state. This approach is appropriate for preemptive multitasking operating systems, where the application cannot know when it is going to be preempted and cannot prepare in advance for task switching. Here, the operating system is responsible for saving and restoring the task and the x87 FPU and SSE states when necessary.
- The operating system can take the responsibility for saving the x87 FPU, XMM, and MXCSR registers as part of the task switch process, but delay the saving of the x87 FPU and SSE state until an x87 FPU, MMX, or SSE instruction is actually executed by the new task. Using this approach, the x87 FPU and SSE state is saved only if an x87 FPU, MMX, or SSE instruction needs to be executed in the new task. (See Section 13.5.1, "Using the TS Flag to Control the Saving of the x87 FPU and SSE State," for more information.)

13.5.1 Using the TS Flag to Control the Saving of the x87 FPU and SSE State

Saving the x87 FPU and SSE state using FXSAVE, XSAVE, or XSAVEOPT requires processor overhead. If the new task does not access x87 FPU, XMM, and MXCSR registers, an operating system might avoid overhead by not automatically saving the state on a task switch.

The TS flag in control register CR0 is provided to allow the operating system to delay saving the x87 FPU and SSE state until an instruction that actually accesses this state is encountered in a new task. When the TS flag is set, the processor monitors the instruction stream for x87 FPU, MMX, SSE instructions. When the processor detects one of these instructions, it raises a device-not-available exception (#NM) prior to executing the instruction. The #NM exception handler can then be used to save the x87 FPU and SSE state for the previous task (using an FXSAVE, XSAVE, or XSAVEOPT instruction) and load the x87 FPU and SSE state for the current task (using an FXRSTOR or XRSOTR instruction). If the task never encounters an x87 FPU, MMX, or SSE instruction, the device-not-available exception will not be raised and a task state will not be saved unnecessarily.

NOTE

The CRC32 and POPCNT instructions do not operate on the x87 FPU or SSE state. They operate on the general-purpose registers and are not involved with the techniques described above.

The TS flag can be set either explicitly (by executing a MOV instruction to control register CR0) or implicitly (using the IA-32 architecture's native task switching mechanism). When the native task switching mechanism is used, the processor automatically sets the TS flag on a task switch. After the device-not-available handler has saved the x87 FPU and SSE state, it should execute the CLTS instruction to clear the TS flag.

Figure 13-1 gives an example of an operating system that implements x87 FPU and SSE state saving using the TS flag. In this example, task A is the currently running task and task B is the new task. The operating system maintains a save area for the x87 FPU and SSE state for each task and defines a variable (x87_SSE_StateOwner) that indicates the task that "owns" the state. In this example, task A is the current owner.

On a task switch, the operating system task switching code must execute the following pseudo-code to set the TS flag according to the current owner of the x87 FPU and SSE state. If the new task (task B in this example) is not the current owner of this state, the TS flag is set to 1; otherwise, it is set to 0.

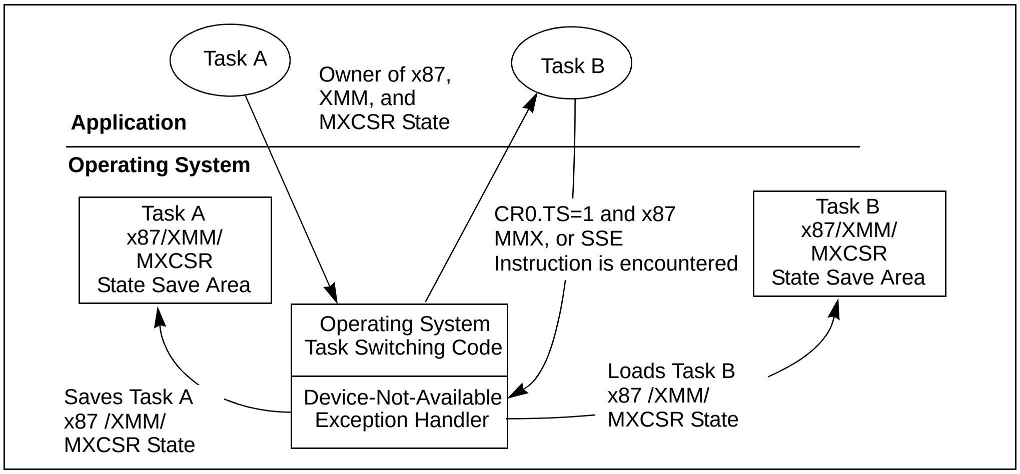


Figure 13-1 Example of Saving the x87 FPU and SSE State During an Operating-System Controlled Task Switch

```
IF Task_Being_Switched_To ≠ x87_XMM_MXCSR_StateOwner
  THEN
    CR0.TS ← 1;
  ELSE
    CR0.TS ← 0;
FI;
```

If a new task attempts to access an x87 FPU, XMM, or MXCSR register while the TS flag is set to 1, a device-not-available exception (#NM) is generated. The device-not-available exception handler executes the following pseudo-code (for FXSAVE and FXRSTOR).

```
FXSAVE "To x87/XMM/MXCSR State Save Area for Current
   x87_MMX_MXCSR_StateOwner";
FXRSTOR "x87/XMM/MXCSR State From Current Task's
   x87/XMM/MXCSR State Save Area";
x87_XMM_MXCSR_StateOwner ← Current_Task;
CR0.TS ← 0;
```

This exception handler code performs the following tasks:

- Saves the x87 FPU, XMM, or MXCSR registers in the state save area for the current owner of the x87 FPU, XMM, and MXCSR state.
- Restores the x87 FPU, XMM, or MXCSR registers from the new task's save area for the x87 FPU, XMM, and MXCSR state.
- Updates the current x87 FPU/XMM/MXCSR state owner to be the current task.
- Clears the TS flag.

13.6 THE XSAVE FEATURE SET AND PROCESSOR EXTENDED STATE MANAGEMENT

The XSAVE feature set includes the following:

- An extensible data layout for existing and future processor state extensions. The layout of the XSAVE area extends from the 512-byte FXSAVE/FXRSTOR layout to provide compatibility and migration path from managing the legacy FXSAVE/FXRSTOR area. The XSAVE area is described in more detail in Section 13.4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.
- CPUID enhancements for feature enumeration. See Section 13.2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.
- Control register enhancement and dedicated register for enabling each processor extended state. See Section 13.3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.
- Instructions to save state to and restore state from the XSAVE area. See Section 13.6 through Section 13.8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

13.7 INTEROPERABILITY OF THE XSAVE FEATURE SET AND FXSAVE/FXRSTOR

The FXSAVE instruction writes x87 FPU and SSE state information to a 512-byte FXSAVE save area. FXRSTOR restores the processor's x87 FPU and SSE states from an FXSAVE area. The XSAVE features set supports x87 FPU and SSE states using the same layout as the FXSAVE area to provide interoperability of FXSAVE versus XSAVE, and FXRSTOR versus XRSTOR. The XSAVE feature set allows system software to manage SSE state independent of x87 FPU states. Thus system software that had been using FXSAVE and FXRSTOR to manage x87 FPU and SSE states can transition to using the XSAVE feature set to manage x87 FPU, SSE and other processor extended states in a systematic and forward-looking manner. See Section 10.5 and Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* for more details.

System software can implement forward-looking processor extended state management using the XSAVE feature set. In this case, system software must specify the bit vector mask in EDX:EAX appropriately when executing XSAVE/XRSTOR instructions.

For instance, the OS can supply instructions in the XSAVE feature set with a bit vector in EDX:EAX with the two least significant bits (corresponding to x87 FPU and SSE state) equal to 0. Then, the XSAVE instruction will not write the processor's x87 FPU and SSE state into memory. Similarly, the XRSTOR instruction executed with a value in EDX:EAX with the least two significant bit equal to 0 will not restore nor initialize the processor's x87 FPU and SSE state.

The processor's action as a result of executing XRSTOR is given in Section 13.7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*. The instruction may be used to initialized x87 FPU or XMM registers. When the MXCSR register is updated from memory, reserved bit checking is enforced. The saving/restoring of MXCSR is bound to the SSE state, independent of the x87 FPU state. The action of XSAVE is given in Section 13.6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Instructions in the XSAVE feature set cause a #NM (Device Not Available) exceptions if CR0.TS is set. Thus, system software can implement the “lazy save and restore” technique of managing x87 FPU and SSE state using either FXSAVE and FXRSTOR or the XSAVE feature set.

13.8 INTEL ADVANCED VECTOR EXTENSIONS (INTEL AVX) AND YMM STATE

Intel AVX instructions comprises of 256-bit and 128-bit instructions that operates on 256-bit YMM registers. The XSAVE feature set allows software to save and restore the state of these registers. See Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.

following sections describes system software support requirements for 256-bit YMM states.

For processors that support YMM states, the YMM state exists in all operating modes. However, the available instruction interfaces to access YMM states may vary in different modes. The XSAVE feature set is available in all operating modes.

13.9 YMM STATE MANAGEMENT

Operating systems must use the XSAVE feature set for YMM state management. The XSAVE feature set also provides flexible and efficient interface to manage XMM/MXCSR states and x87 FPU states in conjunction with newer processor extended states like YMM states.

An operating system must enable its YMM state management to support AVX and any 256-bit extensions that operate on YMM registers. Otherwise, an attempt to execute an instruction in AVX extensions (including an enhanced 128-bit SIMD instructions using VEX encoding) will cause a #UD exception. See Section 13.3 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* for more details.

Detection of hardware support for new processor extended state is provided by the CPUID instruction. See Section 13.2 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* for more details.

13.9.1 Enabling of YMM State

An operating system can enable YMM state support with the following steps:

- Verify the processor supports the XSAVE feature set by checking CPUID.1.ECX.XSAVE[bit 26]=1.
- Verify the processor supports YMM state by checking CPUID.(EAX=0DH, ECX=0):EAX.YMM[2]. The operating system should also verify CPUID.(EAX=0DH, ECX=0):EAX.SSE[bit 1]=1, because the lower 128-bits of each YMM register are aliased to an XMM register.

  The operating system must determine the buffer size requirement for the XSAVE area that will be used by XSAVE/XRSTOR (see Section 13.2 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).
- Set CR4.OSXSAVE[bit 18]=1 to enable the XSAVE feature set.
- Supply an appropriate mask via EDX:EAX to execute XSETBV with ECX = 0 to set XCR0 to enable the processor state components that the operating system desires to manage using the XSAVE feature set. To enable x87 FPU, SSE and YMM state management by the XSAVE feature set, the enable mask is EDX=0H, EAX=7H (the individual bits of XCR0 are specified in Section 13.3 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).

  To enable YMM state, the operating system must ensure that EAX[2:1] = 11B when executing XSETBV. An attempt to execute XSETBV with EDX:EAX[2:1] = 10B causes a general-protection exception (#GP).

13.9.2 Enabling of SIMD Floating-Exception Support

AVX instructions may generate SIMD floating-point exceptions. An OS must enable SIMD floating-point exception support by setting CR4.OSXMMEXCPT[bit 10]=1.

The effect of CR4 setting that affects AVX enabling is listed in Table 13-3.

Table 13-3 CR4 bits for AVX New Instructions technology support

Bit	Meaning
CR4.OSXSAVE[bit 18]	If set, the OS supports use of the XSAVE feature set to manage processor extended state. Must be set to '1' to enable AVX.
CR4.OSXMMEXCPT[bit 10]	Must be set to 1 to enable SIMD floating-point exceptions. This applies to AVX operating on YMM states, and legacy 128-bit SIMD floating-point instructions operating on XMM states.
CR4.OSFXSR[bit 9]	Ignored by AVX instructions operating on YMM states. Must be set to 1 to enable SIMD instructions operating on XMM state.

The operation of XSAVE, XRSTOR, and XSAVEOPT is detailed in Section 13.6 through Section 13.8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

...

17. Updates to Chapter 14, Volume 3B

Change bars show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

14.3.4 Performance and Energy Bias Hint support

Intel 64 processors may support additional software hint to guide the hardware heuristic of power management features to favor increasing dynamic performance or conserve energy consumption.

Software can detect processor's capability to support performance-energy bias preference hint by examining bit 3 of ECX in CPUID leaf 6. The processor supports this capability if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H).

Software can program the lowest four bits of IA32_ENERGY_PERF_BIAS MSR with a value from 0 - 15. The values represent a sliding scale, where a value of 0 (the default reset value) corresponds to a hint preference for highest performance and a value of 15 corresponds to the maximum energy savings. A value of 7 roughly translates into a hint to balance performance with energy consumption.

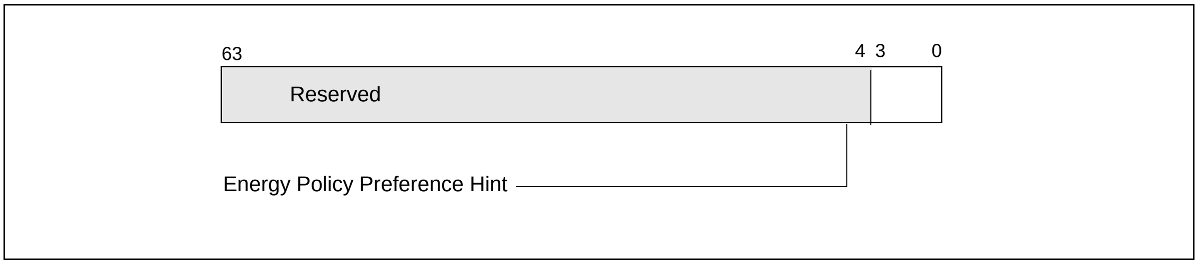


Figure 14-4 IA32_ENERGY_PERF_BIAS Register

The layout of IA32_ENERGY_PERF_BIAS is shown in Figure 14-4. The scope of IA32_ENERGY_PERF_BIAS is per logical processor, which means that each of the logical processors in the package can be programmed with a different value. This may be especially important in virtualization scenarios, where the performance / energy requirements of one logical processor may differ from the other. Conflicting "hints" from various logical processors at higher hierarchy level will be resolved in favor of performance over energy savings.

Software can use whatever criteria it sees fit to program the MSR with the appropriate value. However, the value only serves as a hint to the hardware and the actual impact on performance and energy savings is model specific.

...

18. 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.

...

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 45nm Intel 64 processor on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-M" 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 35, “Model-Specific Registers (MSRs),” 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.

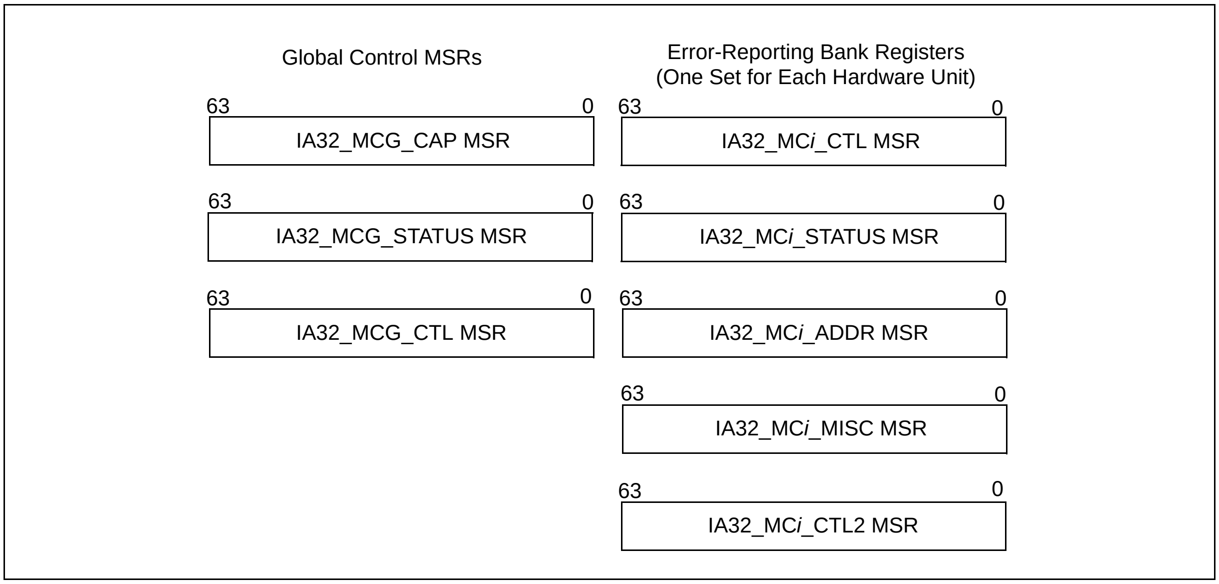


Figure 15-1 Machine-Check MSRs

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.

...

15.3.1.1 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 structure of the register in Pentium 4, Intel Xeon, Intel Atom, and P6 family processors.

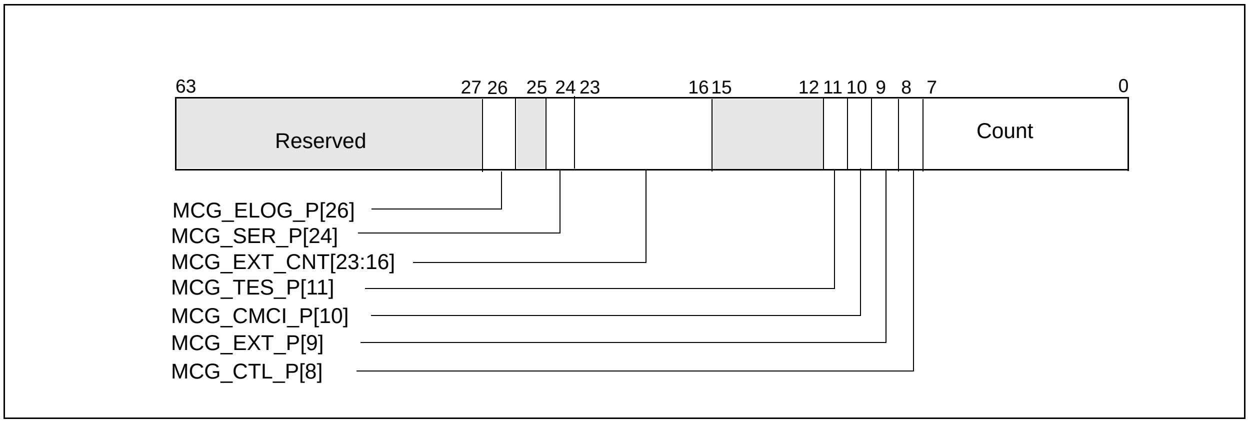


Figure 15-2 IA32_MCG_CAP Register

...

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 35, “Model-Specific Registers (MSRs),” 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.

...

15.3.2.4 IA32_MCi_MISC MSRs

The IA32_MC*i*_MISC MSR contains additional information describing the machine-check error if the MISCV flag in the IA32_MC*i*_STATUS register is set. The IA32_MCi_MISC_MSR is either not implemented or does not contain additional information if the MISCV 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 MISCV and IA32_MCG_CAP[24] are set, the IA32_MCi_MISC_MSR is defined according to Figure 15-7 to support software recovery of uncorrected errors (see Section 15.6):

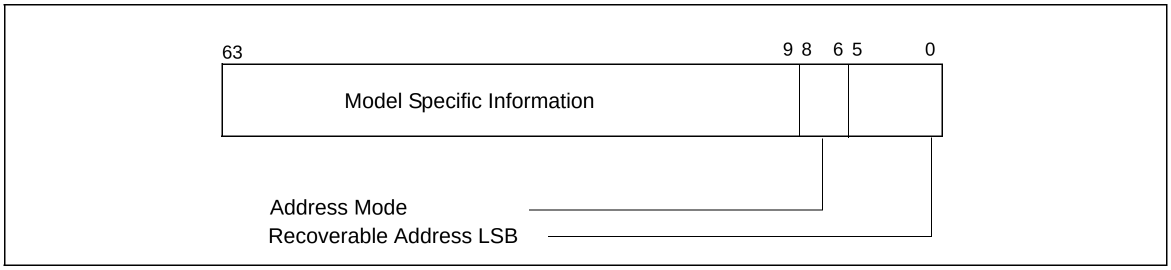


Figure 15-7 UCR Support in IA32_MCi_MISC Register

...

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_MC*i*_STATUS and IA32_MC*i*_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_MC*i*_STATUS and IA32_MC*i*_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.

...

15.8 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_MC*i*_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.

...

15.9.2.1 Correction Report Filtering (F) Bit

Starting with Intel Core Duo processors, bit 12 in the “Form” column in Table 15-9 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.

...

15.9.2.3 Level (LL) Sub-Field

The 2-bit LL sub-field (see Table 15-11) 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 15-11 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

...

15.9.5 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.

...

15.10.1 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 recovery from a machine-check error may be possible, 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_MC*i*_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_MC*i*_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.
- 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.
- Correctable errors are corrected automatically by the processor. The UC flag in each IA32_MC*i*_STATUS register indicates whether the processor automatically corrected an error.
- 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.

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;
        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-M” 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.
- 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 uncorrected errors (UC=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. See Chapter 19: Table 19-15 to find the errors that do not generate machine check exceptions.
- 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.
- When 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 cannot take recovery action as the information of the SRAO error in the IA32_MCi_STATUS register was potentially lost due to the overflow condition. 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.

...

19. Updates to Chapter 16, Volume 3B

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

...

16.4.3 Integrated Memory Controller Machine Check Errors

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

...

16.5 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY WITH CPUID DISPLAYFAMILY_DISPLAYMODEL SIGNATURE 06_3EH, MACHINE ERROR CODES FOR MACHINE CHECK

Intel Xeon processor E5-2600 v2 product family is based on Intel® microarchitecture code name Ivy Bridge-EP and can be identified with CPUID DisplayFamily_DisplaySignature 06_3EH. Incremental error codes for internal machine check error from PCU controller is reported in the register bank IA32_MC4, Table 16-17 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS. Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC5. Information listed in Table 16-14 for QPI MC error code apply to IA32_MC5_STATUS. Incremental error codes for the memory controller unit is reported in the register banks IA32_MC9-IA32_MC16. Table 16-18 lists model-specific error codes apply to IA32_MCi_STATUS, i = 9-16.

...

16.5.2 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC9_STATUS-IA32_MC16_STATUS. The supported error codes are follows the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture,").

MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in IA32_MCi_STATUS and IA32_MCi_MISC; (i = 9, 16).

Table 16-18 Intel IMC MC Error Codes for IA32-MCi_STATUS (i= 9, 16)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	0-15	MCACOD	Bus error format: 1PPTRRRRIILL
Model specific errors	31:16	Reserved except for the following	0x001 - Address parity error 0x002 - HA Wrt buffer Data parity error 0x004 - HA Wrt byte enable parity error 0x008 - Corrected patrol scrub error
			0x010 - Uncorrected patrol scrub error 0x020 - Corrected spare error 0x040 - Uncorrected spare error 0x080 - Corrected memory read error. (Only applicable with iMC's "Additional Error logging" Mode-1 enabled.) 0x100 - iMC, WDB, parity errors
	36-32	Other info	When MSR_ERROR_CONTROL.[1] is set, logs an encoded value from the first error device.
	37	Reserved	Reserved
	56-38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	57-63		

NOTES:

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

...

16.6 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY 0FH MACHINE ERROR CODES FOR MACHINE CHECK

Table 16-20 provides information for interpreting additional family 0FH model-specific fields for external bus errors. These errors are reported in the IA32_MCi_STATUS MSRs. They are reported architecturally) as compound errors with a general form of *0000 1PPT RRRR IILL* in the MCA error code field. See Chapter 15 for information on the interpretation of compound error codes.

...

20. Updates to Chapter 17, Volume 3B

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

...

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

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

The LBR facility also supports an alternate capability to profile call stack profiles. Configuring the LBR facility to conduct call stack profiling is by writing 1 to the MSR_LBR_SELECT.EN_CALLSTACK[bit 9]; see Table 17-12. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 17.7.

Table 17-12 MSR_LBR_SELECT for Intel microarchitecture code name Haswell

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches occurring in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches occurring in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
EN_CALLSTACK[1]	9		Enable LBR stack to use LIFO filtering to capture Call stack profile
Reserved	63:10		Must be zero

NOTES:

1. Must set valid combination of bits 0-8 in conjunction with bit 9, otherwise the counter result is undefined.

The call stack profiling capability is an enhancement of the LBR facility. The LBR stack is a ring buffer typically used to profile control flow transitions resulting from branches. However, the finite depth of the LBR stack often become less effective when profiling certain high-level languages (e.g. C++), where a transition of the execution flow is accompanied by a large number of leaf function calls, each of which returns an individual parameter to form the list of parameters for the main execution function call. A long list of such parameters returned by the leaf functions would serve to flush the data captured in the LBR stack, often losing the main execution context.

When the call stack feature is enabled, the LBR stack will capture unfiltered call data normally, but as return instructions are executed the last captured branch record is flushed from the on-chip registers in a last-in first-out (LIFO) manner. Thus, branch information relative to leaf functions will not be captured, while preserving the call stack information of the main line execution path.

The configuration of the call stack facility is summarized below:

- Set IA32_DEBUGCTL.LBR (bit 0) to enable the LBR stack to capture branch records. The source and target addresses of the call branches will be captured in the 16 pairs of From/To LBR MSRs that form the LBR stack.
- Program the Top of Stack (TOS) MSR that points to the last valid from/to pair. This register is incremented by 1, modulo 16, before recording the next pair of addresses.
- Program the branch filtering bits of MSR_LBR_SELECT (bits 0:8) as desired.
- Program the MSR_LBR_SELECT to enable LIFO filtering of return instructions with:
  - — The following bits in MSR_LBR_SELECT must be set to ‘1’: JCC, NEAR_IND_JMP, NEAR_REL_JMP, FAR_BRANCH, EN_CALLSTACK;
  - — The following bits in MSR_LBR_SELECT must be cleared: NEAR_REL_CALL, NEAR-IND_CALL, NEAR_RET;
  - — At most one of CPL_EQ_0, CPL_NEQ_0 is set.

Note that when call stack profiling is enabled, "zero length calls" are excluded from writing into the LBRs. (A "zero length call" uses the attribute of the call instruction to push the immediate instruction pointer on to the stack and then pops off that address into a register. This is accomplished without any matching return on the call.)

...

21. Updates to Chapter 18, Volume 3B

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

...

18.4.2 Global Counter Control Facilities

Processors based on Intel Core microarchitecture provides simplified performance counter control that simplifies the most frequent operations in programming performance events, i.e. enabling/disabling event counting and checking the status of counter overflows. This is done by the following three MSRs:

- MSR_PERF_GLOBAL_CTRL enables/disables event counting for all or any combination of fixed-function PMCs (MSR_PERF_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.
- MSR_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs (MSR_PERF_FIXED_CTRx) or general-purpose PMCs via a single RDMSR.
- MSR_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs (MSR_PERF_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.

MSR_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting in each performance counter (see Figure 18-11). Each enable bit in MSR_PERF_GLOBAL_CTRL is AND’ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or MSR_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND’ed results is true; counting is disabled when the result is false.

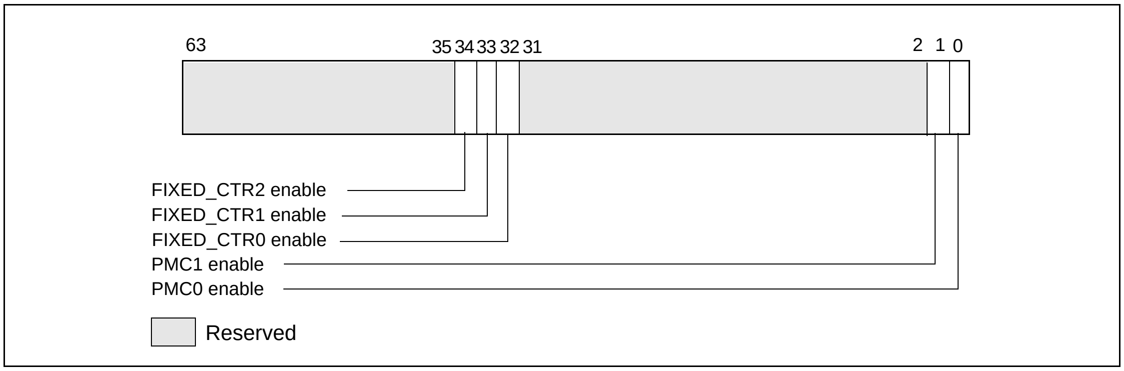


Figure 18-11 Layout of MSR_PERF_GLOBAL_CTRL MSR

MSR_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. The MSR also provides additional status bit to indicate overflow conditions when counters are programmed for precise-event-based sampling (PEBS). The MSR_PERF_GLOBAL_STATUS MSR also provides a 'sticky bit' to indicate changes to the state of performance monitoring hardware (see Figure 18-12). A value of 1 in bits 34:32, 1, 0 indicates an overflow condition has occurred in the associated counter.

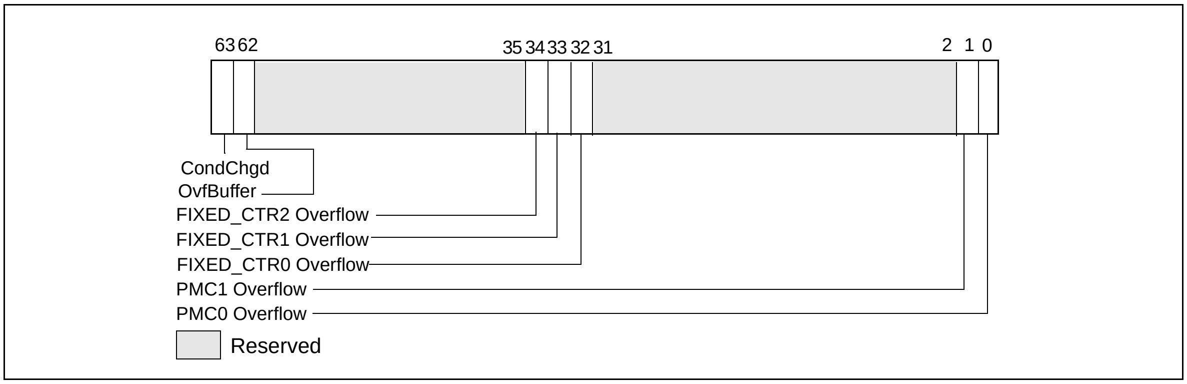


Figure 18-12 Layout of MSR_PERF_GLOBAL_STATUS MSR

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 17.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

MSR_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 18-13). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or sampling
- Reloading counter values to continue sampling
- Disabling event counting or sampling

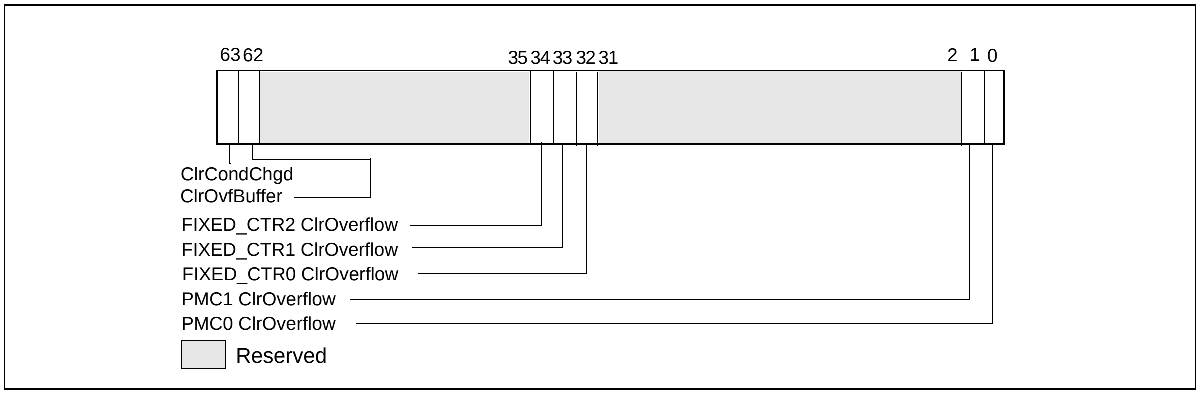


Figure 18-13 Layout of MSR_PERF_GLOBAL_OVF_CTRL MSR

...

18.6 PERFORMANCE MONITORING (PROCESSORS BASED ON THE SILVERMONT MICROARCHITECTURE)

Intel processors based on the Silvermont microarchitecture support architectural performance monitoring capability with version ID 3 (see Section 18.2.2.2) and a host of non-architectural monitoring capabilities. Processors based on the Silvermont microarchitecture provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

Non-architectural performance monitoring in the Silvermont microarchitecture uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events is listed in Table 19-18.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 18-6 and described in Section 18.2.1.1 and Section 18.2.2.2. Architectural and non-architectural performance monitoring events in the Silvermont microarchitecture ignore the AnyThread qualification regardless of its setting in IA32_PERFEVTSELx MSR.

18.6.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- The width of counter reported by CPUID.0AH:EAX[23:16] is 40 bits.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests.

18.6.1.1 Precise Event Based Sampling (PEBS)

Processors based on the Silvermont microarchitecture support precise event based sampling (PEBS). PEBS is supported using IA32_PMC0 (see also Section 17.4.9, “BTS and DS Save Area”).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 18.4.4).

The list of PEBS events supported in the Silvermont microarchitecture is shown in Table 18-12.

Table 18-12 PEBS Performance Events for the Silvermont Microarchitecture

Event Name	Event Select	Sub-event	UMask
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		FAR_BRANCH	BFH
		RETURN	F7H
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H
		L2_MISS_LOADS	04H
		DLTB_MISS_LOADS	08H
		HITM	20H
REHABQ	03H	LD_BLOCK_ST_FORWARD	01H
		LD_SPLITS	08H

PEBS Record Format The PEBS record format supported by processors based on the Intel Silvermont microarchitecture is shown in Table 18-13, and each field in the PEBS record is 64 bits long.

Table 18-13 PEBS Record Format for the Silvermont Microarchitecture

Byte Offset	Field	Byte Offset	Field
0x0	R/EFLAGS	0x60	R10
0x8	R/EIP	0x68	R11
0x10	R/EAX	0x70	R12
0x18	R/EBX	0x78	R13

Table 18-13 PEBS Record Format for the Silvermont Microarchitecture

Byte Offset	Field	Byte Offset	Field
0x20	R/ECX	0x80	R14
0x28	R/EDX	0x88	R15
0x30	R/ESI	0x90	IA32_PERF_GLOBAL_STATUS
0x38	R/EDI	0x98	Reserved
0x40	R/EBP	0xA0	Reserved
0x48	R/ESP	0xA8	Reserved
0x50	R8	0x80	EventingRIP
0x58	R9	0xB8	Reserved

18.6.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 0x1A6) in conjunction with umask value 01H or MSR_OFFCORE_RSP1 (address 0x1A7) in conjunction with umask value 02H. Table 19-18 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 18-14 OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-3	0xB7	0x01	MSR_OFFCORE_RSP0 (address 0x1A6)
PMC0-3	0xB7	0x02	MSR_OFFCORE_RSP1 (address 0x1A7)

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are shown in Figure 18-32 and Figure 18-33. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 18.6.3 for details.

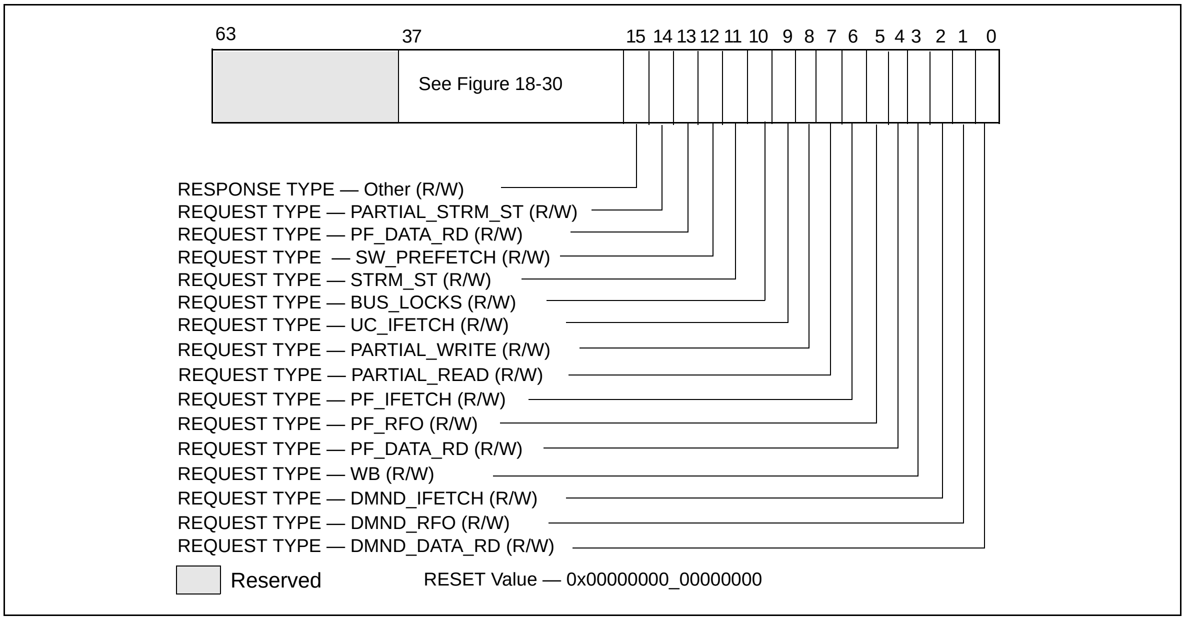


Figure 18-14 Request_Type Fields for MSR_OFFCORE_RSPx

Table 18-15 MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	(R/W). Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	(R/W). Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	(R/W). Counts the number of demand and DCU prefetch instruction cacheline reads. Does not count L2 code read prefetches.
WB	3	(R/W). Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	(R/W). Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	(R/W). Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	(R/W). Counts the number of code reads generated by L2 prefetchers.
PARTIAL_READ	7	(R/W). Counts the number of demand reads of partial cache lines (including UC and WC).
PARTIAL_WRITE	8	(R/W). Counts the number of demand RFO requests to write to partial cache lines (includes UC, WT and WP)
UC_IFETCH	9	(R/W). Counts the number of UC instruction fetches.
BUS_LOCKS	10	(R/W). Bus lock and split lock requests
STRM_ST	11	(R/W). Streaming store requests
SW_PREFETCH	12	(R/W). Counts software prefetch requests
PF_DATA_RD	13	(R/W). Counts DCU hardware prefetcher data read requests

Table 18-15 MSR_OFFCORE_RSPx Request_Type Field Definition (Contd.)

Bit Name	Offset	Description
PARTIAL_STRM_ST	14	(R/W). Streaming store requests
OTHER	15	(R/W). Any other request that crosses IDI, including I/O.

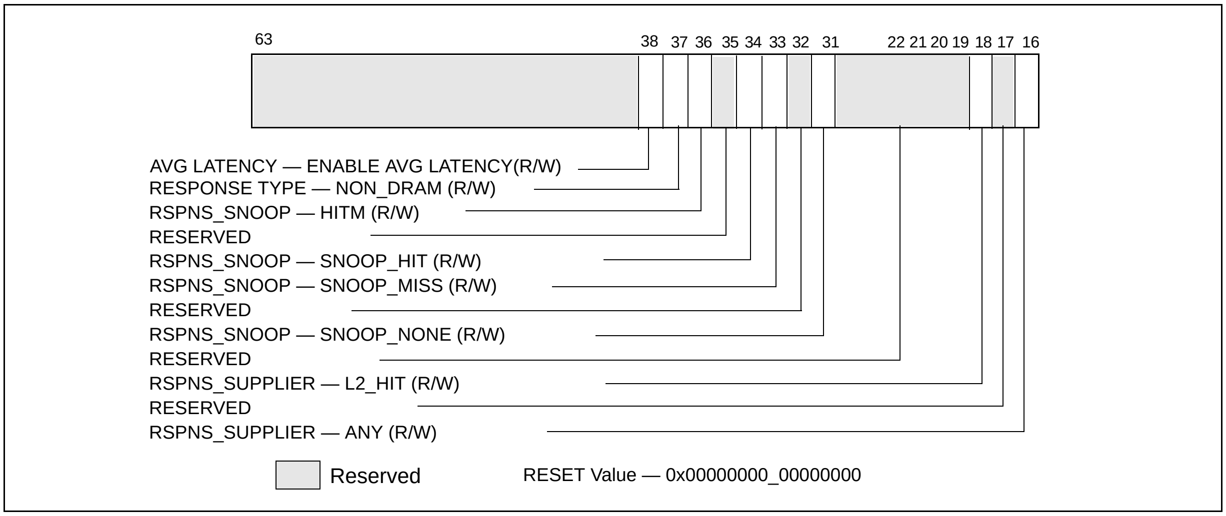


Figure 18-15 Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSPx

To properly program this extra register, software must set at least one request type bit and a valid response type pattern. Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 18-16 MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	(R/W). Catch all value for any response types.
Supplier Info	Reserved	17	Reserved
	L2_HIT	18	(R/W). Cache reference hit L2 in either M/E/S states.
	Reserved	30:19	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY“ bit is set, the supplier and snoop info bits are ignored.

Table 18-17 MSR_OFFCORE_RSPx Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	(R/W). No details on snoop-related information
	Reserved	32	Reserved
	SNOOP_MISS	33	(R/W). Counts the number of snoop misses when L2 misses
	SNOOP_HIT	34	(R/W). Counts the number of snoops hit in the other module where no modified copies were found
	Reserved	35	Reserved
	HITM	36	(R/W). Counts the number of snoops hit in the other module where modified copies were found in other core's L1 cache.
	NON_DRAM	37	(R/W). Target was non-DRAM system address. This includes MMIO transactions.
	AVG_LATENCY	38	(R/W). Enable average latency measurement by counting weighted cycles of outstanding offcore requests of the request type specified in bits 15:0 and any response (bits 37:16 cleared to 0). This bit is available in MSR_OFFCORE_RESP0. The weighted cycles is accumulated in the specified programmable counter IA32_PMCx and the occurrence of specified requests are counted in the other programmable counter.

18.6.3 Average Offcore Request Latency Measurement

Measurement of average latency of offcore transaction requests can be enabled using MSR_OFFCORE_RSP0.[bit 38] with the choice of request type specified in MSR_OFFCORE_RSP0.[bit 15:0] and MSR_OFFCORE_RSP0.[bit 37:16] set to 0.

When average latency measurement is enabled, e.g. with IA32_PERFEVTSEL0.[bits 15:0] = 0x01B7 and chosen value of MSR_OFFCORE_RSP0, IA32_PMC0 will accumulate weighted cycles of outstanding transaction requests for the specified transaction request type. At the same time, IA32_PMC1 will accumulated the number of occurrences each time a new transaction request of specified type is made.

...

18.9.1 Global Counter Control Facilities In Intel® Microarchitecture Code Name Sandy Bridge

The number of general-purpose performance counters visible to a logical processor can vary across Processors based on Intel microarchitecture code name Sandy Bridge. Software must use CPUID to determine the number performance counters/event select registers (See Section 18.2.1.1).

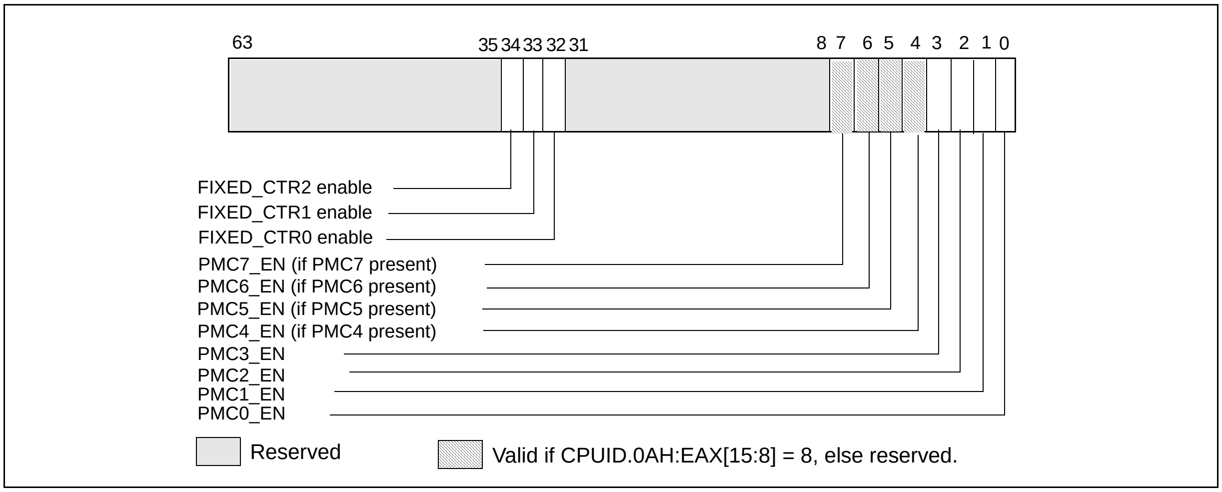


Figure 18-28 IA32_PERF_GLOBAL_CTRL MSR in Intel® Microarchitecture Code Name Sandy Bridge

Figure 18-11 depicts the layout of IA32_PERF_GLOBAL_CTRL MSR. The enable bits (PMC4_EN, PMC5_EN, PMC6_EN, PMC7_EN) corresponding to IA32_PMC4-IA32_PMC7 are valid only if CPUID.0AH:EAX[15:8] reports a value of ‘8’. If CPUID.0AH:EAX[15:8] = 4, attempts to set the invalid bits will cause #GP.

Each enable bit in IA32_PERF_GLOBAL_CTRL is AND’ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND’ed results is true; counting is disabled when the result is false.
IA32_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. The MSR also provides additional status bit to indicate overflow conditions when counters are programmed for precise-event-based sampling (PEBS). The IA32_PERF_GLOBAL_STATUS MSR also provides a ‘sticky bit’ to indicate changes to the state of performance monitoring hardware (see Figure 18-29). A value of 1 in each bit of the PMCx_OVF field indicates an overflow condition has occurred in the associated counter.

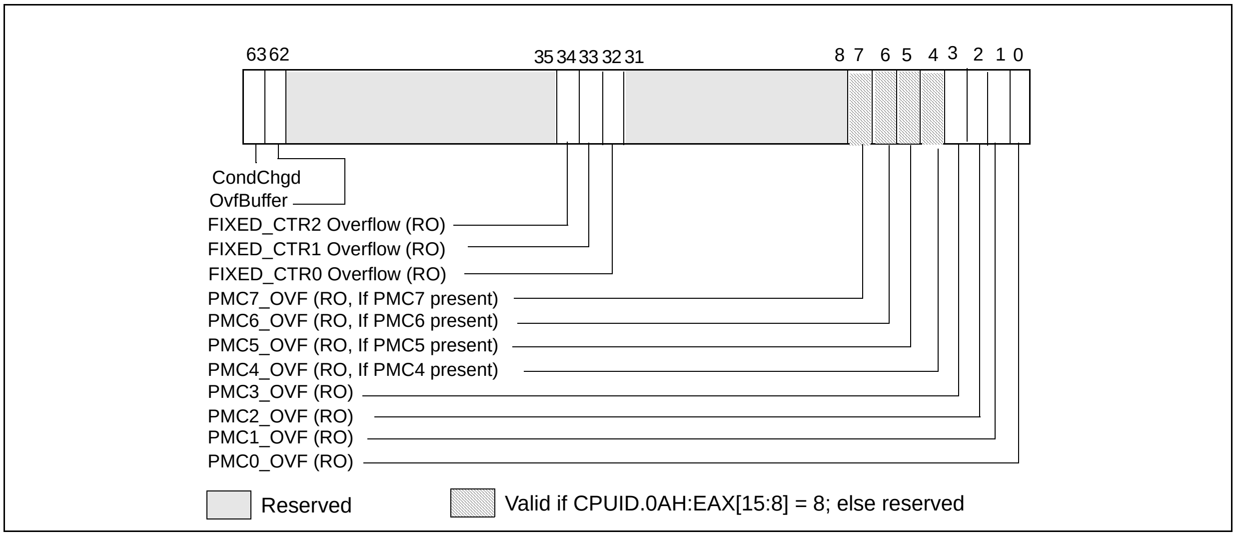


Figure 18-29 IA32_PERF_GLOBAL_STATUS MSR in Intel® Microarchitecture Code Name Sandy Bridge

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 17.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 18-30). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or sampling
- Reloading counter values to continue sampling
- Disabling event counting or sampling

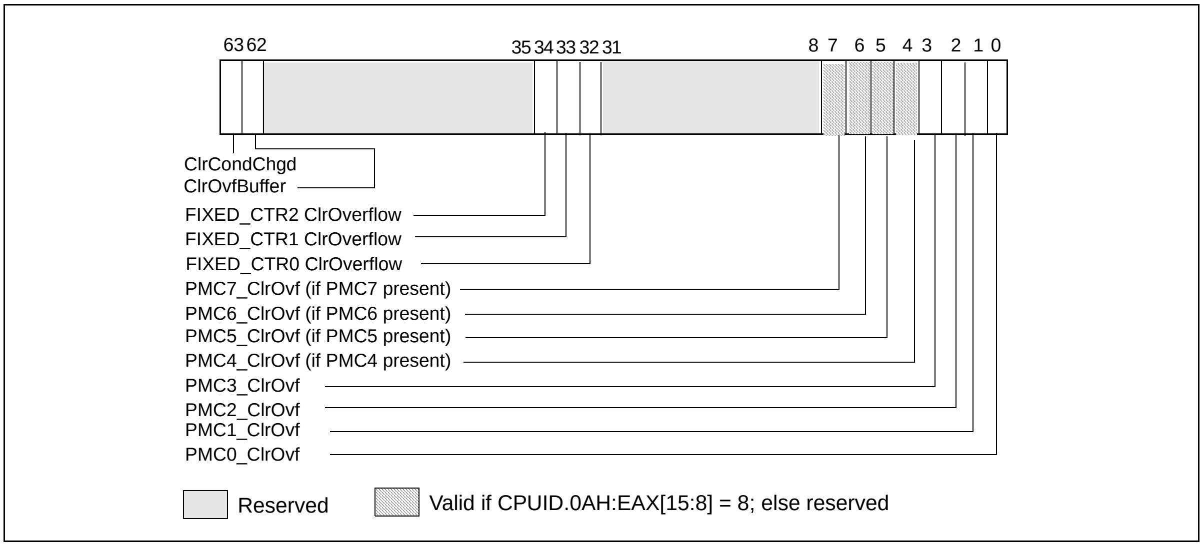


Figure 18-30 IA32_PERF_GLOBAL_OVF_CTRL MSR in Intel microarchitecture code name Sandy Bridge

...

18.9.8 Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5-2600 product family based on Intel microarchitecture Sandy Bridge has some similarities with those of the Intel Xeon processor E7 family based on Intel microarchitecture Sandy Bridge. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope. For example, each Cbox caching agent has a set of local performance counters, and the power controller unit (PCU) has its own local performance counters. Up to 8 C-Box units are supported in the uncore subsystem.

Table 18-36 summarizes the uncore PMU facilities providing MSR interfaces.

Table 18-36 Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	4	44	Yes	per-box	None
PCU	1	4	48	Yes	per-box	Match/Mask
U-Box	1	2	44	Yes	uncore	None

...

22. Updates to Chapter 31, Volume 3C

Change bars show changes to Chapter 31 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

...

31.9.3 IA-32e Mode Hosts

An IA-32e mode host is required to support 64-bit guest environments. Because activating IA-32e mode currently requires that paging be disabled temporarily and VMX entry requires paging to be enabled, IA-32e mode must be enabled before entering VMX operation. For this reason, it is not possible to toggle in and out of IA-32e mode in a VMM.

Section 31.5 describes the steps required to launch a VMM. An IA-32e mode host is also required to set the "host address-space size" VMCS VM-exit control to 1. The value of this control is then loaded in the IA32_EFER.LME/LMA and CS.L bits on each VM exit. This establishes a 64-bit host environment as execution transfers to the VMM entry point. At a minimum, the entry point is required to be in a 64-bit code segment. Subsequently, the VMM can, if it chooses, switch to 32-bit compatibility mode on a code-segment basis (see Section 31.9.1). Note, however, that VMX instructions other than VMCALL and VMFUNC are not supported in compatibility mode; they generate an invalid opcode exception if used.

The following VMCS controls determine the value of IA32_EFER when a VM exit occurs: the "host address-space size" control (described above), the "load IA32_EFER" VM-exit control, the "VM-exit MSR-load count," and the "VM-exit MSR-load address" (see Section 27.3).

If the "load IA32_EFER" VM-exit control is 1, the value of the LME and LMA bits in the IA32_EFER field in the host-state area must be the value of the "host address-space size" VM-exit control.

The loading of IA32_EFER.LME/LMA and CS.L bits established by the "host address-space size" control precede any loading of the IA32_EFER MSR due from the VM-exit MSR-load area. If IA32_EFER is specified in the VM-exit MSR-load area, the value of the LME bit in the load image of IA32_EFER should match the setting of the "host address-space size" control. Otherwise the attempt to modify the LME bit (while paging is enabled) will lead to a VMX-abort. However, IA32_EFER.LMA is always set by the processor to equal IA32_EFER.LME & CR0.PG; the value specified for LMA in the load image of the IA32_EFER MSR is ignored. For these and performance reasons, VMM writers may choose to not use the VM-exit/entry MSR-load/save areas for IA32_EFER.

On a VMM teardown, VMX operation should be exited before deactivating IA-32e mode if the latter is required.

...

23. 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.

...

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.

...

24. Updates to Chapter 35, Volume 3C

Change bars show changes to Chapter 35 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

This chapter list MSRs across Intel processor families. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions.

...

Table 35-1 CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_3DH	Next Generation Intel Core Processor
06_3FH	Future Generation Intel Xeon Processor
06_3CH, 06_45H, 06_46H	4th Generation Intel Core Processor and Intel Xeon Processor E3-1200 v3 Product Family based on Intel® microarchitecture code name Haswell
06_3EH	Next Generation Intel Xeon Processor E7 Family based on Intel® microarchitecture code name Ivy Bridge-EP
06_3EH	Intel Xeon Processor E5-1600 v2/E5-2600 v2 Product Families based on Intel® microarchitecture code name Ivy Bridge-EP, 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 Intel® microarchitecture code name Ivy Bridge
06_2DH	Intel Xeon Processor E5 Family based on Intel microarchitecture code name Sandy Bridge, Intel Core i7-39xx Processor Extreme Edition

Table 35-1 CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_2FH	Intel Xeon Processor E7 Family
06_2AH	Intel Xeon Processor E3-1200 Product Family; 2nd Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series
06_25H, 06_2CH	Intel Xeon processors 3600, 5600 series, Intel Core i7, i5 and i3 Processors
06_1EH, 06_1FH	Intel Core i7 and i5 Processors
06_1AH	Intel Core i7 Processor, Intel Xeon Processor 3400, 3500, 5500 series
06_1DH	Intel Xeon Processor MP 7400 series
06_17H	Intel Xeon Processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_37H, 06_4DH	Intel Atom Processor C2000, E3000 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 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

...

Table 35-2 IA-32 Architectural MSRs

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
0H	0	IA32_P5_MC_ADDR (P5_MC_ADDR)	See Section 35.16, “MSRs in Pentium Processors.”	**Pentium Processor (05_01H)**

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	**Decimal**			
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Section 35.16, “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 17.13, “Time-Stamp Counter.”	05_01H
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	**Platform ID (RO)** The operating system can use this MSR to determine “slot” information for the processor and the proper microcode update to load.	06_01H
		49:0	Reserved.	
		52:50	**Platform Id (RO)** Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7	
		63:53	Reserved.	
1BH	27	IA32_APIC_BASE (APIC_BASE)		06_01H
		7:0	Reserved	
		8	BSP flag (R/W)	
		9	Reserved	
		10	Enable x2APIC mode	06_1AH
		11	APIC Global Enable (R/W)	
		(MAXPHYWID - 1):12	APIC Base (R/W)	
		63: MAXPHYWID	Reserved	
3AH	58	IA32_FEATURE_CONTROL	**Control Features in Intel 64 Processor (R/W)**	If CPUID.01H: ECX[bit 5 or bit 6] = 1

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written, writes to this bit will result in GP(0). Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support	If CPUID.01H:ECX[bit 5 or bit 6] = 1
			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_MSR contents are preserved across RESET when PWRGOOD is not deasserted.	
		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[bit 5 and bit 6] are set to 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[bit 5 or bit 6] = 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[bit 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[bit 6] = 1
		63:16	Reserved	
3BH	59	IA32_TSC_ADJUST	Per Logical Processor TSC Adjust (R/Write to clear)	If CPUID.(EAX=07H, ECX=0H): EBX[1] = 1

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		63:0	**THREAD_ADJUST:** Local offset value of the IA32_TSC for a logical processor. Reset value is Zero. A write to IA32_TSC will modify the local offset in IA32_TSC_ADJUST and the content of IA32_TSC, but does not affect the internal invariant TSC hardware.	
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.	
9BH	155	IA32_SMM_MONITOR_CTL	SMM Monitor Configuration (R/W)	If CPUID.01H: ECX[bit 5 or bit 6] = 1
		0	Valid (R/W)	
		1	Reserved	
		2	Controls SMI unblocking by VMXOFF (see Section 34.14.4)	If IA32_VMX_MISC[bit 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[bit 15])
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
C2H	194	IA32_PMC1 (PERFCTR1)	General Performance Counter 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
C3H	195	IA32_PMC2	General Performance Counter 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
C4H	196	IA32_PMC3	General Performance Counter 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
C5H	197	IA32_PMC4	General Performance Counter 4 (R/W)	If CPUID.0AH: EAX[15:8] > 4
C6H	198	IA32_PMC5	General Performance Counter 5 (R/W)	If CPUID.0AH: EAX[15:8] > 5
C7H	199	IA32_PMC6	General Performance Counter 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
C8H	200	IA32_PMC7	General Performance Counter 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7
E7H	231	IA32_MPERF	Maximum Qualified Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	**C0_MCNT: C0 Maximum Frequency Clock Count** Increments at fixed interval (relative to TSC freq.) when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_APERF.	
E8H	232	IA32_APERF	Actual Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	**C0_ACNT: C0 Actual Frequency Clock Count** Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_MPERF.	
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (RO) 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.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		63:12	Reserved.	
174H	372	IA32_SYSENTER_CS	SYSENTER_CS_MSR (R/W)	06_01H
		15:0	CS Selector	
		63:16	Reserved.	
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_1AH
		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
		63:27	Reserved.	
17AH	378	IA32_MCG_STATUS (MCG_STATUS)	Global Machine Check Status (RO)	06_01H
17BH	379	IA32_MCG_CTL (MCG_CTL)	Global Machine Check Control (R/W)	06_01H
180H-185H	384-389	Reserved		06_0EH[1]
186H	390	IA32_PERFEVTSEL0 (PERFEVTSEL0)	Performance Event Select Register 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
		7:0	Event Select: Selects a performance event logic unit.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		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[2]
198H	408	IA32_PERF_STATUS	(RO)	0F_03H
		15:0	Current performance State Value	
		63:16	Reserved.	
199H	409	IA32_PERF_CTL	(R/W)	0F_03H
		15:0	Target performance State Value	
		31:16	Reserved.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		32	IDA Engage. (R/W) When set to 1: disengages IDA	06_0FH (Mobile)
		63:33	Reserved.	
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation Control (R/W) See Section 14.5.3, "Software Controlled Clock Modulation."	0F_0H
		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.	
		4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	
		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 14.5.2, "Thermal Monitor."	0F_0H
		0	High-Temperature Interrupt Enable	
		1	Low-Temperature Interrupt Enable	
		2	PROCHOT# Interrupt Enable	
		3	FORCEPR# Interrupt Enable	
		4	Critical Temperature Interrupt Enable	
		7:5	Reserved.	
		14:8	Threshold #1 Value	
		15	Threshold #1 Interrupt Enable	
		22:16	Threshold #2 Value	
		23	Threshold #2 Interrupt Enable	
		24	Power Limit Notification Enable	If CPUID.06H:EAX[4] = 1
		63:25	Reserved.	
19CH	412	IA32_THERM_STATUS	Thermal Status Information (RO) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 14.5.2, "Thermal Monitor"	0F_0H

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		0	Thermal Status (RO):	
		1	Thermal Status Log (R/W):	
		2	PROCHOT # or FORCEPR# event (RO)	
		3	PROCHOT # or FORCEPR# log (R/WC0)	
		4	Critical Temperature Status (RO)	
		5	Critical Temperature Status log (R/WC0)	
		6	Thermal Threshold #1 Status (RO)	If CPUID.01H:ECX[8] = 1
		7	Thermal Threshold #1 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		8	Thermal Threshold #2 Status (RO)	If CPUID.01H:ECX[8] = 1
		9	Thermal Threshold #1 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		10	Power Limitation Status (RO)	If CPUID.06H:EAX[4] = 1
		11	Power Limitation log (R/WC0)	If CPUID.06H:EAX[4] = 1
		15:12	Reserved.	
		22:16	Digital Readout (RO)	If CPUID.06H:EAX[0] = 1
		26:23	Reserved.	
		30:27	Resolution in Degrees Celsius (RO)	If CPUID.06H:EAX[0] = 1
		31	Reading Valid (RO)	If CPUID.06H:EAX[0] = 1
		63:32	Reserved.	
1A0H	416	IA32_MISC_ENABLE	**Enable Misc. Processor Features (R/W)** Allows a variety of processor functions to be enabled and disabled.	
		0	**Fast-Strings Enable** When set, the fast-strings feature (for REP MOVS and REP STORS) is enabled (default); when clear, fast-strings are disabled.	0F_0H
		2:1	Reserved.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		3	**Automatic Thermal Control Circuit Enable (R/W)** 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled (default). 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.	0F_0H
		6:4	Reserved	
		7	**Performance Monitoring Available (R)** 1 = Performance monitoring enabled 0 = Performance monitoring disabled	0F_0H
		10:8	Reserved.	
		11	**Branch Trace Storage Unavailable (RO)** 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported	0F_0H
		12	**Precise Event Based Sampling (PEBS) Unavailable (RO)** 1 = PEBS is not supported; 0 = PEBS is supported.	06_0FH
		15:13	Reserved.	
		16	**Enhanced Intel SpeedStep Technology Enable (R/W)** 0= Enhanced Intel SpeedStep Technology disabled 1 = Enhanced Intel SpeedStep Technology enabled	06_0DH
		17	Reserved.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		18	**ENABLE MONITOR FSM (R/W)** When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
		21:19	Reserved.	
		22	**Limit CPUID Maxval (R/W)** When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 3. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 3. 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 3, the bit is supported. Otherwise, the bit is not supported. Writing to this bit when the maximum value is greater than 3 may generate a #GP exception. Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 3.	0F_03H
		23	**xTPR Message Disable (R/W)** When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.	if CPUID.01H:ECX[14] = 1
		33:24	Reserved.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		34	**XD Bit Disable (R/W)** When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H: EDX[20]=0). When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages. BIOS must not alter the contents of this bit location, if XD bit is not supported.. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.	if CPUID.80000001H:EDX[20] = 1
		63:35	Reserved.	
1B0H	432	IA32_ENERGY_PERF_BIAS	Performance Energy Bias Hint (R/W)	if CPUID.6H:ECX[3] = 1
		3:0	Power Policy Preference: 0 indicates preference to highest performance. 15 indicates preference to maximize energy saving.	
		63:4	Reserved.	
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package Thermal Status Information (RO) Contains status information about the package’s thermal sensor. See Section 14.6, “Package Level Thermal Management.”	If CPUID.06H: EAX[6] = 1
		0	Pkg Thermal Status (RO):	
		1	Pkg Thermal Status Log (R/W):	
		2	Pkg PROCHOT # event (RO)	
		3	Pkg PROCHOT # log (R/WC0)	
		4	Pkg Critical Temperature Status (RO)	
		5	Pkg Critical Temperature Status log (R/WC0)	
		6	Pkg Thermal Threshold #1 Status (RO)	
		7	Pkg Thermal Threshold #1 log (R/WC0)	
		8	Pkg Thermal Threshold #2 Status (RO)	
		9	Pkg Thermal Threshold #1 log (R/WC0)	
		10	Pkg Power Limitation Status (RO)	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		11	Pkg Power Limitation log (R/WC0)	
		15:12	Reserved.	
		22:16	Pkg Digital Readout (RO)	
		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 14.6, "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	Pkr 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)	Trace/Profile Resource Control (R/W)	06_0EH
		0	LBR: Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	06_01H
		1	BTF: Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	06_01H
		5:2	Reserved.	
		6	TR: Setting this bit to 1 enables branch trace messages to be sent.	06_0EH
		7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	06_0EH

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	**Decimal**			
		8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	06_0EH
		9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH
		10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH
		11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H: ECX[15] = 1 and 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 3BFH) on a PMI request	If CPUID.01H: ECX[15] = 1 and 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
		63:15	Reserved.	
1F2H	498	IA32_SMRR_PHYSBASE	**SMRR Base Address (Writeable only in SMM)** Base address of SMM memory range.	If IA32_MTRR_CAP[SMRR] = 1
		7:0	Type. Specifies memory type of the range.	
		11:8	Reserved.	
		31:12	**PhysBase.** SMRR physical Base Address.	
		63:32	Reserved.	
1F3H	499	IA32_SMRR_PHYSMASK	**SMRR Range Mask. (Writeable only in SMM)** Range Mask of SMM memory range.	If IA32_MTRR_CAP[SMRR] = 1
		10:0	Reserved.	
		11	**Valid** Enable range mask.	
		31:12	**PhysMask** SMRR address range mask.	
		63:32	Reserved.	
1F8H	504	IA32_PLATFORM_DCA_CAP	DCA Capability (R)	06_0FH

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
1F9H	505	IA32_CPU_DCA_CAP	If set, CPU supports Prefetch-Hint type.	
1FAH	506	IA32_DCA_0_CAP	DCA type 0 Status and Control register.	06_2EH
		0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	06_2EH
		2:1	TRANSACTION	06_2EH
		6:3	DCA_TYPE	06_2EH
		10:7	DCA_QUEUE_SIZE	06_2EH
		12:11	Reserved.	06_2EH
		16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	06_2EH
		23:17	Reserved.	06_2EH
		24	SW_BLOCK: SW can request DCA block by setting this bit.	06_2EH
		25	Reserved.	06_2EH
		26	HW_BLOCK: Set when DCA is blocked by HW (e.g. CR0.CD = 1).	06_2EH
		31:27	Reserved.	06_2EH
200H	512	IA32_MTRR_PHYSBASE0 (MTRRphysBase0)	See Section 11.11.2.3, “Variable Range MTRRs.”	06_01H
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	06_01H
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	06_01H
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	06_01H
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	06_01H
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	06_01H
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	06_01H
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	06_01H
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	06_01H
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	06_01H
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	06_01H
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	06_01H
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	06_01H
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	06_01H
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	06_01H
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	06_01H
210H	528	IA32_MTRR_PHYSBASE8	MTRRphysBase8	if IA32_MTRR_CAP[7:0] > 8

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
211H	529	IA32_MTRR_PHYSMASK8	MTRRphysMask8	if IA32_MTRR_CAP[7:0] > 8
212H	530	IA32_MTRR_PHYSBASE9	MTRRphysBase9	if IA32_MTRR_CAP[7:0] > 9
213H	531	IA32_MTRR_PHYSMASK9	MTRRphysMask9	if IA32_MTRR_CAP[7:0] > 9
250H	592	IA32_MTRR_FIX64K_00000	MTRRfix64K_00000	06_01H
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	06_01H
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	06_01H
268H	616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	See Section 11.11.2.2, “Fixed Range MTRRs.”	06_01H
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	06_01H
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	06_01H
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	06_01H
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	06_01H
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	06_01H
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	06_01H
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	06_01H
277H	631	IA32_PAT	IA32_PAT (R/W)	06_05H
		2:0	PA0	
		7:3	Reserved.	
		10:8	PA1	
		15:11	Reserved.	
		18:16	PA2	
		23:19	Reserved.	
		26:24	PA3	
		31:27	Reserved.	
		34:32	PA4	
		39:35	Reserved.	
		42:40	PA5	
		47:43	Reserved.	
		50:48	PA6	
		55:51	Reserved.	
		58:56	PA7	
		63:59	Reserved.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
280H	640	IA32_MC0_CTL2	(R/W)	06_1AH
		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_MC0_CTL2.	06_1AH
282H	642	IA32_MC2_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
283H	643	IA32_MC3_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
284H	644	IA32_MC4_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
285H	645	IA32_MC5_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
286H	646	IA32_MC6_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
287H	647	IA32_MC7_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
288H	648	IA32_MC8_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
289H	649	IA32_MC9_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28AH	650	IA32_MC10_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28BH	651	IA32_MC11_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28CH	652	IA32_MC12_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28DH	653	IA32_MC13_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28EH	654	IA32_MC14_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28FH	655	IA32_MC15_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
290H	656	IA32_MC16_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
291H	657	IA32_MC17_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
292H	658	IA32_MC18_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
293H	659	IA32_MC19_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
294H	660	IA32_MC20_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
295H	661	IA32_MC21_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
2FFH	767	IA32_MTRR_DEF_TYPE	MTRRdefType (R/W)	06_01H
		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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
30AH	778	IA32_FIXED_CTR1 (MSR_PERF_FIXED_CTR1)	Fixed-Function Performance Counter 1 0 (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 0 0 (R/W): Counts CPU_CLK_Unhalted.Ref	If CPUID.0AH: EDX[4:0] > 2
345H	837	IA32_PERF_CAPABILITIES	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.	
38DH	909	IA32_FIXED_CTR_CTRL (MSR_PERF_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 1to count while CPL = 0.	
		5	EN1_Usr: Enable Fixed Counter 1to count while CPL > 0.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		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.	
38EH	910	IA32_PERF_GLOBAL_STATUS (MSR_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[7:0] > 0
		1	Ovf_PMC1: Overflow status of IA32_PMC1.	If CPUID.0AH: EAX[7:0] > 0
		2	Ovf_PMC2: Overflow status of IA32_PMC2.	06_2EH
		3	Ovf_PMC3: Overflow status of IA32_PMC3.	06_2EH
		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
		60:35	Reserved.	
		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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		63	CondChg: status bits of this register has changed.	If CPUID.0AH: EAX[7:0] > 0
38FH	911	IA32_PERF_GLOBAL_CTRL (MSR_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[7:0] > 0
		1	EN_PMC1	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved.	
		32	EN_FIXED_CTR0	If CPUID.0AH: EAX[7:0] > 1
		33	EN_FIXED_CTR1	If CPUID.0AH: EAX[7:0] > 1
		34	EN_FIXED_CTR2	If CPUID.0AH: EAX[7:0] > 1
		63:35	Reserved.	
390H	912	IA32_PERF_GLOBAL_OVF_CTRL (MSR_PERF_GLOBAL_OVF_CTRL)	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[7:0] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved.	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH: EAX[7:0] > 1
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EAX[7:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EAX[7:0] > 1
		60:35	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 1to clear CondChg: bit.	If CPUID.0AH: EAX[7:0] > 0
3F1H	1009	IA32_PEBS_ENABLE	PEBS Control (R/W)	
		0	Enable PEBS on IA32_PMC0.	06_0FH
		1-3	Reserved or Model specific .	
		31:4	Reserved.	
		35-32	Reserved or Model specific .	
		63:36	Reserved.	
400H	1024	IA32_MC0_CTL	MC0_CTL	P6 Family Processors
401H	1025	IA32_MC0_STATUS	MC0_STATUS	P6 Family Processors
402H	1026	IA32_MC0_ADDR[1]	MC0_ADDR	P6 Family Processors

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
403H	1027	IA32_MC0_MISC	MC0_MISC	P6 Family Processors
404H	1028	IA32_MC1_CTL	MC1_CTL	P6 Family Processors
405H	1029	IA32_MC1_STATUS	MC1_STATUS	P6 Family Processors
406H	1030	IA32_MC1_ADDR[2]	MC1_ADDR	P6 Family Processors
407H	1031	IA32_MC1_MISC	MC1_MISC	P6 Family Processors
408H	1032	IA32_MC2_CTL	MC2_CTL	P6 Family Processors
409H	1033	IA32_MC2_STATUS	MC2_STATUS	P6 Family Processors
40AH	1034	IA32_MC2_ADDR[1]	MC2_ADDR	P6 Family Processors
40BH	1035	IA32_MC2_MISC	MC2_MISC	P6 Family Processors
40CH	1036	IA32_MC3_CTL	MC3_CTL	P6 Family Processors
40DH	1037	IA32_MC3_STATUS	MC3_STATUS	P6 Family Processors
40EH	1038	IA32_MC3_ADDR[1]	MC3_ADDR	P6 Family Processors
40FH	1039	IA32_MC3_MISC	MC3_MISC	P6 Family Processors
410H	1040	IA32_MC4_CTL	MC4_CTL	P6 Family Processors
411H	1041	IA32_MC4_STATUS	MC4_STATUS	P6 Family Processors
412H	1042	IA32_MC4_ADDR[1]	MC4_ADDR	P6 Family Processors
413H	1043	IA32_MC4_MISC	MC4_MISC	P6 Family Processors
414H	1044	IA32_MC5_CTL	MC5_CTL	06_0FH
415H	1045	IA32_MC5_STATUS	MC5_STATUS	06_0FH
416H	1046	IA32_MC5_ADDR[1]	MC5_ADDR	06_0FH
417H	1047	IA32_MC5_MISC	MC5_MISC	06_0FH
418H	1048	IA32_MC6_CTL	MC6_CTL	06_1DH
419H	1049	IA32_MC6_STATUS	MC6_STATUS	06_1DH
41AH	1050	IA32_MC6_ADDR[1]	MC6_ADDR	06_1DH
41BH	1051	IA32_MC6_MISC	MC6_MISC	06_1DH
41CH	1052	IA32_MC7_CTL	MC7_CTL	06_1AH
41DH	1053	IA32_MC7_STATUS	MC7_STATUS	06_1AH
41EH	1054	IA32_MC7_ADDR[1]	MC7_ADDR	06_1AH
41FH	1055	IA32_MC7_MISC	MC7_MISC	06_1AH
420H	1056	IA32_MC8_CTL	MC8_CTL	06_1AH
421H	1057	IA32_MC8_STATUS	MC8_STATUS	06_1AH
422H	1058	IA32_MC8_ADDR[1]	MC8_ADDR	06_1AH
423H	1059	IA32_MC8_MISC	MC8_MISC	06_1AH
424H	1060	IA32_MC9_CTL	MC9_CTL	06_2EH

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
425H	1061	IA32_MC9_STATUS	MC9_STATUS	06_2EH
426H	1062	IA32_MC9_ADDR[1]	MC9_ADDR	06_2EH
427H	1063	IA32_MC9_MISC	MC9_MISC	06_2EH
428H	1064	IA32_MC10_CTL	MC10_CTL	06_2EH
429H	1065	IA32_MC10_STATUS	MC10_STATUS	06_2EH
42AH	1066	IA32_MC10_ADDR[1]	MC10_ADDR	06_2EH
42BH	1067	IA32_MC10_MISC	MC10_MISC	06_2EH
42CH	1068	IA32_MC11_CTL	MC11_CTL	06_2EH
42DH	1069	IA32_MC11_STATUS	MC11_STATUS	06_2EH
42EH	1070	IA32_MC11_ADDR[1]	MC11_ADDR	06_2EH
42FH	1071	IA32_MC11_MISC	MC11_MISC	06_2EH
430H	1072	IA32_MC12_CTL	MC12_CTL	06_2EH
431H	1073	IA32_MC12_STATUS	MC12_STATUS	06_2EH
432H	1074	IA32_MC12_ADDR[1]	MC12_ADDR	06_2EH
433H	1075	IA32_MC12_MISC	MC12_MISC	06_2EH
434H	1076	IA32_MC13_CTL	MC13_CTL	06_2EH
435H	1077	IA32_MC13_STATUS	MC13_STATUS	06_2EH
436H	1078	IA32_MC13_ADDR[1]	MC13_ADDR	06_2EH
437H	1079	IA32_MC13_MISC	MC13_MISC	06_2EH
438H	1080	IA32_MC14_CTL	MC14_CTL	06_2EH
439H	1081	IA32_MC14_STATUS	MC14_STATUS	06_2EH
43AH	1082	IA32_MC14_ADDR[1]	MC14_ADDR	06_2EH
43BH	1083	IA32_MC14_MISC	MC14_MISC	06_2EH
43CH	1084	IA32_MC15_CTL	MC15_CTL	06_2EH
43DH	1085	IA32_MC15_STATUS	MC15_STATUS	06_2EH
43EH	1086	IA32_MC15_ADDR[1]	MC15_ADDR	06_2EH
43FH	1087	IA32_MC15_MISC	MC15_MISC	06_2EH
440H	1088	IA32_MC16_CTL	MC16_CTL	06_2EH
441H	1089	IA32_MC16_STATUS	MC16_STATUS	06_2EH
442H	1090	IA32_MC16_ADDR[1]	MC16_ADDR	06_2EH
443H	1091	IA32_MC16_MISC	MC16_MISC	06_2EH
444H	1092	IA32_MC17_CTL	MC17_CTL	06_2EH
445H	1093	IA32_MC17_STATUS	MC17_STATUS	06_2EH
446H	1094	IA32_MC17_ADDR[1]	MC17_ADDR	06_2EH

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
447H	1095	IA32_MC17_MISC	MC17_MISC	06_2EH
448H	1096	IA32_MC18_CTL	MC18_CTL	06_2EH
449H	1097	IA32_MC18_STATUS	MC18_STATUS	06_2EH
44AH	1098	IA32_MC18_ADDR[1]	MC18_ADDR	06_2EH
44BH	1099	IA32_MC18_MISC	MC18_MISC	06_2EH
44CH	1100	IA32_MC19_CTL	MC19_CTL	06_2EH
44DH	1101	IA32_MC19_STATUS	MC19_STATUS	06_2EH
44EH	1102	IA32_MC19_ADDR[1]	MC19_ADDR	06_2EH
44FH	1103	IA32_MC19_MISC	MC19_MISC	06_2EH
450H	1104	IA32_MC20_CTL	MC20_CTL	06_2EH
451H	1105	IA32_MC20_STATUS	MC20_STATUS	06_2EH
452H	1106	IA32_MC20_ADDR[1]	MC20_ADDR	06_2EH
453H	1107	IA32_MC20_MISC	MC20_MISC	06_2EH
454H	1108	IA32_MC21_CTL	MC21_CTL	06_2EH
455H	1109	IA32_MC21_STATUS	MC21_STATUS	06_2EH
456H	1110	IA32_MC21_ADDR[1]	MC21_ADDR	06_2EH
457H	1111	IA32_MC21_MISC	MC21_MISC	06_2EH
480H	1152	IA32_VMX_BASIC	**Reporting Register of Basic VMX Capabilities (R/O)** See Appendix A.1, “Basic VMX Information.”	If CPUID.01H:ECX.[bit 5] = 1
481H	1153	IA32_VMX_PINBASED_CTLS	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Appendix A.3.1, “Pin-Based VM-Execution Controls.”	If CPUID.01H:ECX.[bit 5] = 1
482H	1154	IA32_VMX_PROCBASED_CTLS	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)** See Appendix A.3.2, “Primary Processor-Based VM-Execution Controls.”	If CPUID.01H:ECX.[bit 5] = 1
483H	1155	IA32_VMX_EXIT_CTLS	**Capability Reporting Register of VM-exit Controls (R/O)** See Appendix A.4, “VM-Exit Controls.”	If CPUID.01H:ECX.[bit 5] = 1
484H	1156	IA32_VMX_ENTRY_CTLS	**Capability Reporting Register of VM-entry Controls (R/O)** See Appendix A.5, “VM-Entry Controls.”	If CPUID.01H:ECX.[bit 5] = 1

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
485H	1157	IA32_VMX_MISC	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Appendix A.6, "Miscellaneous Data."	If CPUID.01H:ECX.[bit 5] = 1
486H	1158	IA32_VMX_CR0_FIXED0	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0."	If CPUID.01H:ECX.[bit 5] = 1
487H	1159	IA32_VMX_CR0_FIXED1	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0."	If CPUID.01H:ECX.[bit 5] = 1
488H	1160	IA32_VMX_CR4_FIXED0	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Appendix A.8, "VMX-Fixed Bits in CR4."	If CPUID.01H:ECX.[bit 5] = 1
489H	1161	IA32_VMX_CR4_FIXED1	**Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)** See Appendix A.8, "VMX-Fixed Bits in CR4."	If CPUID.01H:ECX.[bit 5] = 1
48AH	1162	IA32_VMX_VMCS_ENUM	**Capability Reporting Register of VMCS Field Enumeration (R/O)** See Appendix A.9, "VMCS Enumeration."	If CPUID.01H:ECX.[bit 5] = 1
48BH	1163	IA32_VMX_PROCBASED_CTLS2	**Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)** See Appendix A.3.3, "Secondary Processor-Based VM-Execution Controls."	If (CPUID.01H:ECX.[bit 5] and IA32_VMX_PROCBASED_CTLS[bit 63])
48CH	1164	IA32_VMX_EPT_VPID_CAP	**Capability Reporting Register of EPT and VPID (R/O)** See Appendix A.10, "VPID and EPT Capabilities."	If (CPUID.01H:ECX.[bit 5], IA32_VMX_PROCBASED_CTLS[bit 63], and either IA32_VMX_PROCBASED_CTLS2[bit 33] or IA32_VMX_PROCBASED_CTLS2[bit 37])
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	**Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)** See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If (CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Appendix A.4, "VM-Exit Controls."	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	**Capability Reporting Register of VM-entry Flex Controls (R/O)** See Appendix A.5, "VM-Entry Controls."	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])
491H	1169	IA32_VMX_VMFUNC	**Capability Reporting Register of VM-function Controls (R/O)**	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 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
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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	**Decimal**			
600H	1536	IA32_DS_AREA	**DS Save Area (R/W)** Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 18.12.4, "Debug Store (DS) Mechanism."	0F_0H
		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 iff not in IA-32e mode.	
6E0H	1760	IA32_TSC_DEADLINE	**TSC Target of Local APIC's TSC Deadline Mode (R/W)**	If(CPUID.01H:ECX.[bit 25] = 1
802H	2050	IA32_X2APIC_APICID	**x2APIC ID Register (R/O)** See x2APIC Specification	If (CPUID.01H:ECX.[bit 21] = 1)
803H	2051	IA32_X2APIC_VERSION	**x2APIC Version Register (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
808H	2056	IA32_X2APIC_TPR	**x2APIC Task Priority Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
80AH	2058	IA32_X2APIC_PPR	**x2APIC Processor Priority Register (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80BH	2059	IA32_X2APIC_EOI	**x2APIC EOI Register (W/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80DH	2061	IA32_X2APIC_LDR	**x2APIC Logical Destination Register (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80FH	2063	IA32_X2APIC_SIVR	**x2APIC Spurious Interrupt Vector Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
810H	2064	IA32_X2APIC_ISR0	**x2APIC In-Service Register Bits 31:0 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
811H	2065	IA32_X2APIC_ISR1	**x2APIC In-Service Register Bits 63:32 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
812H	2066	IA32_X2APIC_ISR2	**x2APIC In-Service Register Bits 95:64 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
813H	2067	IA32_X2APIC_ISR3	**x2APIC In-Service Register Bits 127:96 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
814H	2068	IA32_X2APIC_ISR4	**x2APIC In-Service Register Bits 159:128 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
815H	2069	IA32_X2APIC_ISR5	**x2APIC In-Service Register Bits 191:160 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
816H	2070	IA32_X2APIC_ISR6	**x2APIC In-Service Register Bits 223:192 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
817H	2071	IA32_X2APIC_ISR7	**x2APIC In-Service Register Bits 255:224 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
818H	2072	IA32_X2APIC_TMR0	**x2APIC Trigger Mode Register Bits 31:0 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
819H	2073	IA32_X2APIC_TMR1	**x2APIC Trigger Mode Register Bits 63:32 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81AH	2074	IA32_X2APIC_TMR2	**x2APIC Trigger Mode Register Bits 95:64 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81BH	2075	IA32_X2APIC_TMR3	**x2APIC Trigger Mode Register Bits 127:96 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81CH	2076	IA32_X2APIC_TMR4	**x2APIC Trigger Mode Register Bits 159:128 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81DH	2077	IA32_X2APIC_TMR5	**x2APIC Trigger Mode Register Bits 191:160 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81EH	2078	IA32_X2APIC_TMR6	**x2APIC Trigger Mode Register Bits 223:192 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81FH	2079	IA32_X2APIC_TMR7	**x2APIC Trigger Mode Register Bits 255:224 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
820H	2080	IA32_X2APIC_IRR0	**x2APIC Interrupt Request Register Bits 31:0 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
821H	2081	IA32_X2APIC_IRR1	**x2APIC Interrupt Request Register Bits 63:32 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
822H	2082	IA32_X2APIC_IRR2	**x2APIC Interrupt Request Register Bits 95:64 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
823H	2083	IA32_X2APIC_IRR3	**x2APIC Interrupt Request Register Bits 127:96 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
824H	2084	IA32_X2APIC_IRR4	**x2APIC Interrupt Request Register Bits 159:128 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
825H	2085	IA32_X2APIC_IRR5	**x2APIC Interrupt Request Register Bits 191:160 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
826H	2086	IA32_X2APIC_IRR6	**x2APIC Interrupt Request Register Bits 223:192 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
827H	2087	IA32_X2APIC_IRR7	**x2APIC Interrupt Request Register Bits 255:224 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
828H	2088	IA32_X2APIC_ESR	**x2APIC Error Status Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
82FH	2095	IA32_X2APIC_LVT_CMCI	**x2APIC LVT Corrected Machine Check Interrupt Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
830H	2096	IA32_X2APIC_ICR	**x2APIC Interrupt Command Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
832H	2098	IA32_X2APIC_LVT_TIMER	**x2APIC LVT Timer Interrupt Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
833H	2099	IA32_X2APIC_LVT_THERMAL	**x2APIC LVT Thermal Sensor Interrupt Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
834H	2100	IA32_X2APIC_LVT_PMI	**x2APIC LVT Performance Monitor Interrupt Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
835H	2101	IA32_X2APIC_LVT_LINT0	**x2APIC LVT LINT0 Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
836H	2102	IA32_X2APIC_LVT_LINT1	**x2APIC LVT LINT1 Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
837H	2103	IA32_X2APIC_LVT_ERROR	**x2APIC LVT Error Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
838H	2104	IA32_X2APIC_INIT_COUNT	**x2APIC Initial Count Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
839H	2105	IA32_X2APIC_CUR_COUNT	**x2APIC Current Count Register (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
83EH	2110	IA32_X2APIC_DIV_CONF	**x2APIC Divide Configuration Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
83FH	2111	IA32_X2APIC_SELF_IPI	**x2APIC Self IPI Register (W/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
C8DH	3213	IA32_QM_EVTSEL	**QoS Monitoring Event Select Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX.[bit 12] = 1)
		7:0	**Event ID:** ID of a supported QoS monitoring event to report via IA32_QM_CTR.	
		31: 8	**Reserved**.	
		N+31:32	**Resource Monitoring ID:** ID for QoS monitoring hardware to report monitored data via IA32_QM_CTR.	N = Log_2 (CPUID.(EAX= 0FH, ECX=0H).EBX[31:0] +1)
		63:N+32	**Reserved.**	
C8EH	3214	IA32_QM_CTR	**QoS Monitoring Counter Register (R/O)**	If (CPUID.(EAX=07H, ECX=0):EBX.[bit 12] = 1)
		61:0	**Resource Monitored Data**	
		62	**Unavailable**: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	**Decimal**			
		63	**Error:** If 1, indicates and unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
C8FH	3215	IA32_PQR_ASSOC	**QoS Resource Association Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX.[bit 12] = 1)
		N-1:0	**Resource Monitoring ID:** ID for QoS monitoring hardware to track internal operation, e.g. memory access.	N = Log_2 (CPUID.(EAX= 0FH, ECX=0H).EBX[31:0] +1)
		63:N	**Reserved.**	
4000_ 0000H - 4000_ 00FFH		Reserved MSR Address Space	**All existing and future processors will not implement MSR in this range.**	
C000_ 0080H		IA32_EFER	**Extended Feature Enables**	If (CPUID.80000001.EDX.[bit 20] or CPUID.80000001.EDX.[bit 29])
		0	**SYSCALL Enable (R/W)** Enables SYSCALL/SYSRET instructions in 64-bit mode.	
		7:1	Reserved.	
		8	**IA-32e Mode Enable (R/W)** Enables IA-32e mode operation.	
		9	Reserved.	
		10	**IA-32e Mode Active (R)** Indicates IA-32e mode is active when set.	
		11	**Execute Disable Bit Enable (R/W)**	
		63:12	Reserved.	
C000_ 0081H		IA32_STAR	**System Call Target Address (R/W)**	If CPUID.80000001.EDX.[bit 29] = 1
C000_ 0082H		IA32_LSTAR	**IA-32e Mode System Call Target Address (R/W)**	If CPUID.80000001.EDX.[bit 29] = 1
C000_ 0084H		IA32_FMASK	**System Call Flag Mask (R/W)**	If CPUID.80000001.EDX.[bit 29] = 1

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
C000_0100H		IA32_FS_BASE	**Map of BASE Address of FS (R/W)**	If CPUID.80000001.EDX.[bit 29] = 1
C000_0101H		IA32_GS_BASE	**Map of BASE Address of GS (R/W)**	If CPUID.80000001.EDX.[bit 29] = 1
C000_0102H		IA32_KERNEL_GS_BASE	**Swap Target of BASE Address of GS (R/W)**	If CPUID.80000001.EDX.[bit 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_MC*i*_STATUS. See Section 15.3.2.3 and Section 15.3.2.4 for more information.

...

35.3 MSRS IN THE INTEL® ATOM™ PROCESSOR FAMILY

Table 35-4 lists model-specific registers (MSRs) for Intel Atom processor family, architectural MSR addresses are also included in Table 35-4. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1CH, 06_26H, 06_27H, 06_35H and 06_36H, see Table 35-1.

The column “Shared/Unique” applies to logical processors sharing the same core in processors based on the Intel Atom microarchitecture. “Unique” means each logical processor has a separate MSR, or a bit field in an MSR governs only a logical processor. “Shared” means the MSR or the bit field in an MSR address governs the operation of both logical processors in the same core.

Table 35-4 MSRs in Intel® Atom™ Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 35.16, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Shared	See Section 35.16, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, “Monitor/Mwait Address Range Determination.” andTable 35-2
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.13, “Time-Stamp Counter,” and see Table 35-2.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
17H	23	IA32_PLATFORM_ID	Shared	**Platform ID (R)** See Table 35-2.
17H	23	MSR_PLATFORM_ID	Shared	**Model Specific Platform ID (R)**
		7:0		Reserved.
		12:8		**Maximum Qualified Ratio (R)** The maximum allowed bus ratio.
		63:13		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, “Local APIC Status and Location,” and Table 35-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	**Processor Hard Power-On Configuration (R/W)** Enables and disables processor features; **(R)** indicates current processor configuration.
		0		Reserved.
		1		**Data Error Checking Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		2		**Response Error Checking Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		3		**AERR# Drive Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		4		**BERR# Enable for initiator bus requests (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved.
		6		Reserved.
		7		**BINIT# Driver Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved.
		9		**Execute BIST (R/O)** 1 = Enabled; 0 = Disabled
		10		**AERR# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		12		**BINIT# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		13		**Reserved.**
		14		**1 MByte Power on Reset Vector (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		**APIC Cluster ID (R/O)** Always 00B.
		19: 18		Reserved.
		21: 20		**Symmetric Arbitration ID (R/O)** Always 00B.
		26:22		**Integer Bus Frequency Ratio (R/O)**
3AH	58	IA32_FEATURE_CONTROL	Unique	**Control Features in Intel 64Processor (R/W)** See Table 35-2.
40H	64	MSR_ LASTBRANCH_0_FROM_IP	Unique	**Last Branch Record 0 From IP (R/W)** One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction** for one of the last eight branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.11, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_ LASTBRANCH_1_FROM_IP	Unique	**Last Branch Record 1 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_ LASTBRANCH_2_FROM_IP	Unique	**Last Branch Record 2 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_ LASTBRANCH_3_FROM_IP	Unique	**Last Branch Record 3 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_ LASTBRANCH_4_FROM_IP	Unique	**Last Branch Record 4 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_ LASTBRANCH_5_FROM_IP	Unique	**Last Branch Record 5 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_ LASTBRANCH_6_FROM_IP	Unique	**Last Branch Record 6 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_ LASTBRANCH_7_FROM_IP	Unique	**Last Branch Record 7 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
60H	96	MSR_ LASTBRANCH_0_TO_IP	Unique	**Last Branch Record 0 To IP (R/W)** One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last eight branches, exceptions, or interrupts taken by the processor.
61H	97	MSR_ LASTBRANCH_1_TO_IP	Unique	**Last Branch Record 1 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_ LASTBRANCH_2_TO_IP	Unique	**Last Branch Record 2 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_ LASTBRANCH_3_TO_IP	Unique	**Last Branch Record 3 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_ LASTBRANCH_4_TO_IP	Unique	**Last Branch Record 4 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_ LASTBRANCH_5_TO_IP	Unique	**Last Branch Record 5 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_ LASTBRANCH_6_TO_IP	Unique	**Last Branch Record 6 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
67H	103	MSR_ LASTBRANCH_7_TO_IP	Unique	**Last Branch Record 7 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Shared	**BIOS Update Trigger Register (W)** See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Unique	**Performance counter register** See Table 35-2.
C2H	194	IA32_PMC1	Unique	**Performance Counter Register** See Table 35-2.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO)** This field indicates the intended scaleable bus clock speed for processors based on Intel Atom microarchitecture:
		2:0		▪ 111B: 083 MHz (FSB 333) ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
		63:3		Reserved.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Shared	**Memory Type Range Register (R)** See Table 35-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		**L2 Hardware Enabled (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.
		23		**L2 Not Present (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 35-2.
179H	377	IA32_MCG_CAP	Unique	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Unique	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2		**MCIP** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSEL0	Unique	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 35-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 35-2.
198H	408	MSR_PERF_STATUS	Shared	
		15:0		Current Performance State Value.
		39:16		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		63:45		Reserved.
199H	409	IA32_PERF_CTL	Unique	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	**Clock Modulation (R/W)** See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	**Thermal Interrupt Control (R/W)** See Table 35-2.
19CH	412	IA32_THERM_STATUS	Unique	**Thermal Monitor Status (R/W)** See Table 35-2.
19DH	413	MSR_THERM2_CTL	Shared	
		15:0		Reserved.
		16		**TM_SELECT (R/W)** Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:17		Reserved.
1A0	416	IA32_MISC_ENABLE	Unique	**Enable Misc. Processor Features (R/W)** Allows a variety of processor functions to be enabled and disabled.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		0		**Fast-Strings Enable** See Table 35-2.
		2:1		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2.
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available (R)** See Table 35-2.
		8		Reserved.
		9		Reserved.
		10	Shared	**FERR# Multiplexing Enable (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Shared	**Precise Event Based Sampling Unavailable (RO)** See Table 35-2.
		13	Shared	**TM2 Enable (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved.
		16	Shared	**Enhanced Intel SpeedStep Technology Enable (R/W)** See Table 35-2.
		18	Shared	**ENABLE MONITOR FSM (R/W)** See Table 35-2.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		19		Reserved.
		20	Shared	**Enhanced Intel SpeedStep Technology Select Lock (R/WO)** When set, this bit causes the following bits to become read-only: ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit), ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved.
		22	Unique	**Limit CPUID Maxval (R/W)** See Table 35-2.
		23	Shared	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Unique	**XD Bit Disable (R/W)** See Table 35-2.
		63:35		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Unique	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	**Debug Control (R/W)** See Table 35-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	**Last Exception Record From Linear IP (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	**Last Exception Record To Linear IP (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Shared	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Shared	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Shared	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Shared	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Shared	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Shared	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Shared	See Table 35-2.

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
207H	519	IA32_MTRR_PHYSMASK3	Shared	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Shared	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Shared	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Shared	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Shared	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Shared	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Shared	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Shared	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Shared	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_ 00000	Shared	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Shared	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Shared	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Shared	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Shared	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Shared	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Shared	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Shared	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Shared	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Shared	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Shared	See Table 35-2.
277H	631	IA32_PAT	Unique	See Table 35-2.
309H	777	IA32_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Shared	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_ STAUS	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Unique	See Table 35-2. See Section 18.4.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MC0_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC3_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC3_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC3_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC4_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC4_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
412H	1042	MSR_MC4_ADDR	Shared	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2. See Appendix A.1, “Basic VMX Information.”
481H	1153	IA32_VMX_PINBASED_ CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, “VM-Execution Controls.”
482H	1154	IA32_VMX_PROCBASED_ CTLS	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)** See Appendix A.3, “VM-Execution Controls.”
483H	1155	IA32_VMX_EXIT_CTLS	Unique	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, “VM-Exit Controls.”
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, “VM-Entry Controls.”
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. See Appendix A.6, “Miscellaneous Data.”
486H	1158	IA32_VMX_CR0_FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.7, “VMX-Fixed Bits in CR0.”
487H	1159	IA32_VMX_CR0_FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.7, “VMX-Fixed Bits in CR0.”
488H	1160	IA32_VMX_CR4_FIXED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.8, “VMX-Fixed Bits in CR4.”
489H	1161	IA32_VMX_CR4_FIXED1	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.8, “VMX-Fixed Bits in CR4.”

Table 35-4 MSRs in Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	**Capability Reporting Register of VMCS Field Enumeration (R/O)** See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	**DS Save Area (R/W)** See Table 35-2. See Section 18.12.4, "Debug Store (DS) Mechanism."
C000_0080H		IA32_EFER	Unique	**Extended Feature Enables** See Table 35-2.
C000_0081H		IA32_STAR	Unique	**System Call Target Address (R/W)** See Table 35-2.
C000_0082H		IA32_LSTAR	Unique	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_0084H		IA32_FMASK	Unique	**System Call Flag Mask (R/W)** See Table 35-2.
C000_0100H		IA32_FS_BASE	Unique	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_0101H		IA32_GS_BASE	Unique	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Unique	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.

Table 35-5 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor with the CPUID signature with DisplayFamily_DisplayModel of 06_27H.

Table 35-5 MSRs Supported by Intel® Atom™ Processors with CPUID Signature 06_27H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3F8H	1016	MSR_PKG_C2_RESIDENCY	Package	**Package C2 Residency** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States
		63:0	Package	Package C2 Residency Counter. (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.

Table 35-5 MSRs Supported by Intel® Atom™ Processors (Contd.)with CPUID Signature 06_27H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3F9H	1017	MSR_PKG_C4_RESIDENCY	Package	**Package C4 Residency** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States
		63:0	Package	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	**Package C6 Residency** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States
		63:0	Package	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.

35.4 MSRS IN THE PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 35-6 lists model-specific registers (MSRs) for Intel processors based on the Silvermont microarchitecture These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_37H and 06_4DH, see Table 35-1.

The column “Scope” lists the core/shared/package granularity of sharing in the Silvermont microarchitecture. “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Shared” means the MSR or the bit field is shared by more than one processor cores in the physical package. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 35.16, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Shared	See Section 35.16, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Core	See Section 8.10.5, “Monitor/Mwait Address Range Determination.” and Table 35-2
10H	16	IA32_TIME_STAMP_COUNTER	Core	See Section 17.13, “Time-Stamp Counter,” and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Shared	**Platform ID (R)** See Table 35-2.
17H	23	MSR_PLATFORM_ID	Shared	**Model Specific Platform ID (R)**
		7:0		Reserved.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		12:8		**Maximum Qualified Ratio (R)** The maximum allowed bus ratio.
		49:13		Reserved.
		52:50		**See** Table 35-2
		63:33		Reserved.
1BH	27	IA32_APIC_BASE	Core	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	**Processor Hard Power-On Configuration (R/W)** Enables and disables processor features; **(R)** indicates current processor configuration.
		0		Reserved.
		1		**Data Error Checking Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		2		**Response Error Checking Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		3		**AERR# Drive Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		4		**BERR# Enable for initiator bus requests (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved.
		6		Reserved.
		7		**BINIT# Driver Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved.
		9		**Execute BIST (R/O)** 1 = Enabled; 0 = Disabled
		10		**AERR# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved.
		12		**BINIT# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		13		**Reserved.**
		14		**1 MByte Power on Reset Vector (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		**APIC Cluster ID (R/O)** Always 00B.
		19: 18		Reserved.
		21: 20		**Symmetric Arbitration ID (R/O)** Always 00B.
		26:22		**Integer Bus Frequency Ratio (R/O)**
34H	52	MSR_SMI_COUNT	Core	**SMI Counter (R/O)**
		31:0		**SMI Count (R/O)** Running count of SMI events since last RESET.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Core	**Control Features in Intel 64Processor (R/W)** See Table 35-2.
40H	64	MSR_ LASTBRANCH_0_FROM_IP	Core	**Last Branch Record 0 From IP (R/W)** One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction** for one of the last eight branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.11, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_ LASTBRANCH_1_FROM_IP	Core	**Last Branch Record 1 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_ LASTBRANCH_2_FROM_IP	Core	**Last Branch Record 2 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_ LASTBRANCH_3_FROM_IP	Core	**Last Branch Record 3 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_ LASTBRANCH_4_FROM_IP	Core	**Last Branch Record 4 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_ LASTBRANCH_5_FROM_IP	Core	**Last Branch Record 5 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_ LASTBRANCH_6_FROM_IP	Core	**Last Branch Record 6 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
47H	71	MSR_ LASTBRANCH_7_FROM_IP	Core	**Last Branch Record 7 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_ LASTBRANCH_0_TO_IP	Core	**Last Branch Record 0 To IP (R/W)** One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last eight branches, exceptions, or interrupts taken by the processor.
61H	97	MSR_ LASTBRANCH_1_TO_IP	Core	**Last Branch Record 1 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_ LASTBRANCH_2_TO_IP	Core	**Last Branch Record 2 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_ LASTBRANCH_3_TO_IP	Core	**Last Branch Record 3 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_ LASTBRANCH_4_TO_IP	Core	**Last Branch Record 4 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_ LASTBRANCH_5_TO_IP	Core	**Last Branch Record 5 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_ LASTBRANCH_6_TO_IP	Core	**Last Branch Record 6 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
67H	103	MSR_ LASTBRANCH_7_TO_IP	Core	**Last Branch Record 7 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Core	**BIOS Update Trigger Register (W)** See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Core	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Core	**Performance counter register** See Table 35-2.
C2H	194	IA32_PMC1	Core	**Performance Counter Register** See Table 35-2.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO)** This field indicates the intended scaleable bus clock speed for processors based on Silvermont microarchitecture:
		2:0		▪ 100B: 080.0 MHz ▪ 000B: 083.3 MHz ▪ 001B: 100.0 MHz ▪ 010B: 133.3 MHz ▪ 011B: 116.7 MHz
		63:3		Reserved.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Shared	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-sate support) 001b: C1 (Behavior is the same as 000b) 100b: C4 110b: C6 111b: C7 (Silvermont only).
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_ BASE	Shared	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PMG_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.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
E7H	231	IA32_MPERF	Core	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Core	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Core	**Memory Type Range Register (R)** See Table 35-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		**L2 Hardware Enabled (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.
		23		**L2 Not Present (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Core	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Core	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Core	See Table 35-2.
179H	377	IA32_MCG_CAP	Core	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Core	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		**MCIP** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSEL0	Core	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Core	See Table 35-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 35-2.
199H	409	IA32_PERF_CTL	Core	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Core	**Clock Modulation (R/W)** See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Core	**Thermal Interrupt Control (R/W)** See Table 35-2.
19CH	412	IA32_THERM_STATUS	Core	**Thermal Monitor Status (R/W)** See Table 35-2.
1A0	416	IA32_MISC_ENABLE		**Enable Misc. Processor Features (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0	Core	**Fast-Strings Enable** See Table 35-2.
		2:1		Reserved.
		3	Shared	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2.
		6:4		Reserved.
		7	Core	**Performance Monitoring Available (R)** See Table 35-2.
		10:8		Reserved.
		11	Core	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Core	**Precise Event Based Sampling Unavailable (RO)** See Table 35-2.
		15:13		Reserved.
		16	Shared	**Enhanced Intel SpeedStep Technology Enable (R/W)** See Table 35-2.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		18	Core	**ENABLE MONITOR FSM (R/W)** See Table 35-2.
		21:19		Reserved.
		22	Core	**Limit CPUID Maxval (R/W)** See Table 35-2.
		23	Shared	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Core	**XD Bit Disable (R/W)** See Table 35-2.
		37:35		Reserved.
		38	Shared	**Turbo Mode Disable (R/W)** When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. **Note:** the power-on default value is used by BIOS to detect hardware support of turbo mode. If power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.
1A2H	418	MSR_ TEMPERATURE_TARGET	Package	
		15:0		Reserved.
		23:16		**Temperature Target (R)** The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Shared	**Offcore Response Event Select Register (R/W)**
1A7H	423	MSR_OFFCORE_RSP_1	Shared	**Offcore Response Event Select Register (R/W)**
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** **RO** if MSR_PLATFORM_INFO.[28] = 0, **RW** if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		39:32	Package	**Maximum Ratio Limit for 5C** Maximum turbo ratio limit of 5 core active.
		47:40	Package	**Maximum Ratio Limit for 6C** Maximum turbo ratio limit of 6 core active.
		55:48	Package	**Maximum Ratio Limit for 7C** Maximum turbo ratio limit of 7 core active.
		63:56	Package	**Maximum Ratio Limit for 7C** Maximum turbo ratio limit of 7 core active.
1B0H	432	IA32_ENERGY_PERF_BIAS	Core	See Table 35-2.
1C9H	457	MSR_LASTBRANCH_TOS	Core	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Core	**Debug Control (R/W)** See Table 35-2.
1DDH	477	MSR_LER_FROM_LIP	Core	**Last Exception Record From Linear IP (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Core	**Last Exception Record To Linear IP (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 35-2.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 35-2.
277H	631	IA32_PAT	Core	See Table 35-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	**Default Memory Types (R/W)** See Table 35-2.
309H	777	IA32_FIXED_CTR0	Core	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Core	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Core	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Core	See Table 35-2. See Section 17.4.1, “IA32_DEBUGCTL MSR.”
38DH	909	IA32_FIXED_CTR_CTRL	Core	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_STAUS	Core	See Table 35-2. See Section 18.4.2, “Global Counter Control Facilities.”
38FH	911	IA32_PERF_GLOBAL_CTRL	Core	See Table 35-2. See Section 18.4.2, “Global Counter Control Facilities.”

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Core	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Core	See Table 35-2. See Section 18.4.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
3F8H	1016	MSR_PKG_C4_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 C4 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C4 states. Counts at P1 clock frequency (Guaranteed Maximum Frequency).
3F9H	1017	MSR_PKG_C6C_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 C6C Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6C states. Counts at P1 clock frequency (Guaranteed Maximum Frequency)
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 P1 clock frequency (Guaranteed Maximum Frequency)
3FCH	1020	MSR_CORE_C4_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 C4 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C4 states. Counts at P1 clock frequency (Guaranteed Maximum Frequency)
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 P1 clock frequency (Guaranteed Maximum Frequency)
400H	1024	IA32_MC0_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
402H	1026	IA32_MC0_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
414H	1044	MSR_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	MSR_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	MSR_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
480H	1152	IA32_VMX_BASIC	Core	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_ CTLS	Core	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_ CTLS	Core	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Core	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Core	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Core	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Core	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Core	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Core	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Core	**Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Core	**Capability Reporting Register of VMCS Field Enumeration (R/O)** See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_ CTLS2	Core	**Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	**Capability Reporting Register of EPT and VPID (R/O)** See Table 35-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	Core	**Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)** See Table 35-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Core	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Table 35-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Core	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Table 35-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Core	**Capability Reporting Register of VM-entry Flex Controls (R/O)** See Table 35-2
491H	1169	IA32_VMX_FMFUNC	Core	**Capability Reporting Register of VM-function Controls (R/O)** See Table 35-2
4C1H	1217	IA32_A_PMC0	Core	See Table 35-2.
4C2H	1218	IA32_A_PMC1	Core	See Table 35-2.
600H	1536	IA32_DS_AREA	Core	**DS Save Area (R/W)** See Table 35-2. See Section 18.12.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)** See Section 14.7.1, "RAPL Interfaces."
610H	1552	MSR_PKG_POWER_LIMIT	Package	**PKG RAPL Power Limit Control (R/W)** See Section 14.7.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	**PKG Energy Status (R/O)** See Section 14.7.3, "Package RAPL Domain."
638H	1592	MSR_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control (R/W)** See Section 14.7.4, "PP0/PP1 RAPL Domains."
639H	1593	MSR_PP0_ENERGY_STATUS	Package	**PP0 Energy Status (R/O)** See Section 14.7.4, "PP0/PP1 RAPL Domains."
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C1 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C1 states. Counts at P1 clock frequency (Guaranteed Maximum Frequency)
6E0H	1760	IA32_TSC_DEADLINE	Core	**TSC Target of Local APIC's TSC Deadline Mode (R/W)** See Table 35-2

Table 35-6 MSRs in Intel Processors Based on the Silvermont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
C000_ 0080H		IA32_EFER	Core	**Extended Feature Enables** See Table 35-2.
C000_ 0081H		IA32_STAR	Core	**System Call Target Address (R/W)** See Table 35-2.
C000_ 0082H		IA32_LSTAR	Core	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_ 0084H		IA32_FMASK	Core	**System Call Flag Mask (R/W)** See Table 35-2.
C000_ 0100H		IA32_FS_BASE	Core	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_ 0101H		IA32_GS_BASE	Core	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0102H		IA32_KERNEL_GSBASE	Core	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0103H		IA32_TSC_AUX	Core	**AUXILIARY TSC Signature. (R/W)** See Table 35-2

...

35.8 MSRS IN INTEL® PROCESSOR FAMILY BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Table 35-12 lists model-specific registers (MSRs) that are common to Intel® processor family based on Intel microarchitecture code name Sandy Bridge. All architectural MSRs listed in Table 35-2 are supported. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2AH, 06_2DH, see Table 35-1. Additional MSRs specific to 06_2AH are listed in Table 35-13.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 35.16, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Thread	See Section 35.16, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_ SIZE	Thread	See Section 8.10.5, “Monitor/Mwait Address Range Determination,” and Table 35-2.
10H	16	IA32_TIME_STAMP_ COUNTER	Thread	See Section 17.13, “Time-Stamp Counter,” and see Table 35-2.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17H	23	IA32_PLATFORM_ID	Package	**Platform ID (R)** See Table 35-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
34H	52	MSR_SMI_COUNT	Thread	**SMI Counter (R/O)**
		31:0		**SMI Count (R/O)** Count SMIs.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64Processor (R/W)** See Table 35-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	**BIOS Update Trigger Register (W)** See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Thread	**Performance Counter Register** See Table 35-2.
C2H	194	IA32_PMC1	Thread	**Performance Counter Register** See Table 35-2.
C3H	195	IA32_PMC2	Thread	**Performance Counter Register** See Table 35-2.
C4H	196	IA32_PMC3	Thread	**Performance Counter Register** See Table 35-2.
C5H	197	IA32_PMC4	Core	**Performance Counter Register** See Table 35-2.
C6H	198	IA32_PMC5	Core	**Performance Counter Register** See Table 35-2.
C7H	199	IA32_PMC6	Core	**Performance Counter Register** See Table 35-2.
C8H	200	IA32_PMC7	Core	**Performance Counter Register** See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		24:16		Reserved.
		25		**C3 state auto demotion enable (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		**C1 state auto demotion enable (R/W)** When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		**Enable C3 undemotion (R/W)** When set, enables undemotion from demoted C3.
		28		**Enable C1 undemotion (R/W)** When set, enables undemotion from demoted C1.
		63:29		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PMG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_ PERFEVTSEL0	Thread	See Table 35-2.
187H	391	IA32_ PERFEVTSEL1	Thread	See Table 35-2.
188H	392	IA32_ PERFEVTSEL2	Thread	See Table 35-2.
189H	393	IA32_ PERFEVTSEL3	Thread	See Table 35-2.
18AH	394	IA32_ PERFEVTSEL4	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18BH	395	IA32_ PERFEVTSEL5	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18CH	396	IA32_ PERFEVTSEL6	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18DH	397	IA32_ PERFEVTSEL7	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
198H	408	IA32_PERF_STATUS	Package	See Table 35-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
198H	408	MSR_PERF_STATUS	Package	

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		47:32		Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2^13).
199H	409	IA32_PERF_CTL	Thread	See Table 35-2.
19AH	410	IA32_CLOCK_ MODULATION	Thread	**Clock Modulation (R/W)** See Table 35-2 IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		3:0		**On demand Clock Modulation Duty Cycle (R/W)** In 6.25% increment
		4		**On demand Clock Modulation Enable (R/W)**
		63:5		Reserved.
19BH	411	IA32_THERM_INTERRUPT	Core	**Thermal Interrupt Control (R/W)** See Table 35-2.
19CH	412	IA32_THERM_STATUS	Core	**Thermal Monitor Status (R/W)** See Table 35-2.
1A0	416	IA32_MISC_ENABLE		**Enable Misc. Processor Features (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0	Thread	**Fast-Strings Enable** See Table 35-2
		6:1		Reserved.
		7	Thread	**Performance Monitoring Available (R)** See Table 35-2.
		10:8		Reserved.
		11	Thread	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Thread	**Precise Event Based Sampling Unavailable (RO)** See Table 35-2.
		15:13		Reserved.
		16	Package	**Enhanced Intel SpeedStep Technology Enable (R/W)** See Table 35-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 35-2.
		21:19		Reserved.
		22	Thread	**Limit CPUID Maxval (R/W)** See Table 35-2.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23	Thread	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Thread	**XD Bit Disable (R/W)** See Table 35-2.
		37:35		Reserved.
		38	Package	**Turbo Mode Disable (R/W)** When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. **Note:** the power-on default value is used by BIOS to detect hardware support of turbo mode. If power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.
1A2H	418	MSR_TEMPERATURE_TARGET	Unique	
		15:0		Reserved.
		23:16		**Temperature Target (R)** The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	**Offcore Response Event Select Register (R/W)**
1A7H	422	MSR_OFFCORE_RSP_1	Thread	**Offcore Response Event Select Register (R/W)**
1AAH	426	MSR_MISC_PWR_MGMT		See http://biosbits.org.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 35-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 35-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 35-2.
1C8H	456	MSR_LBR_SELECT	Thread	**Last Branch Record Filtering Select Register (R/W)** See Section 17.6.2, "Filtering of Last Branch Records."
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control (R/W)** See Table 35-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	**Last Exception Record From Linear IP (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	**Last Exception Record To Linear IP (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
1FCH	508	MSR_POWER_CTL	Core	See http://biosbits.org.
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 35-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 35-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 35-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 35-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 35-2.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
250H	592	IA32_MTRR_FIX64K_ 00000	Thread	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Thread	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Thread	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 35-2.
277H	631	IA32_PAT	Thread	See Table 35-2.
280H	640	IA32_MC0_CTL2	Core	See Table 35-2.
281H	641	IA32_MC1_CTL2	Core	See Table 35-2.
282H	642	IA32_MC2_CTL2	Core	See Table 35-2.
283H	643	IA32_MC3_CTL2	Core	See Table 35-2.
284H	644	MSR_MC4_CTL2	Package	Always 0 (CMCI not supported).
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	**Default Memory Types (R/W)** See Table 35-2.
309H	777	IA32_FIXED_CTR0	Thread	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Thread	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Thread	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format. See Table 35-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 35-2.
		11:8		PEBS_REC_FORMAT. See Table 35-2.
		12		SMM_FREEZE. See Table 35-2.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:13		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_STAUS	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.7.1.1, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		63:36		Reserved.
3F6H	1014	MSR_PEBS_LD_LAT	Thread	see See Section 18.7.1.2, "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 35-12 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."

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
410H	1040	MSR_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
		0		**PCU Hardware Error (R/W)** When set, enables signaling of PCU hardware detected errors.
		1		**PCU Controller Error (R/W)** When set, enables signaling of PCU controller detected errors
		2		**PCU Firmware Error (R/W)** When set, enables signaling of PCU firmware detected errors
		63:2		Reserved.
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
480H	1152	IA32_VMX_BASIC	Thread	**Reporting Register of Basic VMX Capabilities (R/O)** SeeTable 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_ CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_ CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Thread	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
487H	1159	IA32_VMX_CR0_FIXED1	Thread	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	**Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	**Capability Reporting Register of VMCS Field Enumeration (R/O)** See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Thread	**Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Thread	**Capability Reporting Register of EPT and VPID (R/O)** See Table 35-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)** See Table 35-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Table 35-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Thread	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Table 35-22
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Flex Controls (R/O)** See Table 35-2
4C1H	1217	IA32_A_PMC0	Thread	See Table 35-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 35-2
4C3H	1219	IA32_A_PMC2	Thread	See Table 35-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 35-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 35-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 35-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 35-2.
4C8H	200	IA32_A_PMC7	Core	See Table 35-2.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
600H	1536	IA32_DS_AREA	Thread	**DS Save Area (R/W)** See Table 35-2. See Section 18.12.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)** See Section 14.7.1, "RAPL Interfaces."
60AH	1546	MSR_PKGC3_IRTL	Package	**Package C3 Interrupt Response Limit (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		**Time Unit (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved.
		15		**Valid (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
60BH	1547	MSR_PKGC6_IRTL	Package	**Package C6 Interrupt Response Limit (R/W)** This MSR defines the budget allocated for the package to exit from C6 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency amy be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C6 state.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		12:10		**Time Unit (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved.
		15		**Valid (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		**Package C2 Residency Counter. (R/O)** Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_POWER_LIMIT	Package	**PKG RAPL Power Limit Control (R/W)** See Section 14.7.3, “Package RAPL Domain.”
611H	1553	MSR_PKG_ENERGY_STATUS	Package	**PKG Energy Status (R/O)** See Section 14.7.3, “Package RAPL Domain.”
614H	1556	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameters (R/W)** See Section 14.7.3, “Package RAPL Domain.”
638H	1592	MSR_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control (R/W)** See Section 14.7.4, “PP0/PP1 RAPL Domains.”
639H	1593	MSR_PP0_ENERGY_STATUS	Package	**PP0 Energy Status (R/O)** See Section 14.7.4, “PP0/PP1 RAPL Domains.”
680H	1664	MSR_ LASTBRANCH_0_FROM_IP	Thread	**Last Branch Record 0 From IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction** for one of the last sixteen branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.6.1, “LBR Stack.”

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
681H	1665	MSR_ LASTBRANCH_1_FROM_IP	Thread	**Last Branch Record 1 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_ LASTBRANCH_2_FROM_IP	Thread	**Last Branch Record 2 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_ LASTBRANCH_3_FROM_IP	Thread	**Last Branch Record 3 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_ LASTBRANCH_4_FROM_IP	Thread	**Last Branch Record 4 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_ LASTBRANCH_5_FROM_IP	Thread	**Last Branch Record 5 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_ LASTBRANCH_6_FROM_IP	Thread	**Last Branch Record 6 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_ LASTBRANCH_7_FROM_IP	Thread	**Last Branch Record 7 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_ LASTBRANCH_8_FROM_IP	Thread	**Last Branch Record 8 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_ LASTBRANCH_9_FROM_IP	Thread	**Last Branch Record 9 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_ LASTBRANCH_10_FROM_ IP	Thread	**Last Branch Record 10 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_ LASTBRANCH_11_FROM_ IP	Thread	**Last Branch Record 11 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_ LASTBRANCH_12_FROM_ IP	Thread	**Last Branch Record 12 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_ LASTBRANCH_13_FROM_ IP	Thread	**Last Branch Record 13 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_ LASTBRANCH_14_FROM_ IP	Thread	**Last Branch Record 14 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_ LASTBRANCH_15_FROM_ IP	Thread	**Last Branch Record 15 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6C0H	1728	MSR_ LASTBRANCH_0_TO_IP	Thread	**Last Branch Record 0 To IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor.
6C1H	1729	MSR_ LASTBRANCH_1_TO_IP	Thread	**Last Branch Record 1 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_ LASTBRANCH_2_TO_IP	Thread	**Last Branch Record 2 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_ LASTBRANCH_3_TO_IP	Thread	**Last Branch Record 3 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_ LASTBRANCH_4_TO_IP	Thread	**Last Branch Record 4 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_ LASTBRANCH_5_TO_IP	Thread	**Last Branch Record 5 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_ LASTBRANCH_6_TO_IP	Thread	**Last Branch Record 6 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_ LASTBRANCH_7_TO_IP	Thread	**Last Branch Record 7 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_ LASTBRANCH_8_TO_IP	Thread	**Last Branch Record 8 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_ LASTBRANCH_9_TO_IP	Thread	**Last Branch Record 9 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_ LASTBRANCH_10_TO_IP	Thread	**Last Branch Record 10 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_ LASTBRANCH_11_TO_IP	Thread	**Last Branch Record 11 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_ LASTBRANCH_12_TO_IP	Thread	**Last Branch Record 12 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_ LASTBRANCH_13_TO_IP	Thread	**Last Branch Record 13 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_ LASTBRANCH_14_TO_IP	Thread	**Last Branch Record 14 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_ LASTBRANCH_15_TO_IP	Thread	**Last Branch Record 15 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6E0H	1760	IA32_TSC_DEADLINE	Thread	See Table Table 35-2.

Table 35-12 MSRs Supported by Intel® Processors Based on Intel® Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
802H-83FH		X2APIC MSRs	Thread	See Table 35-2.
C000_0080H		IA32_EFER	Thread	**Extended Feature Enables** See Table 35-2.
C000_0081H		IA32_STAR	Thread	**System Call Target Address (R/W)** See Table 35-2.
C000_0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_0084H		IA32_FMASK	Thread	**System Call Flag Mask (R/W)** See Table 35-2.
C000_0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Thread	**Swap Target of BASE Address of GS (R/W)** See Table 35-2
C000_0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature (R/W)** See Table 35-2 and Section 17.13.2, "IA32_TSC_AUX Register and RDTSCP Support."

35.8.1 MSRs In 2nd Generation Intel® Core™ Processor Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 35-13 lists 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 35-1.

Table 35-13 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	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.

Table 35-13 MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Core 0 select
		1		Core 1 select
		2		Core 2 select
		3		Core 3 select
		18:4		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
		63:32		Reserved.
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU main status
		0		Fixed counter overflowed
		1		An ARB counter overflowed
		2		Reserved
		3		A CBox counter overflowed (on any slice)
		63:4		Reserved.
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore fixed counter control (R/W)
		19:0		Reserved
		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

Table 35-13 MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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_PER_CTR0	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PER_CTR1	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
60CH	1548	MSR_PKGC7_IRTL	Package	**Package C7 Interrupt Response Limit (R/W)** This MSR defines the budget allocated for the package to exit from C7 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency amy be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C7 state.
		12:10		**Time Unit (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved.
		15		**Valid (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
63AH	1594	MSR_PP0_POLICY	Package	**PP0 Balance Policy (R/W)** See Section 14.7.4, “PP0/PP1 RAPL Domains.”

Table 35-13 MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
63BH	1595	MSR_PP0_PERF_STATUS	Package	**PP0 Performance Throttling Status (R/O)** See Section 14.7.4, "PP0/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	**PP1 RAPL Power Limit Control (R/W)** See Section 14.7.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	**PP1 Energy Status (R/O)** See Section 14.7.4, "PP0/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	**PP1 Balance Policy (R/W)** See Section 14.7.4, "PP0/PP1 RAPL Domains."
700H	1792	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, counter 0 event select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, counter 1 event select MSR
706H	1798	MSR_UNC_CBO_0_PER_CTR0	Package	Uncore C-Box 0, performance counter 0
707H	1799	MSR_UNC_CBO_0_PER_CTR1	Package	Uncore C-Box 0, performance counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, counter 0 event select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, counter 1 event select MSR
716H	1814	MSR_UNC_CBO_1_PER_CTR0	Package	Uncore C-Box 1, performance counter 0
717H	1815	MSR_UNC_CBO_1_PER_CTR1	Package	Uncore C-Box 1, performance counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, counter 0 event select MSR
721H	1824	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, counter 1 event select MSR
726H	1830	MSR_UNC_CBO_2_PER_CTR0	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PER_CTR1	Package	Uncore C-Box 2, performance counter 1
730H	1840	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, counter 0 event select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, counter 1 event select MSR.
736H	1846	MSR_UNC_CBO_3_PER_CTR0	Package	Uncore C-Box 3, performance counter 0.

Table 35-13 MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	**Dec**			
737H	1847	MSR_UNC_CBO_3_PER_CTR1	Package	Uncore C-Box 3, performance counter 1.

35.8.2 MSRs In Intel® Xeon® Processor E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 35-14 lists selected model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family (based on Intel® microarchitecture code name Sandy Bridge). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH, see Table 35-1.

Table 35-14 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Register Address		Register Name	Scope	Bit Description
Hex	**Dec**			
17FH	383	MSR_ERROR_CONTROL	Package	**MC Bank Error Configuration (R/W)**
		0		Reserved
		1		**MemError Log Enable (R/W)** When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		39:32	Package	**Maximum Ratio Limit for 5C** Maximum turbo ratio limit of 5 core active.
		47:40	Package	**Maximum Ratio Limit for 6C** Maximum turbo ratio limit of 6 core active.

Table 35-14 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		55:48	Package	**Maximum Ratio Limit for 7C** Maximum turbo ratio limit of 7 core active.
		63:56	Package	**Maximum Ratio Limit for 8C** Maximum turbo ratio limit of 8 core active.
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
39CH	924	MSR_PEBS_NUM_ALT	Package	
		0		**ENABLE_PEBS_NUM_ALT (RW)** Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see Table 21-9
		63:1		Reserved (must be zero).
414H	1044	MSR_MC5_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
415H	1045	MSR_MC5_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
416H	1046	MSR_MC5_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
417H	1047	MSR_MC5_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
419H	1049	MSR_MC6_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
41AH	1050	MSR_MC6_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
41BH	1051	MSR_MC6_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
41DH	1053	MSR_MC7_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.

Table 35-14 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
41EH	1054	MSR_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	MSR_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	MSR_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
422H	1058	MSR_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	MSR_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
424H	1060	MSR_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	MSR_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
426H	1062	MSR_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	MSR_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	MSR_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	MSR_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42AH	1066	MSR_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	MSR_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	MSR_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	MSR_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42EH	1070	MSR_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	MSR_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	MSR_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	MSR_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
432H	1074	MSR_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	MSR_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	MSR_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	MSR_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
436H	1078	MSR_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	MSR_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	MSR_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	MSR_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43AH	1082	MSR_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	MSR_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	MSR_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	MSR_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43EH	1086	MSR_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."

Table 35-14 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
43FH	1087	MSR_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	MSR_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	MSR_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
442H	1090	MSR_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	MSR_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	MSR_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	MSR_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
446H	1094	MSR_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	MSR_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	MSR_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	MSR_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44AH	1098	MSR_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	MSR_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	MSR_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	MSR_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44EH	1102	MSR_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	MSR_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
613H	1555	MSR_PKG_PERF_STATUS	Package	**Package RAPL Perf Status (R/O)**
618H	1560	MSR_DRAM_POWER_LIMIT	Package	**DRAM RAPL Power Limit Control (R/W)** See Section 14.7.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_ STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.7.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.7.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.7.5, "DRAM RAPL Domain."

35.9 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY (BASED ON INTEL® MICROARCHITECTURE CODE NAME IVY BRIDGE)

The 3rd generation Intel® Core™ processor family and Intel Xeon processor E3-1200v2 product family (based on Intel microarchitecture code name Ivy Bridge) supports the MSR interfaces listed in Table 35-12, Table 35-13 and Table 35-15.

Table 35-15 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	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		31:30		Reserved.
		32	Package	**Low Power Mode Support (LPM) (R/O)** When set to 1, indicates that LPM is supported, and when set to 0, indicates LPM is not supported.
		34:33	Package	**Number of ConfigTDP Levels (R/O)** 00: Only nominal TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 11: Reserved
		39:35		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		55:48	Package	**Minimum Operating Ratio (R/O)** Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.

Table 35-15 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		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		24:16		Reserved.
		25		**C3 state auto demotion enable (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		**C1 state auto demotion enable (R/W)** When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		**Enable C3 undemotion (R/W)** When set, enables undemotion from demoted C3.
		28		**Enable C1 undemotion (R/W)** When set, enables undemotion from demoted C1.
		63:29		Reserved.
648H	1608	MSR_CONFIG_TDP_ NOMINAL	Package	**Nominal TDP Ratio (R/O)**
		7:0		**Config_TDP_Nominal** Nominal TDP level ratio to be used for this specific processor (in units of 100 MHz).

Table 35-15 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			
		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.
		47		Reserved
		62:48		PKG_MIN_PWR_LVL1. MIN Power setting allowed for ConfigTDP Level 1.
		63		Reserved.
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O)
		14:0		PKG_TDP_LVL2. Power setting for ConfigTDP Level 2.
		15		Reserved
		23:16		Config_TDP_LVL2_Ratio. ConfigTDP level 2 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL2. Max Power setting allowed for ConfigTDP Level 2.
		47		Reserved
		62:48		PKG_MIN_PWR_LVL2. MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved.
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	**ConfigTDP Control (R/W)**
		1:0		**TDP_LEVEL (RW/L)** System BIOS can program this field.
		30:2		Reserved.
		31		**Config_TDP_Lock (RW/L)** When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	**ConfigTDP Control (R/W)**

Table 35-15 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			
		7:0		**MAX_NON_TURBO_RATIO (RW/L)** System BIOS can program this field.
		30:8		Reserved.
		31		**TURBO_ACTIVATION_RATIO_Lock (RW/L)** When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.

35.9.1 MSRs In Intel® Xeon® Processor E5 v2 Product Family (Based on Intel® Microarchitecture Code Name Ivy Bridge-EP)

Table 35-16 lists model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 v2 Product Family (based on Intel microarchitecture code name Ivy Bridge-EP). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3EH, see Table 35-1. These processors supports the MSR interfaces listed in Table 35-12, and Table 35-16.

Table 35-16 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Intel® microarchitecture code name Ivy Bridge-EP)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved.

Table 35-16 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Intel® microarchitecture code name Ivy Bridge-EP) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
179H	377	IA32_MCG_CAP	Thread	**Global Machine Check Capability (R/O)**
		7:0		Count
		8		**MCG_CTL_P**
		9		**MCG_EXT_P**
		10		**MCP_CMCI_P**
		11		**MCG_TES_P**
		15:12		Reserved.

Table 35-16 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Intel® microarchitecture code name Ivy Bridge-EP) (Contd.)

Register Address		**Register Name**	**Scope**	**Bit Description**
Hex	**Dec**			
		23:16		**MCG_EXT_CNT**
		24		**MCG_SER_P**
		25		Reserved.
		26		**MCG_ELOG_P**
		63:27		Reserved.
17FH	383	MSR_ERROR_CONTROL	Package	**MC Bank Error Configuration (R/W)**
		0		Reserved
		1		**MemError Log Enable (R/W)** When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
414H	1044	MSR_MC5_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
415H	1045	MSR_MC5_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
416H	1046	MSR_MC5_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
417H	1047	MSR_MC5_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
419H	1049	MSR_MC6_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
41AH	1050	MSR_MC6_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
41BH	1051	MSR_MC6_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”

Table 35-16 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Intel® microarchitecture code name Ivy Bridge-EP) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	MSR_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41EH	1054	MSR_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	MSR_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	MSR_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
422H	1058	MSR_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	MSR_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
424H	1060	MSR_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	MSR_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
426H	1062	MSR_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	MSR_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	MSR_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	MSR_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42AH	1066	MSR_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	MSR_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	MSR_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	MSR_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42EH	1070	MSR_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	MSR_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	MSR_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	MSR_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
432H	1074	MSR_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	MSR_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	MSR_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	MSR_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
436H	1078	MSR_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	MSR_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	MSR_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	MSR_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43AH	1082	MSR_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	MSR_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	MSR_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 35-16 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Intel® microarchitecture code name Ivy Bridge-EP) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
43DH	1085	MSR_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43EH	1086	MSR_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	MSR_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	MSR_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	MSR_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
442H	1090	MSR_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	MSR_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	MSR_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	MSR_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
446H	1094	MSR_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	MSR_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	MSR_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	MSR_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44AH	1098	MSR_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	MSR_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	MSR_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	MSR_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44EH	1102	MSR_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	MSR_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
450H	1104	MSR_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
451H	1105	MSR_MC20_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
452H	1106	MSR_MC20_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
453H	1107	MSR_MC20_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
454H	1108	MSR_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
455H	1109	MSR_MC21_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
456H	1110	MSR_MC21_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
457H	1111	MSR_MC21_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
458H	1112	MSR_MC22_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
459H	1113	MSR_MC22_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
45AH	1114	MSR_MC22_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
45BH	1115	MSR_MC22_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
45CH	1116	MSR_MC23_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
45DH	1117	MSR_MC23_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.

Table 35-16 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Intel® microarchitecture code name Ivy Bridge-EP) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
45EH	1118	MSR_MC23_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
45FH	1119	MSR_MC23_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
460H	1120	MSR_MC24_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
461H	1121	MSR_MC24_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
462H	1122	MSR_MC24_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
463H	1123	MSR_MC24_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
464H	1124	MSR_MC25_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
465H	1125	MSR_MC25_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
466H	1126	MSR_MC25_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
467H	1127	MSR_MC25_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
468H	1128	MSR_MC26_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
469H	1129	MSR_MC26_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
46AH	1130	MSR_MC26_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
46BH	1131	MSR_MC26_MISC	Package	See Section 153.2.4, "IA32_MCi_MISC MSRs."
613H	1555	MSR_PKG_PERF_STATUS	Package	**Package RAPL Perf Status (R/O)**
618H	1560	MSR_DRAM_POWER_LIMIT	Package	**DRAM RAPL Power Limit Control (R/W)** See Section 14.7.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_ STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.7.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.7.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.7.5, "DRAM RAPL Domain."

35.9.2 Additional MSRs Supported by Next Generation Intel® Xeon Processor E7 family

Next Generation Intel® Xeon Processor E7 Family (based on Intel microarchitecture code name Ivy Bridge) with CPUID DisplayFamily_DisplayModel signature 06_3EH supports the MSR interfaces listed in Table 35-12, Table 35-16, and Table 35-17.

Table 35-17 Additional MSRs Supported by Next Generation Intel® Xeon Processors E7 with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
41BH	1051	IA32_MC6_MISC	Package	Misc MAC information of Integrated I/O. (R/O) see Section 15.3.2.4

Table 35-17 Additional MSRs Supported by Next Generation Intel® Xeon Processors E7 with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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

35.10 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS (BASED ON INTEL® MICROARCHITECTURE CODE NAME HASWELL)

The 4th generation Intel® Core™ processor family and Intel Xeon processor E3-1200v3 product family (based on Intel microarchitecture code name Haswell), with CPUID DisplayFamily_DisplayModel signature 06_3CH/06_45H/06_46H, support the MSR interfaces listed in Table 35-12, Table 35-13, Table 35-15, and Table 35-18.

Table 35-18 Additional MSRs Supported by 4th Generation Intel® Core Processors (based on Intel® microarchitecture code name Haswell)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		31:30		Reserved.
		32	Package	**Low Power Mode Support (LPM) (R/O)** When set to 1, indicates that LPM is supported, and when set to 0, indicates LPM is not supported.

Table 35-18 Additional MSRs Supported by 4th Generation Intel® Core Processors (based on Intel® microarchitecture code name Haswell) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	**Dec**			
		34:33	Package	**Number of ConfigTDP Levels (R/O)** 00: Only nominal TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 11: Reserved
		39:35		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		55:48	Package	**Minimum Operating Ratio (R/O)** Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved.
3BH	59	IA32_TSC_ADJUST	THREAD	**Per-Logical-Processor TSC ADJUST (R/W)** See Table 35-2.
186H	390	IA32_PERFEVTSEL0	THREAD	**Performance Event Select for Counter 0 (R/W)** Supports all fields described in Table 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
187H	391	IA32_PERFEVTSEL1	THREAD	**Performance Event Select for Counter 1 (R/W)** Supports all fields described in Table 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
188H	392	IA32_PERFEVTSEL2	THREAD	**Performance Event Select for Counter 2 (R/W)** Supports all fields described in Table 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
		33		IN_TXCP: see Section 18.11.5.1 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.

Table 35-18 Additional MSRs Supported by 4th Generation Intel® Core Processors (based on Intel® microarchitecture code name Haswell) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
189H	393	IA32_PERFEVTSEL3	THREAD	**Performance Event Select for Counter 3 (R/W)** Supports all fields described in Table 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
491H	1169	IA32_VMX_FMFUNC	THREAD	**Capability Reporting Register of VM-function Controls (R/O)** See Table 35-2
648H	1608	MSR_CONFIG_TDP_ NOMINAL	Package	**Nominal TDP Ratio (R/O)**
		7:0		**Config_TDP_Nominal** Nominal 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.
		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.

Table 35-18 Additional MSRs Supported by 4th Generation Intel® Core Processors (based on Intel® microarchitecture code name Haswell) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63		Reserved.
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	**ConfigTDP Control (R/W)**
		1:0		**TDP_LEVEL (RW/L)** System BIOS can program this field.
		30:2		Reserved.
		31		**Config_TDP_Lock (RW/L)** When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	**ConfigTDP Control (R/W)**
		7:0		**MAX_NON_TURBO_RATIO (RW/L)** System BIOS can program this field.
		30:8		Reserved.
		31		**TURBO_ACTIVATION_RATIO_Lock (RW/L)** When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.

35.10.1 Additional MSRs Supported by 4th Generation Intel® Core™ Processors

The 4th generation Intel® Core™ processor family (based on Intel microarchitecture code name Haswell) with CPUID DisplayFamily_DisplayModel signature 06_45H supports the MSR interfaces listed in Table 35-12, Table 35-13, Table 35-15, Table 35-18, and Table 35-19.

Table 35-19 Additional MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily_DisplayModel Signature 06_45H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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

Table 35-19 Additional MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily_DisplayModel Signature 06_45H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
631H	1585	MSR_PKG_C9_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C9 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.
		63:60		Reserved
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C10 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.
		63:60		Reserved

35.10.2 MSRs In 4th Generation Intel® Core™ Processor Family (Based on Intel® microarchitecture code name Haswell)

Table 35-20 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 Intel microarchitecture code name Haswell). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3CH/06_45H/ 06_46H, see Table 35-1.

Table 35-20 MSRs Supported by 4th Generation Intel® Core™ Processors (Intel® microarchitecture code name Haswell)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17DH	390	MSR_SMM_MCA_CAP	THREAD	**Enhanced SMM Capabilities (SMM-RO)** Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.
		59		Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.
		63:60		Reserved

Table 35-20 MSRs Supported by 4th Generation Intel® Core™ Processors (Intel® microarchitecture code name Haswell) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
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

Table 35-20 MSRs Supported by 4th Generation Intel® Core™ Processors (Intel® microarchitecture code name Haswell) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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_PER_CTR0	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PER_CTR1	Package	Uncore Arb unit, performance counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSEL0	Package	Uncore Arb unit, counter 0 event select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, counter 1 event select MSR
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Core 0 select
		1		Core 1 select
		2		Core 2 select
		3		Core 3 select
		18:4		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
		63:32		Reserved.
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, counter 1 event select MSR
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	**Enhanced SMM Feature Control (SMM-RW)** Reports SMM capability Enhancement. Accessible only while in SMM.

Table 35-20 MSRs Supported by 4th Generation Intel® Core™ Processors (Intel® microarchitecture code name Haswell) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0		**Lock (SMM-RWO)** When set to '1' locks this register from further changes
		1		Reserved
		2		**SMM_Code_Chk_En (SMM-RW)** This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	**SMM Delayed (SMM-RO)** Reports the interruptible state of all logical processors in the package . Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.
		N-1:0		**LOG_PROC_STATE (SMM-RO)** Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	**SMM Blocked (SMM-RO)** Reports the blocked state of all logical processors in the package . Available only while in SMM.
		N-1:0		**LOG_PROC_STATE (SMM-RO)** Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is 0FFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
640H	1600	MSR_PP1_POWER_LIMIT	Package	**PP1 RAPL Power Limit Control (R/W)** See Section 16.7.4, "PP0/PP1 RAPL Domains."

Table 35-20 MSRs Supported by 4th Generation Intel® Core™ Processors (Intel® microarchitecture code name Haswell) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
641H	1601	MSR_PP1_ENERGY_STATUS	Package	**PP1 Energy Status (R/O)** See Section 16.7.4, “PP0/PP1 RAPL Domains.”
642H	1602	MSR_PP1_POLICY	Package	**PP1 Balance Policy (R/W)** See Section 16.7.4, “PP0/PP1 RAPL Domains.”
700H	1792	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, counter 0 event select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, counter 1 event select MSR
706H	1798	MSR_UNC_CBO_0_PER_CTR0	Package	Uncore C-Box 0, performance counter 0
707H	1799	MSR_UNC_CBO_0_PER_CTR1	Package	Uncore C-Box 0, performance counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, counter 0 event select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, counter 1 event select MSR
716H	1814	MSR_UNC_CBO_1_PER_CTR0	Package	Uncore C-Box 1, performance counter 0
717H	1815	MSR_UNC_CBO_1_PER_CTR1	Package	Uncore C-Box 1, performance counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, counter 0 event select MSR
721H	1824	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, counter 1 event select MSR
726H	1830	MSR_UNC_CBO_2_PER_CTR0	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PER_CTR1	Package	Uncore C-Box 2, performance counter 1
730H	1840	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, counter 0 event select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, counter 1 event select MSR.
736H	1846	MSR_UNC_CBO_3_PER_CTR0	Package	Uncore C-Box 3, performance counter 0.
737H	1847	MSR_UNC_CBO_3_PER_CTR1	Package	Uncore C-Box 3, performance counter 1.

...