Intel® Itanium® Architecture Software Developer's Manual Specification Update

November 2003

Notice: Intel® Itanium® architecture processors may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in processor specification updates.

Document Number: 248699-008

Contents

Revision History

Date	Version Number	Description
October 2003	008	Added Specification Changes 2-4; added Specification Clarifications 5-15; added Documentation Changes 1-7.
December 2002	007	Added Specification Change 1; added Specification Clarification 1-4.
June 2002	001-006	Changes from previous Software Developer's Manual Specification Updates were incorporated into version 2.1 of the *Intel® Itanium® Architecture Software Developer's Manual* October 2002.

Preface

This document is an update to the specifications contained in the Affected Documents/Related 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.

Information types defined in Nomenclature are consolidated into the specification update and are no longer published in other documents.

This document may also contain information that was not previously published.

Affected Documents/Related Documents

Title	Document #
IIntel® Itanium® Architecture Software Developer's Manual, Volume 1: Application Architecture	245317-004
Intel® Itanium® Architecture Software Developer's Manual, Volume 2: System Architecture	245318-004
Intel® Itanium® Architecture Software Developer's Manual, Volume 3: Instruction Set Reference	245319-004

Nomenclature

Specification Changes are modifications to the current published specifications for Intel® Itanium® architecture processors. These changes will be incorporated in the next release of the specifications.

Specification Clarifications describe a specification in greater detail or further explain a specification's interpretation. These clarifications will be incorporated in the next release of the specification.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the *Intel® Itanium® Architecture Software Developer's Manual*.

Summary Table of Changes

The following tables indicate the specification changes, specification clarifications, or documentation changes that apply to the *Intel® Itanium® Architecture Software Developer's Manual*.

Specification Changes

No.	Page	SPECIFICATION CHANGES
1	8	Volume 1: ao bit added to CPUID Register 4
2	8	MCA architecture extensions for supporting data-poisoning events
3	11	LID enhancements
4	11	Extend PALE_CHECK exit options

Specification Clarifications

No.	Page	SPECIFICATION CLARIFICATIONS
1	13	Volume 2: PSR.dt serialization clarification
2	13	Volume 2: Unaligned debug fault clarification
3	13	Volume 3: Clarification on PSR requirements for br.ia/rfi instructions during PSR.is transition
4	14	Volume 3: Added Illegal Operation fault to fnma I-page
5	14	Clarify INTA/XTP definition
6	15	Clarify VHPT insert rules
7	15	Adding FP-readers to support table
8	16	cmpxchg clarifications
9	16	Add Illegal Operation fault
10	17	Non-speculative reference for WBL attribute clarification
11	19	Dirty-bit fault ISR.code clarification
12	20	FC data dependency ordering clarification
13	20	PAL_MC_DRAIN clarification
14	21	Add hint instructions to support table
15	21	Clarify speculative operation fault handler requirements

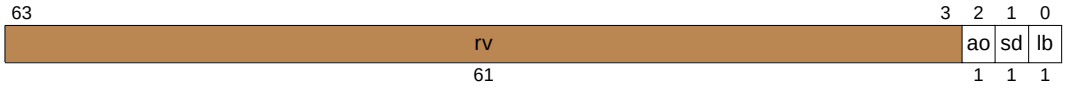
Documentation Changes

No.	Page	DOCUMENTATION CHANGES
1	23	Update IA-32 CPUID I-Page
2	29	PAL_BUS_GET/SET_FEATURES fix
3	29	PAL_COPY_PAL update
4	30	Fixing X-Unit text correction
5	30	PAL_CACHE_SHARED_INFO text correction
6	30	PAL_CACHE_FLUSH clarification and minor code sequence fix
7	30	PAL_GET_PROC_FEATURES table fix

Specification Changes

1. Volume 1: ao bit added to CPUID Register 4

1. New Figure 3-12 (page 1:30) - added a new bit for ao:



2. Table 3-8 (page 1:30) has a new entry for ao:

Field	Bits	Description
lb	0	Processor implements the long branch (brl) instructions.
sd	1	Processor implements spontaneous deferral (see Section 5.5.5, “Deferral of Speculative Load Faults” on page 2:88).
ao	2	Processor implements 16-byte atomic operations (see “ld — Load”, “st — Store” and “cmpxchg — Compare and Exchange” instructions in Volume 3).
rv	63:3	Reserved.

2. MCA architecture extensions for supporting data-poisoning events

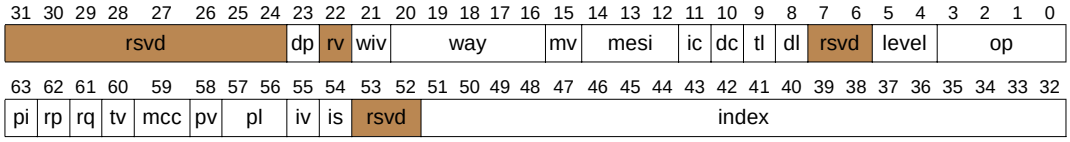
1. Volume 2: Added the following row to Table 11-54 (page 2:360) of PAL_PROC_GET_FEATURES:

Table 11-54. Processor Features

Bit	Class	Control	Description
53	Opt.	Req.	Enable MCA signaling on data-poisoning event detection. When 0, a CMCI will be signaled on error detection. When 1, an MCA will be signaled on error detection. If this feature is not supported, then the corresponding argument is ignored when calling PAL_PROC_SET_FEATURES. Note that the functionality of this bit is independent of the setting in bit 60 (Enable CMCI promotion), and that the bit 60 setting does not affect CMCI signaling for data-poisoning related events.

2. Volume 2: dp bit added to PAL_MC_ERROR_INFO Cache_Check and Bus_Check:
   a. Figure 11-37 (page 2:345) – added new bit for dp:

Figure 11-37. cache_check Layout



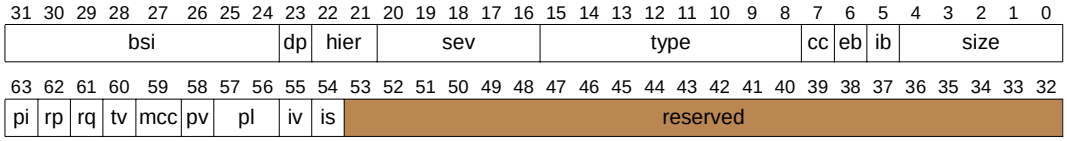
b. Table 11-47 (page 2:345) – added new entry for dp:

Table 11-47. cache_check Fields

Field	Bits	Description
op	3:0	Type of cache operation that caused the machine check: 0 – unknown or internal error 1 – load 2 – store 3 – instruction fetch or instruction prefetch 4 – data prefetch (both hardware and software) 5 – snoop (coherency check) 6 – cast out (explicit or implicit write-back of a cache line) 7 – move in (cache line fill) All other values are reserved.
level	5:4	Level of cache where the error occurred. A value of 0 indicates the first level of cache.
rsvd	7:6	Reserved
dl	8	Failure located in the data part of the cache line.
tl	9	Failure located in the tag part of the cache line.
dc	10	Failure located in the data cache
ic	11	Failure located in the instruction cache
mesi	14:12	0 – cache line is invalid. 1 – cache line is held shared. 2 – cache line is held exclusive. 3 – cache line is modified. All other values are reserved.
mv	15	The *mesi* field in the cache_check parameter is valid.
way	20:16	Failure located in the way of the cache indicated by this value.
wiv	21	The *way* and *index* field in the cache_check parameter is valid.
rsvd	22	Reserved
dp	23	A multiple-bit error was detected, and data was poisoned for the corresponding cache line during castout.
rsvd	31:24	Reserved
index	51:32	Index of the cache line where the error occurred.
rsvd	53:52	Reserved
is	54	Instruction set. If this value is set to zero, the instruction that generated the machine check was an Intel Itanium instruction. If this bit is set to one, the instruction that generated the machine check was IA-32 instruction.
iv	55	The *is* field in the cache_check parameter is valid.
pl	57:56	Privilege level. The privilege level of the instruction bundle responsible for generating the machine check.
pv	58	The *pl* field of the cache_check parameter is valid.
mcc	59	Machine check corrected: This bit is set to one to indicate that the machine check has been corrected.
tv	60	Target address is valid: This bit is set to one to indicate that a valid target address has been logged.
rq	61	Requester identifier: This bit is set to one to indicate that a valid requester identifier has been logged.
rp	62	Responder identifier: This bit is set to one to indicate that a valid responder identifier has been logged.
pi	63	Precise instruction pointer. This bit is set to one to indicate that a valid precise instruction pointer has been logged.

c. Figure 11-39 (page 2:347) – added new bit for dp:

Figure 11-39. bus_check Layout



d. Table 11-49 (page 2:347) – added new entry at bit 23 for dp:

Table 11-49. bus_check Fields

Field	Bits	Description
size	4:0	Size in bytes of the transaction that caused the machine check abort.
ib	5	Internal bus error
eb	6	External bus error
cc	7	Error occurred during a cache to cache transfer.
type	15:8	Type of transaction that caused the machine check abort. 0 – unknown 1 – partial read 2 – partial write 3 – full line read 4 – full line write 5 – implicit or explicit write-back operation 6 – snoop probe 7 – incoming or outgoing ptc.g 8 – write coalescing transactions 9 – I/O space read 10 – I/O space write 11 – inter-processor interrupt message (IPI) 12 – interrupt acknowledge or external task priority cycle All other values are reserved
sev	20:16	Bus error severity. The encodings of error severity are platform specific.
hier	22:21	This value indicates which level or bus hierarchy the error occurred in. A value of 0 indicates the first level of hierarchy.
dp	23	A multiple-bit error was detected, and data was poisoned for the incoming cache line.
bsi	31:24	Bus error status information. It describes the type of bus error. This field is processor bus specific.
reserved	53:32	Reserved
is	54	Instruction set. If this value is set to zero, the instruction that generated the machine check was an Intel Itanium instruction. If this bit is set to one, the instruction that generated the machine check was IA-32 instruction.
iv	55	The *is* field in the bus_check parameter is valid.
pl	57:56	Privilege level. The privilege level of the instruction bundle responsible for generating the machine check.
pv	58	The *pl* field of the bus_check parameter is valid.
mcc	59	Machine check corrected: This bit is set to one to indicate that the machine check has been corrected.
tv	60	Target address is valid: This bit is set to one to indicate that a valid target address has been logged.
rq	61	Requester identifier: This bit is set to one to indicate that a valid requester identifier has been logged.

Table 11-49. bus_check Fields (Continued)

Field	Bits	Description
rp	62	Responder identifier: This bit is set to one to indicate that a valid responder identifier has been logged.
pi	63	Precise instruction pointer. This bit is set to one to indicate that a valid precise instruction pointer has been logged.

3. LID enhancements

1. Volume 2, Part I, Section 5.8.3.1, page 2:104, first paragraph should now read:

   "The LID register contains the processor's local interrupt identifier. Two fields (id and eid) serve as the processor's physical name for all interrupt messages (external interrupts, INITs, and PMIs). LID is loaded by firmware during platform initialization based on the processor's physical location within the system. Processors receiving an interrupt message on the system interconnect may or may not compare their id/eid fields with the target address for the interrupt message, depending on the type of system interconnect. If this comparison is performed, then a match would indicate that the interrupt received was intended for this processor. In case of no comparison, processors use other system topology mechanisms to determine the correct target of the interrupt message."

2. Volume 2, Part I, Section 5.8.3.1, page 2:104, second paragraph:

   — Change from:

   "LID is a read-write register."

   to:

   "The LID register fields are read-only or read-write. Details of the programmability of these fields is communicated by PAL at PALE_RESET handoff (see Section 11.2.2: 'PALE_RESET Exit State' for details). Read-only LID bits always return a value of 0. Writes to read-only bits are ignored."

3. Volume 2, Part I, Section 11.2.2:

   — On page 2:259, change the GR33 bullet from:

   "GR33 contains the geographically significant unique processor ID. The value is the same as that returned by PAL_FIXED_ADDR"

   to:

   "GR33 contains information about the geographically significant unique processor ID, and a mask that indicates which bits in the LID register (CR64) are read-only. Firmware should write the processor's local interrupt identifier in the programmable portion of the LID register. Writes to the read-only bits are ignored.

   [63:48] Reserved

   [47:40] Mask indicating which bits in eid are programmable
   0 = programmable, 1 = read-only

   [39:32] Mask indicating which bits in id are programmable
   0 = programmable, 1 = read-only

   [31:16] Reserved

   [15:0] Geographically significant processor ID

   The value returned in bits [15:0] is the same as that returned by PAL_FIXED_ADDR."

4. Extend PALE_CHECK exit options

1. Volume 2, Part I, Section 11.3.1:

   a. On page 2:265, first paragraph, change the following sentence from:

"PALE_CHECK terminates by branching to SALE_ENTRY, passing the state of the processor at the time of the error."

to:

"PALE_CHECK terminates either by returning to the interrupted context or by branching to SALE_ENTRY, passing the state of the processor at the time of the error."

b. In the fifth paragraph change the following from:

"PSR.mc is set to 1 by the hardware when PALE_CHECK is entered. PSR.mc will remain set for the duration of PALE_CHECK, and PALE_CHECK will exit with psr.mc set."

to:

"PSR.mc is set to 1 by the hardware when PALE_CHECK is entered. When PALE_CHECK branches to SALE_ENTRY, PSR.mc remains set (PSR.mc is restored to its original value if PALE_CHECK terminates by returning to the interrupted context)."

And delete: "PALE_CHECK must attempt to branch to SALE_ENTRY unless code execution is not possible."

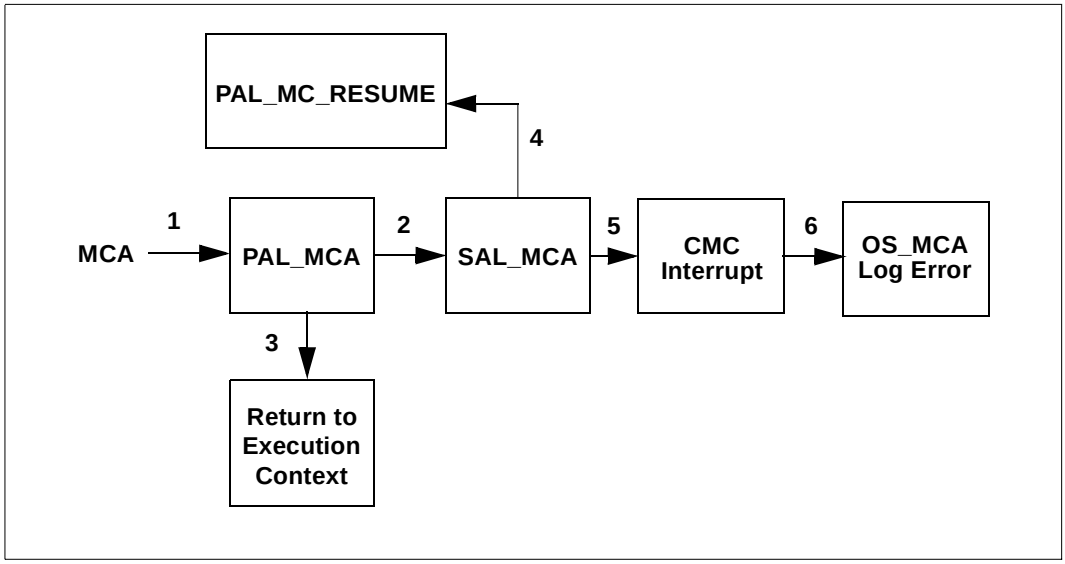
2. Volume 2, Part II, Section 13.3.1, page 2:493:

a. The second paragraph should read:

When the processor detects an error, control is transferred to the PAL_MCA entrypoint. PAL_MCA will perform error analysis and processor error correction where possible. Subsequently, PAL either returns to the interrupted context or hands off control to the SAL_MCA component. The level of recovery provided by PAL_MCA is implementation dependent and is beyond the scope of this specification. SAL_MCA will perform error logging and platform error correction where possible. Errors that are corrected by PAL and SAL firmware are logged and control is transferred back to the interrupted process/context. For corrected errors, no OS intervention is required for error handling, but the OS is notified of the event for logging purposes through a low priority asynchronous corrected machine check interrupt (CMCI). See Section 5.8.3.8, "Corrected Machine Check Vector (CMCV – CR74)" for more information on the CMCI. If the error was not corrected by firmware, SAL hands off control to the OS_MCA handler.

b. Added correctable machine check flow:

Figure 13-3. Correctable Machine Check Code Flow



Specification Clarifications

1. Volume 2: PSR.dt serialization clarification

1. Volume 2, Part I, Section 3.3.2, Table 3-2:

— On page 2:20, change the Serialization Required column for PSR.dt from:

"data"

to:

"inst/data"

and add a cross-reference to footnote c.

2. Volume 2: Unaligned debug fault clarification

1. Volume 2, Part I, Section 8.3, Debug vector page:

— On page 2:175, add the following paragraph to the end of the Notes section:

If unaligned accesses are being performed with debug faults enabled, this fault may be taken even though there is not a match for the address programmed in the breakpoint register. See Volume 2, Section 7.1.2, "Debug Address Breakpoint Match Conditions."

3. Volume 3: Clarification on PSR requirements for `br.ia`/`rfi` instructions during PSR.is transition

1. `br.ia` instruction page (Volume 3, p. 3:18):

a. Under "ia" bullet, add the following paragraph after the 3rd paragraph:

Software must set PSR properly before branching to the IA-32 instruction set; otherwise processor operation is undefined. See Volume 2, Table 3-2, "Processor Status Register Fields" on page 2:19 for details.

b. In the "Operation" section on page 3:22 under "case 'ia'," add below "`tmp_taken = 1;`":

```
if (CR[IPSR].ic==0 || CR[IPSR].dt==0 || CR[IPSR].mc==1 ||
CR[IPSR].it==0)
    undefined_behavior();
```

2. `rfi` instruction page (Volume 3, p. 3:204):

a. In the "Description" section, before the paragraph beginning "Software must issue a `mf` instruction...," add the following paragraph:

If IPSR.is is 1, software must set other IPSR fields properly for IA-32 instruction set execution; otherwise processor operation is undefined. See Volume 2, Table 3-2, "Processor Status Register Fields" on page 2:19 for details.

b. In the "Operation" section:

Add the following below, "`if (CR[IPSR].is == 1) {`":

```
if (CR[IPSR].ic==0 || CR[IPSR].dt==0 || CR[IPSR].mc==1 ||
CR[IPSR].it==0)
    undefined_behavior();
```

3. Table 3-1, Volume 3: Pseudo-Code Functions chapter:

a. On page 3:253, replace the bullet list of faults in the Operation column of the tlb_translate() row of the Pseudo-Code Functions table with this new bullet list:

- Unimplemented Data Address fault
- Data Nested TLB fault
- Alternate Data TLB fault
- VHPT Data fault
- Data TLB fault
- Data Page Not Present fault
- Data NaT Page Consumption fault
- Data Key Miss fault
- Data Key Permission fault
- Data Access Rights fault
- Data Dirty Bit fault
- Data Access Bit fault
- Data Debug fault
- Unaligned Data Reference fault
- Unsupported Data Reference fault

b. Replace the bullet list of faults in the Operation column of the tlb_translate_nonaccess() row of the Pseudo-Code Functions table with this new bullet list:

- Unimplemented Data Address fault
- Data Nested TLB fault
- Alternate Data TLB fault
- VHPT Data fault
- Data TLB fault
- Data Page Not Present fault
- Data NaT Page Consumption fault
- Data Access Rights fault (`fc` only)

4. Volume 3: Added Illegal Operation fault to fnma I-page

1. Volume 3, `fnma` instruction page:

   On page 3:81, add “Illegal Operation fault” to the list of interruptions in the Interruptions section.

5. Clarify INTA/XTP definition

1. On page 2:112, Volume 2, Part I, Section 5.8.4.3, “Interrupt Acknowledge (INTA) Cycle”:
   — Add the following sentence to the end of the 2nd paragraph:
   “Any memory operation to the INTA address other than a single byte load is undefined.”
2. On page 2:112, Volume 2, Part I, Section 5.8.4.4, “External Task Priority (XTP) Cycle”:
   — Add the following sentence to the end of the 1st paragraph:
   “Any memory operation to the XTP address other than a single byte store is undefined.”

6. Clarify VHPT insert rules

1. Volume 2, Part I, Section 4.1.8:

— Replace the 2nd paragraph and insert new table on the bottom of page 2:58 with:

The VHPT walker's inserts into the TC follow purge-before-insert rules similar to those for software inserts (see Table 4-1, “Purge Behavior of TLB Instructions,” on page 2:49). VHPT walker inserts into the DTC behave similar to `itc.d`; VHPT walker inserts into the ITC behave similar to `itc.i`. If an instruction reference results in a VHPT walk that misses in the data TLB, the DTC insert for the translation for the VHPT acts similar to an `itc.d`.

As described in Section 4.1, “Virtual Addressing” on page 2:43, processors may optionally use VRN bits when searching for a matching translation for a memory reference (references other than inserts and purges). In processors that do use VRN bits for such searches, VHPT inserts may also use VRN bits in searching for overlapping entries. Thus, if a VHPT insertion overlaps a translation in the TC, but the VRN of the address being inserted does not match the VRN of the existing TC translation, the purge of the existing TC entry is optional. If a VHPT insertion overlaps a translation in a TR, but the VRN of the address being inserted does not match the VRN of the TR translation, the VHPT insertion is allowed, and a machine check is optional. In processors which do not use VRN bits when searching for a matching translation for a memory reference, the behavior of VHPT inserts is identical to that of software inserts (see Table 4-1, “Purge Behavior of TLB Instructions,” on page 2:49).

If a VHPT insert overlaps with an existing TR entry and the VRN of the insertion matches the VRN of the existing TR entry (for example, if the translation being inserted is for a large page which overlaps with a small page translation in the TR), the VHPT insertion can be done, but a machine check must be raised. Software must not create overlapping translations in the VHPT that are larger than a currently existing TR translation.

The behavior of VHPT inserts is summarized in Table 4-9.

Table 4-9. Behavior of VHPT Inserts

VHPT Inserts	VRN bits used for TLB searching		VRN bits not used for TLB searching
	VRN match	No VRN match	
VHPT insert overlaps TC entry	May insert[a] Must purge[b]	May insert May purge[c]	May insert Must purge
VHPT insert overlaps TR entry	May insert Must Machine Check[d]	May insert May Machine Check[e]	Must not insert Must Machine Check

a. May insert: indicates that the VHPT may perform an insert into the TC
b. Must purge: requires that all partially or fully overlapped translations are removed prior to the insert or purge operation.
c. May purge: indicates that a processor may remove partially or fully overlapped translations prior to the insert or purge operation. However, software must not rely on the purge.
d. Must Machine Check: indicates that a processor will cause a Machine Check abort.
e. May Machine Check: indicates that a processor may cause a Machine Check abort based on the implementation.

7. Adding FP-readers to support table

1. Volume 3, Table 5-5 - “Instruction Classes”, on page 3:352, add:

“mem-writers-fp”

to the row:

“fr-readers”

8. cmpxchg clarifications

1. Volume 2, Section 7.1.2, page 2:134:
   — Add a new bullet list item at the bottom of the first bullet list, with this text:
   "The `cmp8xchg16` operands are treated as 16-byte datums for both read and write breakpoint matching, even though this instruction only reads 8 bytes."
2. `cmpxchg` I-page (Volume 3, page 3:40):
   — In first paragraph of the Description section, change this sentence from:
   "For `cmp8xchg16`, if the two are equal, then 8-bytes from GR r2 are stored at the specified address ignoring bit 3 (GR r3 & ~0x4), and 8 bytes from the Compare and Store Data application register (AR[CSD]) are stored at that address + 8 ((GR r3 & ~0x4) + 8)."
   to:
   "For `cmp8xchg16`, if the two are equal, then 8-bytes from GR r2 are stored at the specified address ignoring bit 3 (GR r3 & ~0x8), and 8 bytes from the Compare and Store Data application register (AR[CSD]) are stored at that address + 8 ((GR r3 & ~0x8) + 8)."

9. Add Illegal Operation fault

1. Volume 3, Part I, Chapter 3:
   a. On page 3:247, add a new function to Table 3-1:
   Function: instruction_implemented (inst)
   Operation: Implementation-dependent routine which returns TRUE or FALSE, depending on whether inst is implemented.
   b. Remove the row "long_branch_implemented" from Table 3-1.
2. Volume 3, `ld` I-page on page 3:131
   a. Add the following paragraph to the end of the Description section:
   "For the sixteen_byte_form, an Illegal Operation fault is raised on processor models that do not support the instruction. CPUID register 4 indicates the presence of the feature on the processor model. See "Processor Identification Registers" on page 1:29 for details."
   b. Operation section, after:
   ```
   if (size == 16) itype |= UNCACHE_OPT;
   ```
   add:
   ```
   if (sixteen_byte_form && !instruction_implemented(LD16))
       illegal_operation_fault();
   ```
3. Volume 3, `st` I-page on page 3:219
   a. Add the following paragraph to the end of the Description section:
   "For the sixteen_byte_form, an Illegal Operation fault is raised on processor models that do not support the instruction. CPUID register 4 indicates the presence of the feature on the processor model. See "Processor Identification Registers" on page 1:29 for details."
   b. Operation section, after:
   ```
   otype = (sttype == 'rel') ? RELEASE : UNORDERED;
   ```
   add:
   ```
   if (sixteen_byte_form && !instruction_implemented(ST16))
       illegal_operation_fault();
   ```

4. Volume 3, `cmpxchg` I-page on page 3:41

   a. Add the following paragraph to the end of the Description section:

   "For `cmp8xchg16`, an Illegal Operation fault is raised on processor models that do not support the instruction. CPUID register 4 indicates the presence of the feature on the processor model. See "Processor Identification Registers" on page 1:29 for details."

   b. Operation section, after:

```
if(PR[qp]) {
```

   add:

```
if (sixteen_byte_form &&
!instruction_implemented(CMP8XCHG16))
     illegal_operation_fault();
```

5. Volume 3, `brl` I-page on 3:26, Operation section:

   — Replace the following:

```
if (!long_branch_implemented())
     illegal_operation_fault();
```

   with:

```
if (!instruction_implemented(BRL))
     illegal_operation_fault();
```

10. Non-speculative reference for WBL attribute clarification

1. Volume 2, Part I, add a new Section 4.4.6.1, at the end of Section 4.4.6:

4.4.6.1 Limited Speculation and the WBL Physical Addressing Attribute

Processors are allowed to reference limited speculation pages (WBL pages) speculatively, in order to increase performance, but this speculation is limited to prevent speculative references to 4Kbyte physical pages for which there is no actual memory (which would cause spurious machine checks).

Processors must not make hardware-generated speculative references to a given WBL 4Kbyte page until a **verified reference** has been made. Processors may optionally implement storage to hold the addresses of WBL 4Kbyte pages for which verified references have been made and may make subsequent hardware-generated speculative references to these pages. Such pages are termed **verified pages**.

A verified reference is an instruction or data reference made to the page by an in-order execution of the program; that is, a reference which would have been made had the instructions from the program been fetched and executed one at a time. A hardware-generated speculative reference does not constitute a verified reference. Hardware-generated speculative references include:

- Instruction fetches when the processor has not yet determined whether prior branches were predicted correctly.
- Instruction fetches when the processor has not yet determined whether prior instructions will raise faults or traps.
- Data references by instructions when the processor has not yet determined whether prior branches were predicted correctly.
- Data references by instructions when the processor has not yet determined whether prior instructions will raise faults or traps.
- Hardware-generated instruction prefetch references.
- Hardware-generated data prefetch references.

- Eager RSE data references.

For an instruction fetch to constitute a verified reference, it must only be determined that an in-order execution of the program requires that the IP point to this address, independent of whether the instruction at this address will subsequently take a fault or interrupt.

For a data reference to constitute a verified reference, the instruction must meet one of the following requirements:

- It executes without any fault or interrupt
- It takes an Unaligned Data Reference fault
- It takes a Data Debug fault
- It takes an External interrupt, but if it had not taken an External interrupt, it would have met one of the above qualifications (execute without fault, take an Unaligned Data Reference fault, or take a Data Debug fault)

Data-speculative loads are treated the same as normal loads, and if an in-order execution of the program requires the execution of a data speculative load, it constitutes a verified reference. Control-speculative loads to limited-speculation pages always defer and thus never constitute verified references.

It is not necessary for a processor to determine whether a reference will complete without generating a machine check for it to be a verified reference. If software actually references a physical address which will cause a machine check, hardware may generate multiple speculative references to the same page, potentially causing multiple machine checks.

Processors may access verified pages normally, as they would WB pages, including the use of caching, pipelining, and hardware-generate speculative references to improve performance.

Calling the PAL_PREFETCH_VISIBILITY procedure forces the processor to clear the storage holding the addresses of verified pages.

2. Remove the two paragraphs from Volume 2, Part I, Section 4.4.6 that talk about limited speculation (the paragraphs beginning, “Limited speculation is used to improve performance...”, and “Unless a limited-speculation page is speculatively accessible,...”).
3. In footnote “d” in Table 4-12 on page 2:68, change this text from:

   “The processor may only issue hardware-generated speculative references to a 4K-byte physical page while the page is speculatively accessible.”

   to:

   “The processor may only issue hardware-generated speculative references to a 4K-byte physical page if it is a verified page.”
4. On page 2:76, Volume 2, Part I, Section 4.4.11.2, change these two paragraphs from:

   “When a non-speculative reference is made to a physical address with the WBL attribute, the 4K page containing that address becomes speculatively accessible. This allows the processor that made the non-speculative reference to subsequently make speculative references to this page. (See the description of limited speculation in Section 4.4.6, “Speculation Attributes” on page 2:70.)

   If the same physical memory is then to be accessed with the U attribute, software must first make all such addresses speculatively inaccessible and flush any cached copies from the cache. Otherwise, an uncacheable reference may hit in cache, causing a Machine Check abort.”

   to:

   “When a verified reference is made to a physical address with the WBL attribute, the 4K page containing that address becomes speculatively accessible. This allows the processor that made the verified reference to subsequently make speculative references to this page.

(See the description of limited speculation in Section 4.4.6.1, “Limited Speculation and the WBL Physical Addressing Attribute” on page 2:70.)

If the same physical memory is then to be accessed with the UC attribute, software must first cause all such 4K pages to no longer be verified pages and flush any cached copies from the cache. Otherwise, an uncacheable reference may hit in cache, causing a Machine Check abort.”

5. Volume 2, Part I, Section 4.4.11.2, bullet point 1 on page 2:76, change this paragraph from:

   “Call PAL_PREFETCH_VISIBILITY with the input argument trans_type equal to one to indicate that the transition is for physical memory attributes. This PAL call terminates the processor's rights to make speculative references to any limited speculation pages (i.e., it makes all WBL pages speculatively inaccessible - see the discussion on limited speculation in Section 4.4.6.)”

   to:

   “Call PAL_PREFETCH_VISIBILITY with the input argument trans_type equal to one to indicate that the transition is for physical memory attributes. This PAL call terminates the processor's rights to make speculative references to any limited speculation pages (i.e., it causes all WBL pages to no longer be verified pages - see the discussion on limited speculation in Section 4.4.6.1.)”

6. Volume 2, Part I, Section 4.4.11.2 on page 2:77, the very last paragraph of this section is not part of bullet point 5, but rather a summation of the bulleted sequence.

7. In Volume 2, Part I, Section 11.9.3 PAL Procedure Specification, PAL_PREFETCH_VISIBILITY (Page 2:358) Description, paragraph 4, last sentence should be changed from:

   “For the processor to make any speculative reference to a limited speculation page after this call, there must be a non-speculative reference made to that page after this call.”

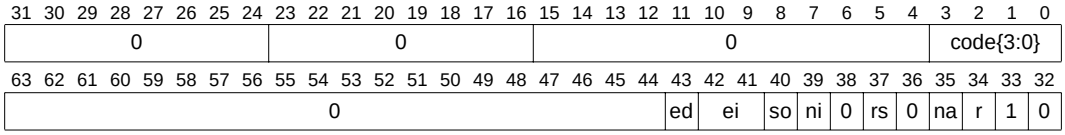
   to:

   “For the processor to make any speculative reference to a limited speculation page after this call, there must be a verified reference made to that page after this call. See the discussion on limited speculation in Section 4.4.6.1."

11. Dirty-bit fault ISR.code clarification

1. Volume 2, Part I, Section 8.3, Dirty-bit vector on page 2:160:

   a. Update diagram for ISR field to indicate that bits[3:0] represent ISR.code as shown below:



   b. Change following ISR statement on same page from:

      “ISR - The value for the ISR bits depend upon the type of access performed and are specified below. For mandatory RSE spill references, ISR.ed is always 0. For IA-32 memory references, ISR.ed, ei, ni, and rs are 0."

      to:

      “ISR - The value for the ISR bits depend upon the type of access performed and are specified below. For mandatory RSE spill references, ISR.ed is always 0. For IA-32 memory references, ISR.ed, ei, ni, and rs are 0. If the interruption was due to a non-access operation then the ISR.code bits {3:0} are set to indicate the type of the non-access instruction; otherwise they are set to 0."

c. Change following statement after the Notes section on the same page from:

"For probe.w.fault and probe.rw.fault the ISR.na bit is set"

to:

"For probe.w.fault and probe.rw.fault the ISR.na bit is set, and the ISR.code field is written with a value of 5."

12. FC data dependency ordering clarification

1. Volume 2, Part I, Section 4.4.7 on page 2:72, change the following sentence from:

"The flush cache instruction (`fc`, `fc.i`) instruction follows data dependency ordering. `fc` and `fc.i` are ordered with respect to previous and subsequent load, store, or semaphore instructions to the same line, regardless of the specified memory attribute."

to:

"The flush cache instruction (`fc`, `fc.i`) instruction follows data dependency ordering. `fc` and `fc.i` are ordered only with respect to previous load, store, or semaphore instructions to the same line, regardless of the specified memory attribute. Subsequent memory operations to the same line need not wait for prior `fc` or `fc.i` completion before being globally visible."

2. Volume 3, `fc` I-page (page 3:55), 5th paragraph, change the following sentence from:

"These instructions follow data dependency rules; they are ordered with respect to preceding and following memory references to the same line. `fc` and `fc.i` have data dependencies in the sense that any prior stores by this processor will be included in the flush operation."

to:

"These instructions follow data dependency ordering rules; they are ordered only with respect to previous load, store, or semaphore instructions to the same line. `fc` and `fc.i` have data dependencies in the sense that any prior stores by this processor will be included in the flush operation. Subsequent memory operations to the same line need not wait for prior `fc` or `fc.i` completion before being globally visible."

13. PAL_MC_DRAIN clarification

1. Volume 2, Part I, PAL_MC_DRAIN on page 2:339:

— Change the first sentence of the Description section from:

"This call causes all outstanding transactions in the processor to be completed (for example, loads get their data returned, store updates are completed, and prefetches are either completed or cancelled)."

to:

"This call causes all outstanding transactions in the processor to be completed. For example:

i. Flushes (`fc`) invalidate the cache; lines that have been modified are written back (issued to the fabric) to memory before invalidation.

ii. Instruction cache coherence flushes (`fc.i`) invalidate lines and/or write them back to main memory, if this is required to make the instruction caches coherent with the data caches.

iii. Loads get their data returned.

iv. Stores either update the cache or issue transactions to the system fabric.

v. Prefetches are either completed or cancelled."

14. Add hint instructions to support table

1. Volume 3, page 3:356, Table 5-5:
   — Add the following to 'pr-readers-br' (in the appropriate alphabetical location):
   “hint.b”
2. Volume 3, page 3:357, Table 5-5:
   — Add the following to 'pr-readers-nobr-nomovpr' (in the appropriate alphabetical location):
   “hint.f, hint.i, hint.m, hint.x”

15. Clarify speculative operation fault handler requirements

1. On the Speculation vector (0x5700) page in Section 8.3 of Volume 2, Part I, page 2:174 add a “Notes” section below the ISR diagram that reads:

   “The Speculative Operation fault handler is required to perform the following steps:

   1. Read the predicates and the IIM, IIP, IPSR, and ISR control registers into scratch bank-0 general registers.
   2. Copy the IIP value to IIPA.
   3. Sign-extend the IIM value (from 21 bits to 64), shift it left by 4 bits, and add it to the IIP value.
   4. Set the IPSR.ri field to 0.
   5. Check whether either IPSR.tb (Taken Branch trap) or IPSR.ss (Single Step enable) is 1. If not, emulation is complete, so restore the predicates and rfi. If so, then the check instruction would have taken one of these traps instead of branching to its target, so this handler needs to branch directly to the appropriate trap handler instead of performing the `rfi` (see steps 6 - 7).
   6. If IPSR.tb was 1, then update ISR.code with its “tb” bit set to 1 and its “ss” bit also set to 1 if IPSR.ss was 1 and all other bits 0. Restore the predicates, execute a srlz.d, and branch to the taken branch vector (IVT offset 0x5f00).
   7. If IPSR.ss was 1 (but not IPSR.tb), then update ISR.code with its “ss” bit set to 1, and all other bits 0. Restore the predicates, execute a srlz.d, and branch to the single step vector (IVT offset 0x6000).”

2. In Table 5-7 “Interruption Vector Table (IVT)”, change the “Reserved” text in the Vector Name column of the offset 0x5800 row to “Reserved for software use” and attach a footnote to this entry. The text of the footnote should read:

   “Unlike the other Reserved IVT vectors, which may be defined in future revisions of the architecture, vector 0x5800 is permanently reserved. Software may use this vector for any purpose, such as placing in this area portions of other handlers that don't fit into their assigned vector.”

3. Add the following to the Speculation vector (0x5700) page, just below the list added by (A):

   “The Speculative Operation fault handler does not need to check for unimplemented instruction addresses. They will be checked automatically by processor hardware when the handler executes its `rfi`. If an emulated check instruction targets an unimplemented address and also needs to take a Single Step trap or Taken Branch trap (or both), the Unimplemented Instruction Address trap will not be raised until after the Single Step and/or Taken Branch trap has been handled, making it appear that the Unimplemented Instruction Address trap has the wrong priority. A Speculative Operation fault handler with this behavior is architecturally compliant.”

4. In Table 5-6 “Interruption Priorities” on page 2:94, add a footnote to the “Unimplemented Instruction Address trap” cell, which reads:

   “Unimplemented Instruction Address traps on emulated check instructions have a lower priority than Taken Branch trap and Single Step trap. See Speculation vector (0x5700) on page 2:174.”

Documentation Changes

1. Update IA-32 CPUID I-Page

Volume 3: Updated IA-32 CPUID Instruction

CPUID—CPU Identification

Opcode	Instruction	Description
0F A2	CPUID	Returns processor identification and feature information in the EAX, EBX, ECX, and EDX registers, according to the input value entered initially in the EAX register.

Description

Returns processor identification and feature information in the EAX, EBX, ECX, and EDX registers. The information returned is selected by entering a value in the EAX register before the instruction is executed. Table 2-4 shows the information returned, depending on the initial value loaded into the EAX register.

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction.

The information returned with the CPUID instruction is divided into two groups: basic information and extended function information. Basic information is returned by entering an input value starting at 0 in the EAX register; extended function information is returned by entering an input value starting at 80000000H. When the input value in the EAX register is 0, the processor returns the highest value the CPUID instruction recognizes in the EAX register for returning basic information. Always use an EAX parameter value that is equal to or greater than zero and less than or equal to this highest EAX return value for basic information. When the input value in the EAX register is 80000000H, the processor returns the highest value the CPUID instruction recognizes in the EAX register for returning extended function information. Always use an EAX parameter value that is equal to or greater than zero and less than or equal to this highest EAX return value for extended function information.

The CPUID instruction can be executed at any privilege level to serialize instruction execution. Serializing instruction execution guarantees that any modifications to flags, registers, and memory for previous instructions are completed before the next instruction is fetched and executed.

Table 2-4. Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0	EAX	Maximum CPUID Input Value
	EBX	756E6547H “Genu” (G in BL)
	ECX	6C65746EH “ntel” (n in CL)
	EDX	49656E69H “inel” (i in DL)

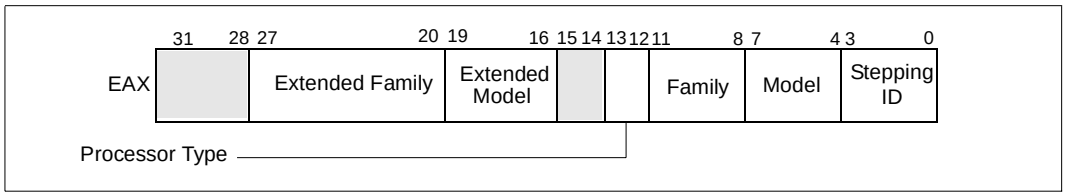
Table 2-4. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
1H	EAX	Version Information (Type, Family, Model, and Stepping ID)
	EBX	Bits 7-0: Brand Index[a] Bits 15-8: CLFLUSH line size (Value * 8 = cache line size in bytes) Bits 23-16: Number of logical processors per physical processor Bits 31-24: Local APIC ID[b]
	ECX	Reserved
	EDX	Feature Information (see Table 2-5)
2H	EAX	Cache and TLB Information
	EBX	Cache and TLB Information
	ECX	Cache and TLB Information
	EDX	Cache and TLB Information
	Extended Function CPUID Information	
8000000H	EAX	Maximum Input Value for Extended Function CPUID Information
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
8000001H	EAX	Extended Processor Signature and Extended Feature Bits. (Currently reserved.)
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
8000002H	EAX	Processor Brand String
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
8000003H	EAX	Processor Brand String Continued
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued

a. This field is not supported for processors based on Itanium architecture, zero (unsupported encoding) is returned.
b. This field is invalid for processors based on Itanium architecture, reserved value is returned.

When the input value is 1, the processor returns version information in the EAX register (see Figure 2-3). The version information consists of an Intel architecture family identifier, a model identifier, a stepping ID, and a processor type.

Figure 2-3. Version Information in Registers EAX



If the values in the family and/or model fields reach or exceed FH, the CPUID instruction will generate two additional fields in the EAX register: the extended family field and the extended model field. Here, a value of FH in either the model field or the family field indicates that the extended model or family field, respectively, is valid. Family and model numbers beyond FH range from 0FH to FFH, with the least significant hexadecimal digit always FH.

See AP-485, *Intel® Processor Identification and the CPUID Instruction* (Order Number 241618) for more information on identifying Intel architecture processors.

When the input value in EAX is 1, three unrelated pieces of information are returned to the EBX register:

- Brand index (low byte of EBX) – this number provides an entry into a brand string table that contains brand strings for IA-32 processors. Please refer to AP-485, *Intel® Processor Identification and the CPUID Instruction* (Order Number 241618) for information on brand indices.

  Note: The Brand index field is not supported for processors based on Itanium architecture, zero (unsupported encoding) is returned.

- CLFLUSH instruction cache line size (second byte of EBX) – this number indicates the size of the cache line flushed with CLFLUSH instruction in 8-byte increments. This field is valid only when the CLFSH feature flag is set.

- Local APIC ID (high byte of EBX) – this number is the 8-bit ID that is assigned to the local APIC on the processor during power up.

  Note: The local APIC ID field is invalid for processors based on the Itanium architecture, reserved value is returned. Software should check the feature flags to make sure they are not running on processors based on the Itanium architecture before interpreting the return value in this field.

When the EAX register contains a value of 1, the CPUID instruction (in addition to loading the processor signature in the EAX register) loads the EDX register with the feature flags. The feature flags (when a Flag = 1) indicate what features the processor supports. Table 2-5 lists the currently defined feature flag values.

A feature flag set to 1 indicates the corresponding feature is supported. Software should identify Intel as the vendor to properly interpret the feature flags.

Table 2-5. Feature Flags Returned in EDX Register

Bit	Mnemonic	Description
0	FPU	**Floating Point Unit On-Chip.** The processor contains an x87 FPU.
1	VME	**Virtual 8086 Mode Enhancements.** Virtual 8086 mode enhancements, including CR4.VME for controlling the feature, CR4.PVI for protected mode virtual interrupts, software interrupt indirection, expansion of the TSS with the software indirection bitmap, and EFLAGS.VIF and EFLAGS.VIP flags.
2	DE	**Debugging Extensions.** Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.
3	PSE	**Page Size Extension.** Large pages of size 4Mbyte are supported, including CR4.PSE for controlling the feature, the defined dirty bit in PDE (Page Directory Entries), optional reserved bit trapping in CR3, PDEs, and PTEs.
4	TSC	**Time Stamp Counter.** The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	**Model Specific Registers RDMSR and WRMSR Instructions.** The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.
6	PAE	**Physical Address Extension.** Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2 Mbyte pages are supported instead of 4 Mbyte pages if PAE bit is 1. The actual number of address bits beyond 32 is not defined, and is implementation specific.

Table 2-5. Feature Flags Returned in EDX Register (Continued)

Bit	Mnemonic	Description
7	MCE	**Machine Check Exception.** Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model-specific processing of the exception, or test for the presence of the Machine Check feature.
8	CX8	**CMPXCHG8B Instruction.** The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).
9	APIC	**APIC On-Chip.** The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default – some processors permit the APIC to be relocated).
10	Reserved	Reserved.
11	SEP	**SYSENTER and SYSEXIT Instructions.** The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	**Memory Type Range Registers.** MTRRs are supported. The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.
13	PGE	**PTE Global Bit.** The global bit in page directory entries (PDEs) and page table entries (PTEs) is supported, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.
14	MCA	**Machine Check Architecture.** The Machine Check Architecture, which provides a compatible mechanism for error reporting is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.
15	CMOV	**Conditional Move Instructions.** The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported.
16	PAT	**Page Attribute Table.** Page Attribute Table is supported. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory on a 4K granularity through a linear address.
17	PSE-36	**32-Bit Page Size Extension.** Extended 4-MByte pages that are capable of addressing physical memory beyond 4 GBytes are supported. This feature indicates that the upper four bits of the physical address of the 4-MByte page is encoded by bits 13-16 of the page directory entry.
18	PSN	**Processor Serial Number.** The processor supports the 96-bit processor identification number feature and the feature is enabled.
19	CLFSH	**CLFLUSH Instruction.** CLFLUSH Instruction is supported.
20	Reserved	Reserved.
21	DS	**Debug Store.** The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities.
22	ACPI	**Thermal Monitor and Software Controlled Clock Facilities.** The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	**Intel MMX Technology.** The processor supports the Intel MMX technology.

Table 2-5. Feature Flags Returned in EDX Register (Continued)

Bit	Mnemonic	Description
24	FXSR	**FXSAVE and FXRSTOR Instructions.** The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions
25	SSE	**SSE.** The processor supports the SSE extensions.
26	SSE2	**SSE2.** The processor supports the SSE2 extensions.
27	SS	**Self Snoop.** The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	**Hyper-Threading Technology.** The processor implements Hyper-Threading technology.
29	TM	**Thermal Monitor.** The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Processor based on the Intel Itanium architecture	The processor is based on the Intel Itanium architecture and is capable of executing the Intel Itanium instruction set. IA-32 application level software MUST also check with the running operating system to see if the system can also support Itanium architecture-based code before switching to the Intel Itanium instruction set.
31	PBE	**Pending Break Enable.** The processor supports the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt. Bit 10 (PBE enable) in the IA32_MISC_ENABLE MSR enables this capability.

When the input value is 2, the processor returns information about the processor's internal caches and TLBs in the EAX, EBX, ECX, and EDX registers. The encoding of these registers is as follows:

- The least-significant byte in register EAX (register AL) indicates the number of times the CPUID instruction must be executed with an input value of 2 to get a complete description of the processor's caches and TLBs.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1 byte descriptors.

Please see the processor-specific supplement for further information on how to decode the return values for the processors internal caches and TLBs.

CPUID performs instruction serialization and a memory fence operation.

CPUID—CPU Identification (Continued)

Operation

```
CASE (EAX) OF
   EAX = 0H:
        EAX ← Highest input value understood by CPUID;
        EBX ← Vendor identification string;
        EDX ← Vendor identification string;
        ECX ← Vendor identification string;
   BREAK;
   EAX = 1H:
        EAX[3:0] ← Stepping ID;
        EAX[7:4] ← Model;
        EAX[11:8] ← Family;
        EAX[13:12] ← Processor Type;
        EAX[15:14] ← Reserved;
        EAX[19:16] ← Extended Model;
        EAX[27:20] ← Extended Family;
        EAX[31:28] ← Reserved;
        EBX[7:0] ← Brand Index; (* Always zero for processors based on Itanium
architecture *)
        EBX[15:8] ← CLFLUSH Line Size;
        EBX[16:23] ← Number of logical processors per physical processor;
        EBX[31:24] ← Initial APIC ID; (* Reserved for processors based on Itanium
architecture *)
        ECX ← Reserved;
        EDX ← Feature flags;
   BREAK;
   EAX = 2H:
        EAX ← Cache and TLB information;
        EBX ← Cache and TLB information;
        ECX ← Cache and TLB information;
        EDX ← Cache and TLB information;
   BREAK;
   EAX = 80000000H:
        EAX ← Highest extended function input value understood by CPUID;
        EBX ← Reserved;
        ECX ← Reserved;
        EDX ← Reserved;
   BREAK;
   EAX = 80000001H:
        EAX ← Extended Processor Signature and Feature Bits; (* Currently Reserved *)
        EBX ← Reserved;
        ECX ← Reserved;
        EDX ← Reserved;
   BREAK;
   EAX = 80000002H:
        EAX ← Processor Name;
        EBX ← Processor Name;
        ECX ← Processor Name;
        EDX ← Processor Name;
   BREAK;
   EAX = 80000003H:
        EAX ← Processor Name;
        EBX ← Processor Name;
        ECX ← Processor Name;
        EDX ← Processor Name;
   BREAK;
   EAX = 80000004H:
        EAX ← Processor Name;
        EBX ← Processor Name;
```

```
            ECX ← Processor Name;
            EDX ← Processor Name;
        BREAK;
        DEFAULT: (* EAX > highest value recognized by CPUID *)
            EAX ← Reserved, Undefined;
            EBX ← Reserved, Undefined;
            ECX ← Reserved, Undefined;
            EDX ← Reserved, Undefined;
        BREAK;
    ESAC;

    memory_fence();
    instruction_serialize();
```

Flags Affected

None.

Additional Itanium System Environment Exceptions

Itanium Reg Faults NaT Register Consumption Abort.

Exceptions (All Operating Modes)

None.

Intel Architecture Compatibility

The CPUID instruction is not supported in early models of the Intel486 processor or in any Intel architecture processor earlier than the Intel486 processor. The ID flag in the EFLAGS register can be used to determine if this instruction is supported. If a procedure is able to set or clear this flag, the CPUID is supported by the processor running the procedure.

2. PAL_BUS_GET/SET_FEATURES fix

1. Volume 2, Part I, Table 11-25.
   - — On page 2:296, change the following bit 52 description from:

     “Enable a bus cache line replacement transaction when a cache line in the shared state is replaced from the highest level processor cache and is not present in the lower level processor caches. When 0, no bus cache line replacement transaction will be seen on the bus. When 1, bus cache line replacement transactions will be seen on the bus when the above condition is detected.”

     to:

     “Enable a bus cache line replacement transaction when a cache line in the shared or exclusive state is replaced from the highest level processor cache and is not present in the lower level processor caches. When 0, no bus cache line replacement transaction will be seen on the bus. When 1, bus cache line replacement transactions will be seen on the bus when the above condition is detected.”

3. PAL_COPY_PAL update

1. Volume 2, Part I, page 2:317:
   a. Change the following argument in the Arguments section of the PAL_COPY_PAL procedure from:

      “processor - Unsigned integer denoting whether the call is being made on the boot processor or an application processor.”

to:

"copy_option - Unsigned integer indicating whether relocatable PAL code and PAL PMI code should be copied from firmware address space to main memory."

b. Change the following sentences in the first paragraph of the Description section of PAL_COPY_PAL from:

"This procedure also updates the PALE_PMI entrypoint in hardware. If the call in made on a application processor the copy is not performed. The processor argument denotes whether the call is made on the boot processor (value of 0) or an application processor (value of 1)."

to:

"A value of 0 for the copy_option indicates that the relocation should be performed; a value of 1 indicates that the relocation should not be performed. This procedure also updates the PALE_PMI entrypoint in hardware."

4. Fixing X-Unit text correction

1. Volume 3, Section 4.7.4 "Nop/Hint (X-Unit)", Table 4-73 on page 3:332, change:

   "nop.m" to "nop.x"

   "hint.m" to "hint.x"

5. PAL_CACHE_SHARED_INFO text correction

1. Volume 2, Part I, page 2:311:

   For the entire PAL_CACHE_SHARED_INFO pages, change any instance of "proc_n_log_info" to "proc_n_cache_info".

6. PAL_CACHE_FLUSH clarification and minor code sequence fix

1. Volume 2, Part I, page 2:431, make a change to the assembly code in Section 5.1.1.3 (first line of assembly code). The code is trying to address region register 2, but indexed it incorrectly.

   Change from: `mov r2 = 2`

   to: `movl r2 = (2 << 61)`

7. PAL_GET_PROC_FEATURES table fix

1. Volume 2, Part I, PAL_GET_PROC_FEATURES table on page 2:361, fix bit 40.

   Change from:

   Bit: 40-0

   Class: N/A

   Control: N/A

   Description: reserved

   to:

   Bit: 40

   Class: N/A

   Control: N/A

   Description: reserved