IA-32 Intel® Architecture Software Developer's Manual

Documentation Changes

June 2003

Notice: The IA-32 Intel® Architecture 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 this specification update.

Document Number: 252046-004

Contents

Revision History

Version	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 Documenation Change #6 - References to bits Gen and Len Deleted Removed Documenation Change #4 - VIF Information Added to CLI Discussion	February 2003
-004	Removed Documentation changes 1-17 Added Documentation changes 1-24	June 2003

Preface

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

Affected Documents/Related Documents

Document Title	Document Number
IA-32 Intel® Architecture Software Developer's Manual: Volume 1, Basic Architecture	245470-011
IA-32 Intel® Architecture Software Developer's Manual: Volume 2, Instruction Set Reference	245471-011
IA-32 Intel® Architecture Software Developer's Manual: Volume 3, System Programming Guide	245472-011

Nomenclature

Documentation Changes include errors or omissions from the current published specifications. These changes will be incorporated in the next release of the Software Development Maunal.

Summary Table of Changes

The following table indicates documentation changes which apply to the IA-32 Intel Architecture. This table uses the following notations:

Codes Used in Summary Table

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

Summary Table of Documentation Changes

Number	DOCUMENTATION CHANGES
1.	Information About CR2 Contents During a Double Fault has Been Added
2.	Wording Changed to Clarify the Treatment of Hardware Interrupts in Regard to VIF Settings
3.	Flag Information has Been Added
4.	A PVI Reference has Been Added
5.	INT Added to a List of Instructions in a General Description
6.	INT 1 Documentation Altered
7.	Erroneous Conditional Phrase Removed From Some Virtual 8086 Mode Exception Descriptions
8.	Note Added to Clarify Aspects of the Pentium 4 Cache System
9.	A Clarification About the Handling of NMIs
10.	Documentation for Postcommit Task Switch Point Updated
11.	Table Captions Changed to Address Problems With a Usage Model
12.	Missing Footnote References on Table
13.	CMOVcc Table Corrected
14.	Introduction to Code Segment Clarified to Properly set Expectations
15.	Entry for CR4 Register Added to Table
16.	Operand Expression Corrected
17.	Information Removed as Irrelevant or Erroneous
18.	Cross Reference Updated
19.	PSE-36 Flag Description Corrected
20.	Description of the Time Stamp Counter has Been Updated
21.	Unclear Description Corrected

Summary Table of Documentation Changes

Number	DOCUMENTATION CHANGES
22.	PEXTRW Description Corrected
23.	Pentium M Processors and #BPs
24.	Error Correction in Appendix A, Volume 3

Documentation Changes

1. Information About CR2 Contents During a Double Fault has Been Added

In Volume 3, Chapter 5, Section: *Interrupt 14—Page-Fault Exception (#PF)* has been upated. A footnote with the following content has been added. The footnote talks about the updating of CR2 in a double fault context.

> Processors update CR2 whenever a page fault is detected. If a second page fault occurs while an earlier page fault is being delivered, the faulting linear address of the second fault will overwrite the contents of CR2 (replacing the previous address). These updates to CR2 occur even if the page fault results in a double fault or occurs during the delivery of a double fault.

2. Wording Changed to Clarify the Treatment of Hardware Interrupts in Regard to VIF Settings

In Volume 3, Section 16.4 has been updated. The area impacted is reproduced below in context; see the changebar and bold text. The old phrase was "If the processor receives a maskable hardware interrupt *when the VIF flag is clear*, the processor invokes the protected-mode interrupt handler."

> When the PVI flag is set to 1, the CPL is 3, and the IOPL is less than 3, the STI and CLI instructions set and clear the VIF flag in the EFLAGS register, leaving IF unaffected. In this mode of operation, an application running in protected mode and at a CPL of 3 can inhibit interrupts in the same manner as is described in Section 16.3.2., "Class 2—Maskable Hardware Interrupt Handling in Virtual-8086 Mode Using the Virtual Interrupt Mechanism", for a virtual-8086 mode task. When the application executes the CLI instruction, the processor clears the VIF flag. **If the processor receives a maskable hardware interrupt, the processor invokes the protected-mode interrupt handler**. This handler checks the state of the VIF flag in the EFLAGS register. If the VIF flag is clear (indicating that the active task does not want to have interrupts handled now), the handler sets the VIP flag in the EFLAGS image on the stack and returns to the privilege-level 3 application, which continues program execution...

3. Flag Information has Been Added

In Volume 3, Section 16.3.1.1, item 4; now specifies a more complete list of cleared flags. The numbered list is reproduced below, with the change.
----

1. Switches to 32-bit protected mode and privilege level 0.
2. Saves the state of the processor on the privilege-level 0 stack. The states of the EIP, CS, EFLAGS, ESP, SS, ES, DS, FS, and GS registers are saved (see Figure 16-4).
3. Clears the segment registers. Saving the DS, ES, FS, and GS registers on the stack and then clearing the registers lets the interrupt or exception handler safely save and restore these registers regardless of the type segment selectors they contain (protected-mode or 8086-style). The interrupt and exception handlers, which may be called in the context of either a protected-mode task or a virtual-8086-mode task, can use the same code sequences for saving and restoring the registers for any task. Clearing these registers before execution of the IRET instruction does not cause a trap in the interrupt handler. Interrupt procedures that expect values in the segment registers or that return values in the segment registers must use the register images saved on the stack for privilege level 0.
4. Clears **VM, NT, RF and TF** flags (in the EFLAGS register).
5. Begins executing the selected interrupt or exception handler.

4. A PVI Reference has Been Added

Volume 3, Section 5.8.1 (text in the fourth paragraph) has been upated to add a relevant PVI reference. The updated text is reproduced in context below.

> The IF flag can be set or cleared with the STI (set interrupt-enable flag) and CLI (clear interrupt-enable flag) instructions, respectively. These instructions may be executed only if the CPL is equal to or less than the IOPL. A general-protection exception (#GP) is generated if they are executed when the CPL is greater than the IOPL. (The effect of the IOPL on these instructions is modified slightly when the virtual mode extension is enabled by setting the VME flag in control register CR4: see Section 16.3., "Interrupt and Exception Handling in Virtual-8086 Mode". **Behavior is also impacted by the PVI flag: see Section 16.4., "Protected-Mode Virtual Interrupts"**.)

5. INT Added to a List of Instructions in a General Description

A paragraph in Volume 3, Section 16.4 (fifth paragraph) has been changed. The new text is shown below (INT was added to the list):

> PUSHF, POPF, IRET and INT are executed like in the Intel486 processor, regardless of whether protected-mode virtual interrupts are enabled.

6. INT 1 Documentation Altered

Volume 3, Table 5.1 has been upated to reflect Intel's current INT 1 usage model. See the table section below (the entire table is not included). The Intel position is that INT 1 is for internal use only.

Vector No.	Mne-monic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	**#DB**	**RESERVED**	**Fault/ Trap**	**No**	**For Intel use only.**
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.

Volume 3, Section 5.14: *Interrupt 1 - Debug Exception (#DB)* has also been updated for the same reason. The new text follows.

Interrupt 1—Debug Exception (#DB)

Reserved for Intel use only.

7. Erroneous Conditional Phrase Removed From Some Virtual 8086 Mode Exception Descriptions

Corrections have been made in Volume 2, Chapter 3 for CLTS, FXRSTOR, FXSAVE, HLT, MULSD, MULSS, RCPSS exception conditions. Various entries were referring to exceptions being raised if CPL was this or that. In the impacted exception conditions, CPL is always 3 (true in V86 mode) so the included 'if statement' was simply not necessary.

8. Note Added to Clarify Aspects of the Pentium 4 Cache System

In Volume 1, Section 2.1.7; a footnote has been added to provide more information about the Pentium 4 cache system. The applicable text is reproduced below.

The Intel Pentium 4 processor uses a cache line size of 64bytes throughout its cache hierarchy. The larger unified cache levels use a sectored implementation, where each 128byte cache sector consists of two associated 64byte cache lines.

9. A Clarification About the Handling of NMIs

In Volume 3, Section 13.7, the first paragraph has been updated to better describe the handling of NMIs. The new text is reproduced in context below.NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequenc. This assumes that NMIs were not blocked before the SMI occurred. If NMIs were blocked before the SMI occurred, they are blocked after execution of RSM.

10. Documentation for Postcommit task Switch Point Updated

In Volume 3, Section 6.3; the numbered list was impacted by improvements to task switch description. The impacted area is reproduced below.

----

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 registers, the EIP register, the general-purpose registers, and the segment selectors. Note that a fault during the load of this state may corrupt architectural state.**
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.**

NOTE

At this point, 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 5, "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.)**

----

11. Table Captions Changed to Address Problems With a Usage Model

Volume 2, Tables A-2 and A-3 have been updated to address a usage model complaint in regard to the way table cells were addressed. New footnotes (the text is provided below) document the A-2, A-3 table access methodology.

----

† All blanks in the opcode map shown in Table A-2 are reserved and should not be used. Do not depend on the operation of these undefined opcodes.

†† To use the table, take the opcode's first Hex character from the row designation and the second character from the column designation. For example: 07H for [POP ES].

12. Missing Footnote References on Table

n Volume 3, Table 10-2; the table footnotes were confusing. In the new version of the document, the need for table footnotes has been eliminated by moving the applicable data into the table. The refreshed table cells are shown.

.

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Strong Uncacheable (UC)	No	No	No	Strong Ordering
Uncacheable (UC-)	No	No	No	Strong Ordering. Can only be selected through the PAT. Can be overridden by WC in MTRRs.
Write Combining (WC)	No	No	Yes	Weak Ordering. Available by programming MTRRs or by selecting it through the PAT.
Write Through (WT)	Yes	No	Yes	Speculative Processor Ordering
Write Back (WB)	Yes	Yes	Yes	Speculative Processor Ordering.
Write Protected (WP)	Yes for reads; no for writes	No	Yes	Speculative Processor Ordering. Available by programming MTRRs.

13. CMOVcc Table Corrected

In Volume 2, Chapter 3, Section: CMOVcc—Conditional Move; the descriptions of ‘CMOVO r16, r/m16’ and ‘CMOVO r32, r/m32’ have been updated. The impacted information is reproduced in context below. Both of the old descriptions erroneously stated OF = 0.

0F q5 */r*	CMOVNZ *r16, r/m16*	Move if not zero (ZF=0)
0F 45 */r*	CMOVNZ *r32, r/m32*	Move if not zero (ZF=0)
0F 40 */r*	CMOVO *r16, r/m16*	**Move if overflow (OF=1)**
0F 40 */r*	CMOVO *r32, r/m32*	**Move if overflow (OF=1)**
0F 4A */r*	CMOVP *r16, r/m16*	Move if parity (PF=1)
0F 4A */r*	CMOVP *r32, r/m32*	Move if parity (PF=1)
0F 4A */r*	CMOVPE *r16, r/m16*	Move if parity even (PF=1)

14. Introduction to Code Segment Clarified to Properly set Expectations

For Volume 3, Example 9-1; the introduction has been updated. Pseudocode supplied in the IA-32 PRM is general in nature. It provides high level examples. To obtain working code, most IA-32 PRM examples must be updated to fit into specific contexts. The old introduction to Example 9-1 seemed to imply that the pseudocode supplied would work in any situation. The old language has been removed; it has been replaced by the following statement:

> Example 9-1 provides high-level sample code designed to move the processor into protected mode. This listing does not include any opcode and offset information.

15. Entry for CR4 Register Added to Table

In Volume 3, Table 13-2; an entry for the CR4 register has been added. The applicable table cells are reproduced in context below.

DS, ES, FS, GS, SS Limits	0FFFFFFFFH
CR0	PE, EM, TS and PG flags set to 0; others unmodified
CR4	**Cleared to zero**
DR6	Undefined

16. Operand Expression Corrected

In Volume 2, Section 3-11; the operand order has been corrected for MOVSD (F2). The applicable table segment is shown below with the relevant expression highlighted.

.

1	MOVUPS Vps, Wps MOVSS (F3) Vss, Wss MOVUPD (66) Vpd, Wpd MOVSD (F2) Vsd, Wsd	MOVUPS Wps, Vps MOVSS (F3) Wss, Vss MOVUPD (66) Wpd, Vpd MOVSD (F2) **Wsd, Vsd**	MOVLPS Wq, Vq MOVLPD (66) Vq, Ws MOVHLPS Vq, Vq	MOVLPS Vq, Wq MOVLPD (66) Vq, Wq	UNPCKLPS Vps, Wq UNPCKLPD (66) Vpd, Wq	UNPCKHPS Vps, Wq UNPCKHPD (66) Vpd, Wq	MOVHPS Vq, Wq MOVHPD (66) Vq, Wq MOVLHPS Vq, Vq	MOVHPS Wq, Vq MOVHPD (66) Wq, Vq

17. Information Removed as Irrelevant or Erroneous

In Volume 3, Section 3.11; the last bulleted item in a bulleted list has been removed (as unnecessary information). Section 3.11 now closes with the following text.

When the processor loads a page-directory or page-table entry for a global page into a TLB, the entry will remain in the TLB indefinitely. The only ways to deterministically invalidate global page entries are as follows:

- Clear the PGE flag and then invalidate the TLBs.
- Execute the INVLPG instruction to invalidate individual page-directory or page-table entries in the TLBs.

For additional information about invalidation of the TLBs, see Section 10.9., *Invalidating the Translation Lookaside Buffers (TLBs)*.

18. Cross Reference Updated

In Volume 3, Section 8.11.1; an erroneous cross reference has been updated. The relevant paragraph is reproduced below (it's item 4 in the numbered list). See the change bar and the highlighted text.

Destination Mode (DM): This bit indicates whether the Destination ID field should be interpreted as logical or physical APIC ID for delivery of the lowest priority interrupt. If RH is 1 and DM is 0, the Destination ID field is in physical destination mode and only the processor in the system that has the matching APIC ID is considered for delivery of that interrupt (this means no re-direction). If RH is 1 and DM is 1, the Destination ID Field is interpreted as in logical destination mode and the redirection is limited to only those processors that are part of the logical group of processors based on the processor's logical APIC ID and the Destination ID field in the message. The logical group of processors consists of those identified by matching the 8-bit Destination ID with the logical destination identified by the Destination Format Register and the Logical Destination Register in each local APIC. The details are similar to those described in **Section 8.6.2., "Determining IPI Destination"**. If RH is 0, then the DM bit is ignored and the message is sent ahead independent of whether the physical or logical destination mode is used.

19. PSE-36 flag Description Corrected

In Volume 2, Table 3-11; an error has been corrected. The old table entry assigned a 32-bit page size extension to PSE-36. The corrected entry is reproduced in context below .

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	**36-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.

20. Description of the Time Stamp Counter has Been Updated

In Volume 3, Section 15.7: the third paragraph has been expanded. The updated version of the paragraph is reproduced below. See the changebar and the bold text.

The time-stamp counter (as implemented in the Pentium 4, Intel Xeon, P6 family, and Pentium processors) is a 64-bit counter that is set to 0 following the hardware reset of the processor. Following reset, the counter is incremented every processor clock cycle, even when the processor is halted by the HLT instruction or the external STPCLK# pin. **However, the assertion of the external DPSLP# pin may cause the time-stamp counter to stop and Intel SpeedStep® technology transitions may cause the frequency at which the time-stamp counter increments to change in accordance with the processor's internal clock frequency**.

21. Unclear Description Corrected

In Volume 3, Section 8.11.1, item number 4 in the numbered list has been re-written to address a clarity issue. The next text is indicated by the changebar and bold text (the old version implied the possibility of re-direction).

4. Destination Mode (DM): This bit indicates whether the Destination ID field should be interpreted as logical or physical APIC ID for delivery of the lowest priority interrupt. **If RH is 1 and DM is 0, the Destination ID field is in physical destination mode and only the processor in the system that has the matching APIC ID is considered for delivery of that interrupt (this means no re-direction).** If RH is 1 and DM is 1, the Destination ID Field is interpreted as in logical destination mode and the redirection is limited to only those processors that are part of the logical group of processors based on the processor's logical APIC ID and the Destination ID field in the message. The logical group of processors consists of those identified by matching the 8-bit Destination ID with the logical destination identified by the Destination Format Register and the Logical Destination Register in each local APIC. The details are similar to those described in Section 8.6.2., "Determining IPI Destination". If RH is 0, then the DM bit is ignored and the message is sent ahead independent of whether the physical or logical destination mode is used.

22. PEXTRW Description Corrected

In Volume 2, Chapter 3, Section: *PEXTRW—Extract Word*; the **Description** paragraph has been updated. The changebar and the bold text mark the spot ('3 least-significant bits' replaces '4 least-significant bits').

Copies the word in the source operand (second operand) specified by the count operand (third operand) to the destination operand (first operand). The source operand can be an MMX technology or an XMM register. The destination operand is the low word of a general-purpose register. The count operand is an 8-bit immediate. When specifying a word location in an MMX technology register, the 2 least-significant bits of the count operand specify the location; for an XMM register, **the 3 least-significant bits specify the location**. The high word of the destination operand is cleared (set to all 0s).

23. Pentium M Processors and #BPs

In Volume 3, Section 15.3.2; information about the Pentium M processor's handling of #BPs has been added. The section is reproduced below; the changebar and bold text mark the spot.

> The breakpoint exception (interrupt 3) is caused by execution of an INT 3 instruction (see Chapter 5, "Interrupt 3—Breakpoint Exception (#BP)"). Debuggers use break exceptions in the same way that they use the breakpoint registers; that is, as a mechanism for suspending program execution to examine registers and memory locations. With earlier IA-32 processors, breakpoint exceptions are used extensively for setting instruction breakpoints.
>
> With the Intel386 and later IA-32 processors, it is more convenient to set breakpoints with the breakpoint-address registers (DR0 through DR3). However, the breakpoint exception still is useful for breakpointing debuggers, because the breakpoint exception can call a separate exception handler. The breakpoint exception is also useful when it is necessary to set more breakpoints than there are debug registers or when breakpoints are being placed in the source code of a program under development.
>
> **Note that with Pentium M processors, #BPs for fast string operations are reported only on cache line boundaries.**

24. Error Correction in Appendix A, Volume 3

In Volume 3, Table A-1; incorrect parameter information has been corrected. For the corrected information, see the bold text in the table segment below. The incorrect entry was 05H.

global_power _events			This event accumulates the time during which a processor is not stopped.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	**013H**	ESCR[31:25]
	ESCR Event Mask	Bit 0: Running	ESCR[24:9] The processor is active (includes the handling of HLT STPCLK and throttling.