Pentium® Processor for Embedded Applications

Specification Update

January 1999

Notice: The Pentium® processor 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 this Specification Update.

Order Number: 273183-003

Contents

Revision History

Revision History

Revision Date	Version	Description
1/11/99	003	Added Specification Updates for embedded Pentium processor with MMX technology devices.
12/11/98	002	Added: Specification Change 20; Errata 83; and Specification Clarification 25. Updated the Processor Identification section to reflect new SL2TU S-spec number.
6/11/98	001	This is the first publication of this document.

Preface

This document is an update to the specifications contained in the *Pentium® Processor Family Developer's Manual*, (order number 273204), the *Intel Architecture Software Developer's Manual*, Volume 1, 2 and 3 (order numbers 243190, 243191, and 243192); and the *Embedded Pentium® Processor* (order number 273202), *Embedded Pentium® Processor with Voltage Reduction Technology* (order number 273203), *Embedded Pentium® Processor with MMX™ Technology* (order number 273214) and *Low-Power Embedded Pentium® Processor with MMX™ Technology* (order number 273184) datasheets. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains Specification Changes, S-Specs, Errata, Specification Clarifications and Documentation Changes for Pentium processors for high-performance embedded applications. The following processors are included in this Specification Update:

- 100 MHz Pentium® Processor
- 133 MHz Pentium® Processor
- 133 MHz Pentium® Processor with Voltage Reduction Technology
- 166 MHz Pentium® Processor
- 200 MHz Pentium Processor with MMX Technology
- 233 MHz Pentium Processor with MMX Technology
- 166 MHz Low-Power Pentium Processor with MMX Technology
- 266 MHz Low-Power Pentium Processor with MMX Technology

For information pertaining to processors not listed above, refer to the *Pentium® Processor Specification Update* (order number 242480).

Nomenclature

Specification Changes are modifications to the current published specifications. These changes will be incorporated in the next release of the specifications.

S-Specs are exceptions to the published specifications and apply only to the units assembled under that s-spec.

Errata are design defects or errors. Errata may cause the Pentium® processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given stepping must assume that all errata documented for that stepping are present on all devices.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

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

Identification Information

The Pentium processor can be identified by the following register contents:

Processor Identification

Family[1]	100/133/166 MHz Pentium® Processor Model[2]	200/233 MHz Pentium Processor with MMX™ Technology Model[2]	166/266 MHz Low-Power Pentium Processor with MMX Technology Model[2]
05H	02H	04H	08H

NOTES:
1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed, and the model field of the Device ID register accessible through Boundary Scan.

Marking Information

Figure 1. Topside Markings Key

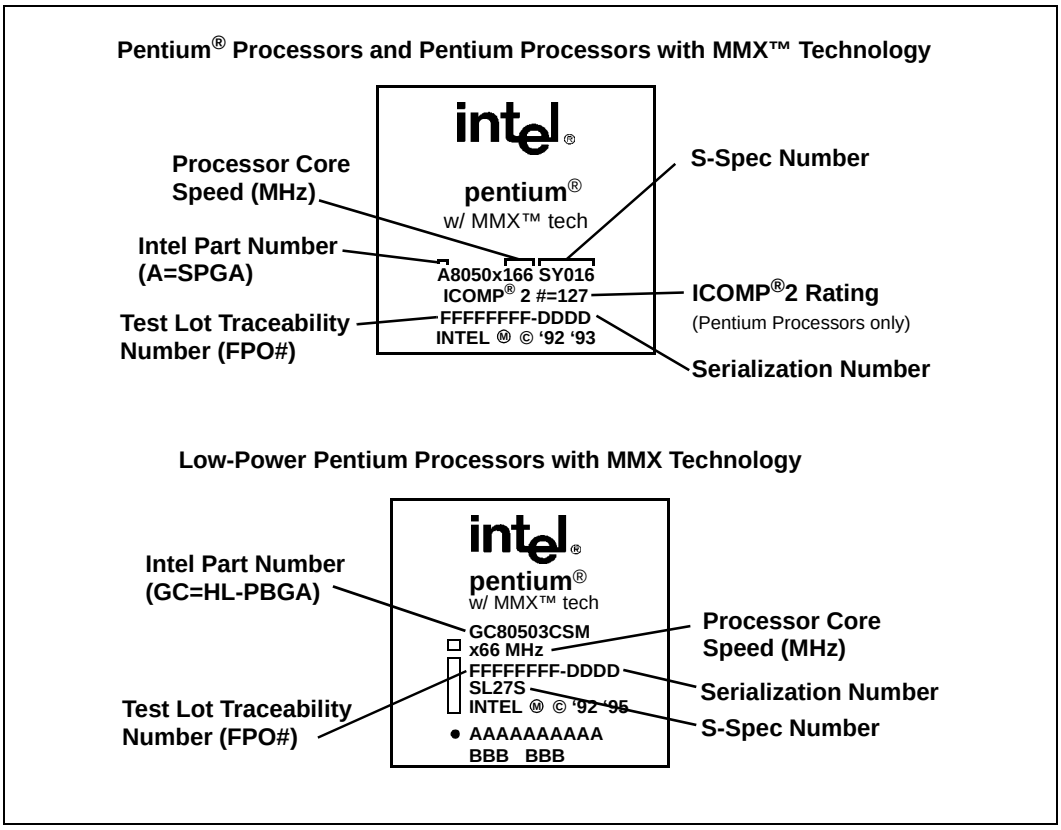
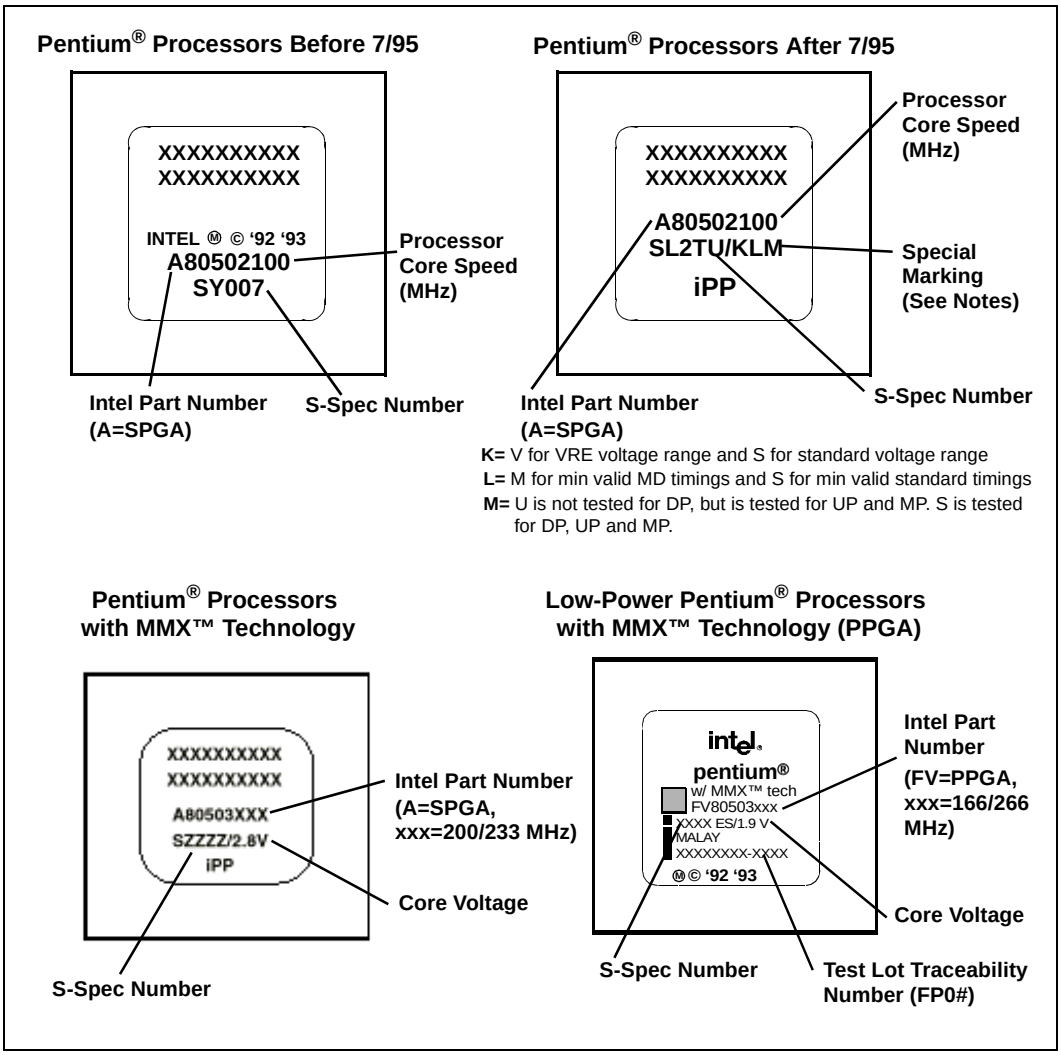
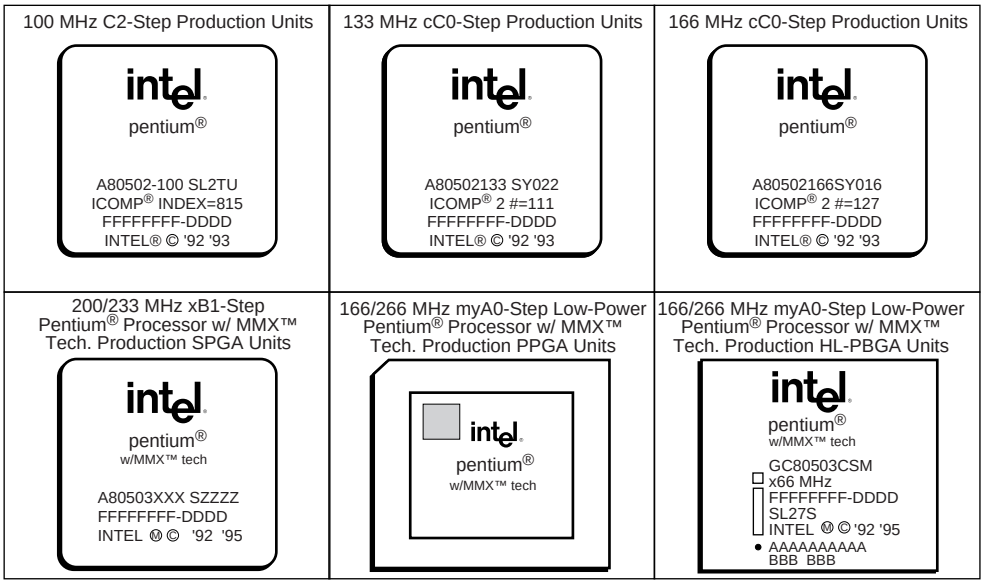


Figure 2. Bottomside Markings Key

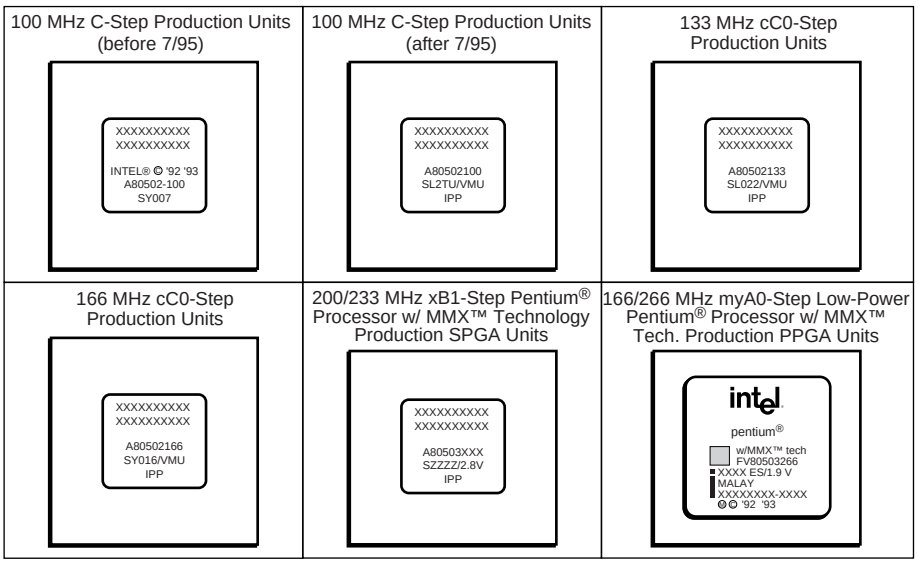


Top Markings



A6349-01

Bottom Markings



A6348-01

Processor Identification

- CPU Type of “2” or “0 or 2” indicates this part supports dual processing.
- The Type corresponds to bits [13:12] of the EDX register after RESET, bits [13:12] of the EAX register after the CPUID instruction is executed. This is shown as two different values based on the operation of the device as the primary processor or the dual processor upgrade.
- The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed.
- The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed.
- The Stepping corresponds to bits [3:0] of the EDX register after RESET, bits [3:0] of the EAX register after the CPUID instruction is executed.
- The absence of a package type in the comments column means the processor is SPGA by default.

Table 1. Pentium® Processor Identification Information

CPUID							
Type	**Family**	**Model**	**Stepping**	**Manufac-turing Stepping**	**Speed (MHz) Core / Bus**	**S-Spec**	**Comments**
0 or 2	5	2	C	cC0	133/66	SY022	STD
0 or 2	5	2	C	cC0	166/66	SY016	VRE[1], No Kit[2]
0	5	2	C	mcC0 [3]	133/66	SY028	SPGA 3.1V
0 or 2	5	2	6	E0	100/50 or 66	SY007	STD[4, 9]
0 or 2	5	2	C	cc0	100/50 or 66	SL2TU	STD
0 or 2	5	4	3	xB1	200/66	SL27J	PPGA[5]
0 or 2	5	4	3	xB1	233/66	SL27S	PPGA[5]
0	5	8	1	myA0	166/66	SL2ZX	PPGA[6]
0	5	8	1	myA0	166/66	SL388	HL-PBGA[7]
0	5	8	1	myA0	266/66	SL2Z4	PPGA[6]
0	5	8	1	myA0	266/66	SL389	HL-PBGA[8]

NOTES:
1. VRE: These parts have a reduced and shifted voltage specification, and reductions in the minimum output valid delays on the pins listed in the specifications in S-Spec 10. The VRE voltage range for the C2 and subsequent steppings of the Pentium processor is VCC = 3.40-3.60V. The VRE voltage range for B-step parts remains at 3.45-3.60V.
2. No Kit means that part meets the specifications but is not tested to support 82498/82493 and 82497/82492 cache timings.
3. The mcC0-step uses Intel’s VRT (Voltage Reduction Technology) to support mobile applications.
4. STD: The VCC specification for the C2 and subsequent steppings of the Pentium processor is VCC = 3.135V to 3.6V. The voltage range for B-step parts remains at 3.135V–3.465V (B-step devices are no longer offered). All E0-step production parts are standard voltage.
5. This is a Pentium processor with MMX technology with a core operating voltage of 2.7V –2.9V.
6. This is a Low-Power Pentium processor with MMX technology with a core operating voltage of 1.75V – 2.04V and an I/O operating voltage of 2.375V – 2.625V.
7. This is a Low-Power Pentium processor with MMX technology with a core operating voltage of 1.665V – 1.935V and an I/O operating voltage of 2.375V – 2.625V.
8. This is a Low-Power Pentium processor with MMX technology with a core operating voltage of 1.85V – 2.15V and an I/O operating voltage of 2.375V – 2.625V.
9. The SY007 product has been discontinued. SY007 was replaced by SL2TU.

Summary Table of Changes

The following table indicates the Specification Changes, S-Specs, Errata, Specification Clarifications or Documentation Changes, which apply to the listed 100/133/166 MHz Pentium processor steppings. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

Codes Used in Summary Table

X:	Erratum, Specification Change or Clarification that applies to this stepping.
Doc:	Document change or update that will be implemented.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
(No mark) or (Blank Box):	This erratum is fixed in listed stepping or specification change does not apply to listed stepping.
DP:	Dual processing related errata.
AP:	APIC related errata.
TCP:	Applies to the listed stepping of a mobile Pentium processor in a TCP package only.
▌	Change bar to left of table row indicates this erratum is either new or modified from the previous version of the document.

Specification Changes

No.	cC0	mcC0	E0	xB1	myA0	Plans	Specification Changes
1	X	X	X	X	X	Doc	IDT limit violation causes GP fault, not interrupt 8
2	Note 1					Doc	150 MHz active power dissipation (typical) change
3	X	X	X	X	X	Doc	Redundant timing spec: t_{42d} for all bus frequencies
4	Note 1					Doc	Stop clock power
5				X		Doc	Max valid delay A3 – A31
6	Note 1					Doc	Max valid delay data bus D0 – D63
7						Doc	Maximum Stop-Grant/AutoHALT Power
8						Doc	TCK V_{IL}
9				X		Doc	2/7 bus fraction
10				X		Doc	233-MHz I_{CC} specifications
11				X		Doc	Active power
12	Note 1					Doc	133-MHz current and power specifications
13						Doc	1/2 bus fraction
14						Doc	Maximum thermal design power
15				X		Doc	PCHK# Low state output current in DP mode
16				X		Doc	Absolute maximum rating for V_{CC3}
17	Note 1					Doc	120/60 MHz V_{CC}, I_{CC}, power, DC and AC specifications
18						Doc	166- and 266-MHz V_{CC2}, I_{CC} and power specifications
19						Doc	Mobile 200/66 MHz on 0.35 micron V_{CC2}, I_{CC}, T_{CASE} and power specifications
20						Doc	Low Voltage 266 MHz on 0.25 micron Process Technology (for TCP only) VCC, ICC, TCASE and Power Specifications

NOTE:
1. This item does not apply to Pentium® processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

S-Specs

No.	cC0	mcC0	E0	xB1	myA0	Plans	S-Spec Changes
1	Note 1					Fixed	t_{6a}, t_{6b}, max valid delay A31 – A3, BE7# – BE0#, ADS#, LOCK#
2						Fixed	Minimum required voltage separation between V_{CC3} and V_{CC2}
3						Fixed	V_{IH} for TRST#
4						Fixed	V_{IL} for BF and BF1 is reduced
5		X				Fixed	Boundary scan timing changes
6		X				Fixed	SPGA V_{CC2} supply voltage change
7	Note 1					Fixed	AC specifications for the Pentium Processor with Voltage Reduction Technology
8		X				Fixed	Reduced V_{IL} for TCK
9	Note 1					Fixed	Mixing steppings in dual processing mode
10	X					Fixed	MD/VR/VRE specifications
11	Note 1					Fixed	120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0732, Q0733, Q0751, Q0775, SK086, SX994, SK098, SU033) do not support dual processing
12						Fixed	120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) do not support FRC
13						Fixed	120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) V_{CC} to CLK startup specification
14						Fixed	120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) current leakage on PICD1 pin
15						Fixed	Mobile stop clock power
16						Fixed	I_{IH}, input leakage current
17						NoFix	Max valid delay A3 – A31 (Replaced by a Spec Change)
18						Fixed	Max valid delay ADS#
19						Fixed	Max valid delay HITM#
20						NoFix	Max valid delay data bus D0 – D63 (Replaced by a Spec Change)
21						Fixed	Desktop stop clock power
22						Fix	Min valid delay data bus D0 – D63

NOTE:
1. This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

Errata

No.	cC0	mcC0	E0	xB1	myA0	Plans	Errata
1	Note 1					Fixed	Branch trace messages during lock cycles
2						Fixed	Breakpoint or single-step may be missed for one instruction following STI
3						Fixed	I/O restart does not function during single stepping or data breakpoint exceptions
4						Fixed	NMI or INIT in SMM with I/O restart during single-stepping
5						Fixed	SMI# and FLUSH# during shutdown
6						Fixed	No shutdown after IERR#
7						Fixed	FLUSH# with a breakpoint pending causes false DR6 values
8						Fixed	Processor core may not serialize on bus idle
9						Fixed	SMIACT# premature assertion during replacement writeback cycle
10							STPCLK# deassertion not recognized for 5 CLKs after BRDY# returned
11						Fixed	Future Pentium OverDrive® processor FERR# contention in two-socket systems
12						Fixed	Code cache lines are not invalidated if snooped during AutoHALT or Stop-Grant states
13						Fixed	STPCLK# assertion during execution of the HALT instruction hangs system
14	X	X	X	X	X	NoFix	NMI or INIT during HALT within SMM may cause large amount of bus activity
15	X	X	X			Fixed	RUNBIST restrictions when run through boundary scan circuitry
16	X		X			Fixed	FRC mode miscompare due to uninitialized internal register
17	Note 1						STPCLK# restrictions during EWBE#
18						Fixed	Multiple allocations into branch target buffer
19						Fixed	100-MHz REP MOVS speed path
20						Fixed	Overflow undetected on some numbers on FIST
21						Fixed	Six operands result in unexpected FIST operation
22						Fixed	Snoop with table-walk violation may not invalidate snooped line
23						Fixed	Slight precision loss for floating-point divides on specific operand pairs
24						Fixed	FLUSH#, INIT or machine check dropped due to floating-point exception
25						Fixed	Floating-point operations may clear alignment check bit
26						Fixed	CMPXCHG8B across page boundary may cause invalid opcode exception
27				X	X	NoFix	Single-step debug exception breaks out of HALT
28	X	X	X			Fixed	Branch trace message corruption

NOTE:
1. This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

Errata

No.	cC0	mcC0	E0	xB1	myA0	Plans	Errata
29	Note 1					Fixed	FRC lock-step failure during APIC write
30						Fixed	BE4# – BE0# sampled incorrectly at min V_{IH}
31						Fixed	Incorrect PCHK# output during boundary scan if in DP mode
32						Fixed	EIP altered after specific FP operations followed by MOV Sreg, Reg
33	X	X	X			Fixed	WRMSR into illegal MSR does not generate GP Fault
34	Note 1					Fixed	Inconsistent data cache state from concurrent snoop and memory write
35						Fixed	BE3# – BE0# not driven during boundary scan if RESET high
36	X	X	X			Fixed	Incorrect FIP after RESET
37	X	X	X	X	X	NoFix	Second assertion of FLUSH# not ignored
38	X	X	X	X	X	NoFix	Segment limit violation by FPU operand may corrupt FPU state
39	X	X	X	X	X	NoFix	FP exception inside SMM with pending NMI hangs system
40	Note 1					Fixed	Current in Stop Clock state exceeds specification
41			X			Fixed	STPCLK# buffer samples incorrectly during boundary scan testing
42	Note 1					Fixed	Incorrect decode of certain 0F instructions
43	X	X	X	X	X	NoFix	Data breakpoint deviations
44	X	X	X	X	X	NoFix	Event monitor counting discrepancies
45	X	X	X	X	X	NoFix	VERR type instructions causing page fault task switch with T bit set may corrupt CS:EIP
46	X	X	X	X	X	NoFix	BUSCHK# interrupt has wrong priority
47			X			Fixed	BF and CPUTYP buffers sample incorrectly during boundary scan testing
48	X	X	X	X	X	NoFix	Matched but disabled data breakpoint can be lost by STPCLK# assertion
49	X	X	X	X	X	NoFix	STPCLK# ignored in SMM when INIT or NMI pending
50	X	X	X	X		Fixed	STPCLK# pullup not engaged at RESET
51	X	X	X	X	X	NoFix	A fault causing a page fault can cause an instruction to execute twice
52	X	X	X	X	X	NoFix	Machine check exception pending, then HLT, can cause skipped or incorrect instruction, or CPU hang
53	X	X	X	X	X	NoFix	FBSTP stores BCD operand incorrectly If address wrap and FPU error both occur
54	X	X	X	X	X	NoFix	V86 interrupt routine at illegal privilege level can cause spurious pushes to stack
55	X	X	X	X	X	NoFix	Corrupted HLT flag can cause skipped or incorrect instruction, or CPU hang
56	X	X	X	X	X	NoFix	Benign exceptions can erroneously cause double fault
57	X	X	X	X	X	NoFix	Double fault counter may not increment correctly
58		X				Fixed	Some input pins may float high when core V_{CC} powers up after I/O V_{CC} (mobile CPU)
59	X	X	X	X	X	NoFix	Short form of mov EAX / AX / AL may not pair
60	X	X	X	X	X	NoFix	Turning off paging may result in prefetch to random location

NOTE:

1. This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

Errata

No.	cC0	mcC0	E0	xB1	myA0	Plans	Errata
61	X	X	X	X	X	NoFix	STPCLK#, FLUSH# or SMI# after STI
62	X	X	X	X	X	NoFix	REP string instruction not interruptible by STPCLK#
63	X	X	X	X	X	NoFix	Single step may not be reported on first instruction after FLUSH#
64	X		X	X	X	NoFix	Double fault may generate illegal bus cycle
65	X	X	X	X	X	NoFix	TRST# not asynchronous
66	X	X	X	X	X	NoFix	STPCLK# on RSM to HLT causes non-standard behavior
67	X	X	X	X	X	NoFix	Code cache dump may cause wrong IERR#
68	X	X	X	X	X	NoFix	Asserting TRST# pin or issuing JTAG instructions does not exit TAP Hi-Z state
69	X	X	X	X	X	NoFix	ADS# may be delayed after HLDA deassertion
70	X	X	X	X	X	NoFix	Stack underflow in IRET gives #GP, not #SS
71	X	X	X	X	X	NoFix	Performance monitoring pins PM[1:0] may count the events incorrectly
72	Note 1					Fixed	BIST is disabled
73				X	X	NoFix	Branch trace messages may cause system hang
74	Note 1					Fixed	Enabling RDPMC in CR4 and also using SMM may cause shutdown
75					X	Fixed	Event monitor counting discrepancies (fix)
76				X	X	NoFix	Event monitor counting discrepancies (Nofix)
77	Note 1					Fixed	INVD may leave valid entries in the cache due to snoop interaction
78				X	X	NoFix	TLB update is blocked after a specific sequence of events with a misaligned descriptor
79	X	X	X	X	X	NoFix	Erroneous debug exception on POPF/IRET instructions with a GP fault
80				X	X	NoFix	CR2 and CR4 Content upon Return from SMM
81	X	X	X	X	X	Fix	Invalid operand with locked CMPXCHG8B instruction
82	X	X	X	X	X	Fix	Event monitor counting discrepancy
83	X	X	X	X	X	NoFix	FBSTP instruction incorrectly sets Accessed and Dirty bits of Page Table entry
1DP	Note 1					Fixed	Problem with external snooping while two cycles are pending on the bus
2DP						Fixed	STPCLK# assertion and the Stop-Grant bus cycle
3DP						Fixed	External snooping with AHOLD asserted may cause processor to hang
4DP						Fixed	Address parity check not supported in dual processing mode
5DP						Fixed	Inconsistent cache state may result from interprocessor pipelined READ into a WRITE
6DP						Fixed	Processors hang during Zero WS, pipelined bus cycles
7DP						Fixed	Bus lock-up problem in a specific dual processing mode sequence
8DP						Fixed	Incorrect assertion of PHITM# without PHIT#
9DP						Fixed	Double issuance of read cycles
10DP						Fixed	Line invalidation may occur on read or prefetch cycles

NOTE:

1. This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

Errata

No.	cC0	mcC0	E0	xB1	myA0	Plans	Errata
11DP	X		X			Fixed	EADS# or floating ADS# may cause extra invalidates
12DP	Note 1					Fixed	HOLD and BOFF# during APIC cycle may cause dual processor arbitration problem
13DP						Fixed	System hang after hold during local APIC second INTA cycle
14DP	X		X			Fixed	External snoop can be incorrectly invalidated
15DP	X		X	X		NoFix	STPCLK# re-assertion recognition constraint with DP
16DP	X		X	X		NoFix	Second assertion of FLUSH# during flush acknowledge cycle may cause hang
17DP				X		NoFix	Asserting FLUSH# may cause a processor deadlock in a DP system with a 2/7 bus fraction
1AP	Note 1					Fixed	Remote read message shows valid status after a checksum error
2AP						Fixed	Chance of clearing an unread error in the error register
3AP						Fixed	Writes to error register clears register
4AP						Fixed	Three interrupts of the same priority causes lost local interrupt
5AP						Fixed	APIC bus synchronization lost due to checksum error on a remote read message
6AP						Fixed	HOLD during a READ from local APIC register may cause incorrect PCHK#
7AP						Fixed	HOLD during an outstanding interprocessor pipelined APIC cycle hangs processor
8AP						Fixed	PICCLK reflection may cause an APIC checksum error
9AP	X		X			Fixed	Spurious interrupt in APIC through local mode
10AP	Note 1					Fixed	Potential for lost interrupts while using APIC in through Local mode
11AP						Fixed	Back to back assertions of HOLD or BOFF# may cause lost APIC write cycle
12AP	X					Fixed	System hangs when BOFF# is asserted during second internal INTA cycle
13AP	X		X			Fixed	APIC pipeline cycle during cache linefill causes restarted cycle to lose its attribute
14AP	X		X	X	X	NoFix	INIT and SMI via the APIC three-wire bus may be lost
15AP	X		X			Fixed	IERR# in FRC lock-step mode during APIC write
16AP	X		X			Fixed	Inadvertent BRDY# during external INTA cycle with BOFF#
17AP	X		X			Fixed	APIC read cycle may not complete upon assertion of BOFF# and HOLD
18AP	X		X	X	X	NoFix	PICCLK must toggle for at least twenty cycles before RESET
19AP	Note 1					Fixed	APIC ID can not be changed
1TCP						Fixed	CPU may not reset correctly due to floating FRCMC# pin
2TCP						Fixed	BRDY# does not have buffer selection capability

NOTE:

1. This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

Specification Clarifications

No.	cC0	mcC0	E0	xB1	myA0	Plans	Specification Clarifications
1	X		X	X		Doc	Pentium processor's response to startup and init IPIs
2	Note 1					Doc	APIC timer use clarification
3						Fixed	PICCLK reflection may cause APIC checksum errors and dropped IPIs
4						Fixed	Boundary scan RUNBIST register requires initialization prior to use
5	X	X	X	X	X	Doc	Only one SMI# can be latched during SMM
6	X		X	X	X	Doc	APIC 8-bit access
7	X		X	X	X	Doc	LOCK prefix excludes APIC memory space
8	X	X	X	X	X	Doc	SMI# activation may cause a nested NMI handling
9	X	X	X	X	X	Doc	Code breakpoints set on meaningless prefixes not guaranteed to be recognized
10	X	X	X	X	X	Doc	Resume flag should be set by software
11	X	X	X	X	X	Doc	Data breakpoints on INS delayed one iteration
12	X	X	X	X	X	Doc	When L1 cache disabled, inquire cycles are blocked
13	X	X	X	X	X	Doc	Serializing operation required when one CPU modifies another CPU's code
14	X	X	X	X	X	Doc	For correct translations, the TLB should be flushed after the PSE bit in CR4 is set
15	X		X	X		Doc	When APIC enabled, its 4K block should not be used in regular memory
16	X	X	X	X	X	Doc	Extra code break can occur on I/O or HLT instruction if SMI coincides
17				X	X	Doc	LRU maybe updated for non-cacheable cycles
18	X	X	X	X	X	Doc	FYL2XP1 does not generate exceptions for X out of range
19	X	X	X	X	X	Doc	Enabling NMI inside SMM
20	X	X	X	X	X	Doc	BF[1:0] must not change values while RESET is active
21	X	X	X	X	X	Doc	Active A20M# during SMM
22	X	X	X	X	X	Doc	POP[ESP] with 16-bit stack size
23	Note 1					Doc	Pin #11 and pin #190 (TCP package) connection
24	X	X	X	X	X	Doc	Line fill order optimization revision
25	X	X	X	X	X	Doc	Test Parity Check Mechanism Clarification

NOTE:
1. This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

Documentation Changes

No.	cC0	mcC0	E0	xB1	myA0	Plans	Documentation Changes
1	X	X	X	X	X	Doc	JMP Cannot Do a Nested Task Switch, Volume 3, Page 13-12
2	X	X	X	X	X	Doc	Interrupt Sampling Window, Volume 3, Page 23-39
3	X	X	X	X	X	Doc	FSETPM Is Like NOP, Not Like FNOP
4	X	X	X	X	X	Doc	Errors in Three Tables of Special Descriptor Types
5	X	X	X	X	X	Doc	Invalid Arithmetic Operations and Masked Responses to Them Relative to FIST/FISTP Instruction
6	X	X	X	X	X	Doc	Incorrect Sequence of Registers Stored in PUSHA/PUSHAD
7	X	X	X	X	X	Doc	One-Byte Opcode Map Correction

Specification Changes

The Specification Changes listed in this section apply to the documents listed in the Preface of this Specification Update. Specification Changes may be incorporated into future versions of the appropriate document(s).

1. IDT Limit Violation Causes GP Fault, Not Interrupt 8

The last sentence in Section 9.3 of the *Pentium® Processor Family Developer's Manual*, Volume 3, says about exception handling in Real Mode: "If an interrupt occurs and its entry in the interrupt table is beyond the limit stored in the IDTR register, a double-fault exception is generated." In fact, in the Pentium processor, there is no difference between Real and Protected Mode when an IDT limit violation occurs. It generates interrupt 13: General Protection Fault in both modes.

2. 150 MHz Active Power Dissipation (Typical) Change

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

3. Redundant Timing Specification: t42d for All Bus Frequencies (50, 60 and 66 MHz)

t_{42d} is the minimum hold time required when BRDYC# is used as a configuration signal and when RESET is driven synchronously with CLK. This timing specification is redundant because it is a subset of t_{21}, hold time for BRDYC# with respect to CLK. Therefore, t_{42d} will be removed from all future documentation.

4. Stop Clock Power

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

5. Max Valid Delay A3 – A31

These are the new specifications for the maximum valid delay for t_{6e} of the Pentium processor with MMX technology. This will replace both the original specifications of t_{6e} in the *Pentium® Processor with MMX™ Technology* datasheet as well as S-Spec 17, mobile max valid delay A3–A31.

Symbol	Parameter	Bus Frequency	Previous Max Valid Delay	New Max Valid Delay
t_{6e}	A3 – A31	60 MHz (mobile only)	6.3 ns	7.0 ns
t_{6e}	A3 – A31	66 MHz	6.3 ns	6.6 ns

6. Max Valid Delay Data Bus D0 – D63

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

7. Maximum Stop-Grant/AutoHALT Power

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

8. TCK VIL

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

9. 2/7 Bus Fraction

Bus Frequency Selections

BF1	BF0	Bus/Core Ratio	Max Bus/Core Frequency (MHz)	Min Bus/Core Frequency (MHz)
0	1	1/3	66/200	33/100
0	0	2/5	66/166	33/83
1	0	1/2 (1)	66/133	33/66
1	1	2/7	66/233	33/116

NOTE:
1. This is the default bus fraction for the Pentium processor with MMX™ technology. If the BF pins are left floating, the processor will be configured for the 1/2 bus to core frequency ratio.

10. 233-MHz ICC Specifications

Specifications (Measured at VCC2 =2.9V and VCC3 =3.6V)

Symbol	Parameter	Min	Max	Unit	Notes
I_{CC2}	Power Supply Current		6500	mA	233 MHz(1)
I_{CC3}	Power Supply Current		750	mA	233 MHz(1)

NOTE:
1. This value should be used for power supply design. It was determined using a worst case instruction mix and maximum V_{CC}. Power supply transient response and decoupling capacitors must be sufficient to handle the instantaneous current changes occurring during transitions from Stop Clock to full Active modes.

11. Active Power

Power Dissipation Requirements for Thermal Design (Measured at VCC2 =2.8V and VCC3 =3.3V)

Parameter	Typical(1)	Max(2)	Unit	Notes
Active Power	7.9(3)	17.0(4)	Watts	233 MHz

NOTE:
1. This is the typical power dissipation in a system. This value is expected to be the average value that will be measured in a system using a typical device at V_{CC2} = 2.8V running typical applications. This value is highly dependent upon the specific system configuration. Typical power specifications are not tested.
2. Systems must be designed to thermally dissipate the maximum active power dissipation. It is determined using worst case instruction mix with V_{CC2} = 2.8V and V_{CC3} = 3.3 and also takes into account the thermal time constants of the package.
3. Active Power (typ) is the average power measured in a system using a typical device running typical applications under normal operating conditions at nominal V_{CC} and room temperature.
4. Active Power (max) is the maximum power dissipation under normal operating conditions at nominal V_{CC2}, worst-case temperature, while executing the worst case power instruction mix. Active power (max) is equivalent to Thermal Design Power (max).

12. 133-MHz Current and Power Specifications

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

13. 1/2 Bus Fraction

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

14. Maximum Thermal Design Power

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

15. PCHK# Low State Output Current in DP Mode

In the *Pentium® Processor with MMX™ Technology* datasheet, Note 4 on page 31 should be modified. Specifically, the PCHK# in a DP system should be changed from 14 mA to 13 mA. The new note should read:

"4. In dual processing systems, up to a 9 mA load from the second processor may be observed on the PCHK# signal. Based on silicon characterization data, VOL3 of PCHK# will remain less than 400 mV even with a 9 mA load. PCHK# VOL3 will increase to approximately 500 mV with a 13 mA load (worst case for a DP system with a 4 mA system load)."

16. Absolute Maximum Rating for V_{CC3}

In the 1997 *Pentium® Processor Family Developer's Manual*, Section 7.2, Table 7-1, it states the maximum VCC3 is 4.6V. This applies to the Pentium processor 75/90/100/120/133/150/166/200 only. The maximum VCC3 for the Pentium processor with MMX technology is 4.0V.

17. 120/60 MHz V_{CC}, I_{CC}, Power, DC and AC Specifications

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

18. 166- and 266-MHz V_{CC2}, I_{CC} and Power Specifications

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

19. Mobile 200/66 MHz on 0.35 micron V_{CC}, I_{CC}, T_{CASE} and Power Specifications

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

20. Low Voltage 266 MHz on 0.25 micron Process Technology (for TCP only) VCC, ICC, TCASE and Power Specifications

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

S-Specs

1. t_{6a}, t_{6c}, Max Valid Delay A31-A3, BE7#-BE0#, ADS#, LOCK#

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

2. Minimum Required Voltage Separation Between V_{CC3} and V_{CC2}

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

3. V_{IH} For TRST#

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

4. V_{IL} For BF0 and BF1 is Reduced

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

5. Boundary Scan Timing Changes

The boundary scan valid delay minimum time for t_{53} and t_{55} has been reduced for the mcC0 stepping as indicated below. This applies to both SPGA and TCP packages.

Symbol	Parameter	Standard Min Time (60/66MHz)	S-Spec Min Time (60/66MHz)
t_{53}	TDO Valid Delay	3.0 nS	2.8 nS
t_{55}	All Non-Test Outputs Valid Delay	3.0 nS	2.5 nS

6. SPGA VCC2 Supply Voltage Change

The core supply voltage (VCC2) required is changed from 2.9V to 3.1V. This applies to SPGA 100/133 MHz units only. I/O voltage supply (VCC3) remains at 3.3V+/-165mV.

Symbol	Parameter	Standard Supply Voltage	S-Spec Supply Voltage
V_{CC2}	Core voltage supply	2.9V+/-165mV	3.1V+/-165mV

7. AC Specifications for the Pentium® Processor with Voltage Reduction Technology

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

8. Reduced VIL For TCK

Symbol	Pin	Standard Min	S-Spec Min	Unit
V_{IL}	TCK	0.8	0.6	Volts

9. Mixing Steppings in Dual Processing Mode

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

10. MD/VR/VRE Specifications

There are some changes to the standard VCC and timing specifications to support the highest performance operation of the Pentium processor.

STD: The VCC specification for the C2 and subsequent steppings of the Pentium processor is VCC = 3.135V to 3.6V. The voltage range for B-step parts remains at 3.135V–3.465V. Note that all E0-step production parts are standard voltage.

VR: This is a reduced voltage specification that has the range of 3.300V–3.465V.

VRE/MD: These parts have a reduced and shifted voltage specification, and reductions in the minimum output valid delays on the list of pins in the table below. The VRE voltage range for the C2 and subsequent steppings of the Pentium processor is VCC = 3.40-3.60V. The VRE voltage range for B-step parts remains at 3.45-3.60V.

MD: This is a reduction in the minimum valid timings on a subset of output pins. Due to faster operation of the core, and faster operation of the transistors at the higher voltages these minimum valid timings need to be met. These parts have the standard VCC specification.

	Previous	Current
Operating V_{CC} Range (VRE)	3.45 to 3.60V	3.40 to 3.60V

There are no allowances for crossing the high and low limits of the voltage specification. Part operation beyond these ranges cannot be guaranteed. For more information on measurement techniques, see Chapter 7 of the *Pentium® Processor Family Developer's Manual* and the application note *Implementation Guidelines for 3.3V Pentium® Processors with VR/VRE Specifications.*

Symbol	Signal	Min Valid STD (50/60/66) Specifications	Min Valid, MD (50/60/66) Specifications	Units
t_{6c}	A3-16	1.1	0.5	nS
t_{6c}	A17-31	1.1	0.6	nS
t_{6a}	W/R#	1.0	0.8	nS
t_{6a}	M/IO#	1.0	0.8	nS
t_{6a}	D/C#	1.0	0.8	nS
t_{6c}	LOCK#	1.1	0.9	nS
t_{10b}	HITM#	1.1	0.7	nS
t_{6a}	BE0-7#	1.0	0.9	nS

11. *120-MHz and 133-MHz Parts (Q0707, Q0708, Q0711, Q0732, Q0733, Q0751, Q0775, SK086, SX994, SU033, SK098) Do Not Support Dual Processing*

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

12. 120-MHz and 133-MHz Parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) Do Not Support FRC

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

13. 120-MHz and 133-MHz Parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) V_{CC} to CLK Startup Specification

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

14. 120-MHz and 133-MHz Parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) Current Leakage on PICD1 Pin

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

15. Mobile Stop Clock Power

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

16. I_{IH} Input Leakage Current

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

17. Max Valid Delay A3-A31

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

18. Max Valid Delay ADS#

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

19. Max Valid Delay HITM#

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

20. Max Valid Delay Data Bus D0-D63

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

21. Desktop Stop Clock Power

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

22. Min Valid Delay Data Bus D0 – D63

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

Errata

1. Branch Trace Message During Lock Cycles

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

2. Breakpoint or Single-Step May Be Missed for One Instruction Following STI

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

3. I/O Restart Does Not Function During Single Stepping or Data Breakpoint Exceptions

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

4. NMI or INIT in SMM with I/O Restart During Single Stepping

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

5. SMI# and FLUSH# During Shutdown

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

6. No Shutdown After IERR#

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

7. FLUSH# with a Breakpoint Pending Causes False DR6 Values

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

8. Processor Core May Not Serialize on Bus Idle

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

9. SMIACT# Premature Assertion During Replacement Writeback Cycle

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

10. STPCLK# Deassertion Not Recognized for 5 CLKs After BRDY# Returned

This erratum has been superseded by a specification change.

11. Future Pentium® OverDrive® Processor FERR# Contention in Two-Socket Systems

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

12. Code Cache Lines Are Not Invalidated if Snooped During AutoHALT or Stop Grant States

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

13. STPCLK# Assertion During Execution of the HALT Instruction Hangs System

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

14. NMI or INIT During HALT Within SMM May Cause Large Amount of Bus Activity

Problem: If a HALT or REP (repeat string instruction) instruction is executed while the processor is in System Management mode (SMM), and an NMI or INIT is asserted prior to interrupt initialization, the processor may continuously re-execute the HALT, and generate the HALT special cycle, or it will perform iterations of the REP instruction that was executed. Normally the processor would ignore NMI and INIT while in SMM. However, NMI and INIT will be enabled inside of SMM if interrupts have been enabled and then an INTR signal is received. Also, exceptions, when taken, enable NMI and INIT inside of SMM, but this behavior is not part of the Intel Architecture.

Implication: The processor may continuously run the same cycle on the bus until a non-masked interrupt is detected. There are no other problems associated with the erratum, the component resumes correct operation at this time. This impacts the “low power operation” that might have been expected with the use of a HALT while in SMM.

Workaround: Use one of the following:

1. Do not use HALT while in SMM.
2. If the system must use HALT in SMM, the system is required to initialize interrupt vector tables prior to use of any interrupts, doing so will ensure the error will not occur. The system must ensure that NMI and INIT are not asserted while the processor is HALTed in System Management mode, prior to interrupt vector initialization.

Status: For the steppings affected see the Summary Table of Changes.

15. RUNBIST Restrictions When Run through Boundary Scan Circuitry

Problem: When the built in self test (Runbist) is run via the Boundary Scan circuitry a failing result is shown on the device. This failing result appears even after initializing the RESET cell as described in Chapter 11 of the *Pentium® Processor Family Developer’s Manual*.

Implication: If one of the workarounds listed is not implemented then the system cannot depend of the result of this test as part of a Boundary Scan generated manufacturing test or power on test.

Workaround: Use one of the following workarounds. Both of these workarounds rely on the initialization of the RESET scan cell as stated in the Specification Clarifications section of this document.

1. Although not IEEE 1149.1 compatible, it has been found that if BRDY# is asserted low for every ADS# the processor generates, the Runbist test completes correctly. If the system can return these BRDY#s to the CPU then the BIST functionality can be utilized on the processor through Boundary Scan.
2. If RESET is held HIGH during the execution of the RUNBIST Boundary Scan instruction and the subsequent 219 core clocks.

Status: For the steppings affected see the Summary Table of Changes.

16. FRC Mode Miscompare Due to Uninitialized Internal Register

Problem: There is a mismatch and a resulting IERR# assertion when running in FRC mode due to an uninitialized internal register in the paging unit. The failure mechanism is due to uninitialized data being driven on the upper 32-bits of the data bus while updating a page table entry on the lower 32-bits (upon enabling paging). The data bits that mismatch are not valid during that bus cycle (byte enables are inactive), so the IERR# output is due to a spurious comparison.

Implication: The FRC mode of the processor requires use of a workaround to initialize the paging unit if addresses in the upper 32 bits are accessed.

Workaround: Initialize this internal register through software by performing a dummy page table lookup on the upper 32 bits. (In a code segment with linear address bit 22 set to '1', turn paging on and then turn it off again immediately).

Status: For the steppings affected see the Summary Table of Changes.

17. STPCLK# Restrictions During EWBE#

This erratum has been superseded by a specification change. This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

18. Multiple Allocations Into Branch Target Buffer

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

19. 100-MHz REP MOVS Speed Path

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

20. Overflow Undetected on Some Numbers on FIST

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

21. Six Operands Result in Unexpected FIST Operation

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

22. Snoop With Table-Walk Violation May Not Invalidate Snooped Line

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

23. Slight Precision Loss for Floating-point Divides on Specific Operand Pairs

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

24. FLUSH#, INIT or Machine Check Dropped Due to Floating-point Exception

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

25. Floating-point Operations May Clear Alignment Check Bit

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

26. CMPXCHG8B Across Page Boundary May Cause Invalid Opcode Exception

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

27. Single Step Debug Exception Breaks Out of HALT

Problem: When Single Stepping is enabled (i.e., the TF flag is set) and the HLT instruction is executed the processor does not stay in the HALT state as it should. Instead, it exits the HALT state and immediately begins servicing the Single Step exception.

Implication: The behavior described above is identical to Intel486 CPU behavior.

Workaround: None identified at this time.

Status: For the steppings affected see the Summary Table of Changes at the beginning of this section.

28. Branch Trace Message Corruption

Problem: When performing execution tracing (in normal or fast mode), the linear address of the instruction causing the taken branch is sent to the bus as part of a branch trace message. In a tight loop of code, the reported linear address of the instruction causing the taken branch may be corrupted in some branch trace messages. If the first branch trace message completes on the bus before the second one is posted, the problem will be avoided. Note that this erratum applies to normal mode for processor steppings prior to C2 and to fast mode on all processor steppings.

This erratum only affects execution tracing, a specialized feature allowing external hardware to track the flow of instructions as they execute in the processor. Regular operation of the processor is not affected.

Workaround: Use normal trace mode for processor steppings C2 and later since these steppings are not affected by this erratum in normal mode.

Status: For the steppings affected see the Summary Table of Changes.

29. FRC Lock Step Failure During APIC Write

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

30. BE4#-BE0# Sampled Incorrectly at Min Vih

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

31. Incorrect PCHK# Output During Boundary Scan if in DP Mode

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

32. EIP Altered After Specific FP Operations Followed by MOV Sreg, Reg

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

33. WRMSR Into Illegal MSR Does Not Generate GP Fault

Problem: The WRMSR and RDMSR instructions allow writing and reading of special MSRs (Model Specific Registers) based on the index number placed in ECX. The architecture was specified to reject access to illegal MSRs by generating the fault GP(0) if WRMSR or RDMSR is executed with an illegal index. However, negative indices, all of which are illegal, do not trigger GP(0).

Implication: If RDMSR is used with negative indices, undefined values will be read into EAX. If WRMSR is used with negative indices, undefined processor behavior may result.

Workaround: Do not use illegal indices with WRMSR and RDMSR.

Status: For the steppings affected see the Summary Table of Changes.

34. Inconsistent Data Cache State From Concurrent Snoop and Memory Write

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

35. BE3#-BE0# Not Driven During Boundary Scan if RESET High

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

36. Incorrect FIP After RESET

Problem: After a RESET, the floating point instruction pointer (FIP) should be initialized to 00000000h. The FIP will instead retain the value it contained prior to the RESET. The FIP gets updated whenever the processor decodes a floating point instruction other than an administrative floating point instruction (FCLEX, FLDCW, FSTCW, FSTSW, FSTSWAX, FSTENV, FLDENV, FSAVE, FRSTOR and FWAIT). If an FSAVE or FSTENV is executed after a RESET and before any non-administrative floating point instruction caused the FIP to be updated, the old value contained in the FIP will be saved. If a non-administrative floating point instruction is the first floating point instruction executed after RESET, the old value in the FIP will be overwritten and any successive FSAVE of FSTENV will save the correct value.

The FIP is used by software exception handlers to determine which floating point instruction caused the exception. The only instructions that can cause an exception are non-administrative floating point instructions, so a non-administrative floating point instruction is usually executed before an FSAVE or FSTENV.

Implication: If an FSAVE or FSTENV is executed after a RESET and before any non-administrative floating point instruction, the incorrect FIP will be saved.

Workaround: If an FSAVE or FSTENV is executed after a RESET and before a non-administrative floating point instruction is executed, perform an FINIT instruction after RESET as recommended in the *Intel Architecture Software Developer's Manual*, Volume 3, Section 8.2. This will set the FIP to 00000000h. Otherwise, no workaround is required.

Status: For the steppings affected see the Summary Table of Changes.

37. Second Assertion of FLUSH# Not Ignored

Problem: If FLUSH# is asserted while the processor is servicing an existing flush request, a second flush operation will follow after the first one completes. Proper operation is for a second assertion of FLUSH# to be ignored between the time the first FLUSH# is asserted and completion of its Flush Acknowledge cycle.

Implication: A system that asserts FLUSH# during a flush that's already in progress will flush the cache a second time. Flushing the cache again is not necessary and results in a slight performance degradation.

Workaround: For best performance, the system hardware should not assert any subsequent FLUSH# while a flush is already being serviced.

Status: For the steppings affected see the Summary Table of Changes.

38. Segment Limit Violation by FPU Operand May Corrupt FPU State

Problem: On the Intel486™, Intel386™ and earlier processors, if the operand of the FSTENV/FLDENV instructions, or the FSAVE/FRSTOR instructions, exceeds a segment limit during execution, the resulting General Protection fault blocks completion of the instruction. (Actually, interrupt #9 is generated in the 80386 and earlier.) This leaves the FPU state (with FLDENV, FRSTOR) or its image in memory (with FSTENV, FSAVE) partly updated, thus corrupted, and the instruction generally is non-restartable. It is stated in the *Intel Architecture Software Developer's Manual*, Volume 3, Chapter 17 that the Pentium processor fixes this problem by starting these instructions with a test read of the first and last bytes of the operand. Thus if there is a segment limit violation, it is triggered before the actual data transfer begins, so partial updates cannot occur.

This improvement works as intended in the large majority of segment limit violations. There is however a special case in which the beginning and end of the FPU operand are within the segment, so the endpoints pass the initial test, but part of the operand exceeds the segment limit. Thus part way through the data transfer, the limit is violated, the GP fault occurs, and thus the FPU state is corrupted. Note that this is a subset of the cases which will cause the same problem with Intel486 and earlier CPUs, so any code that executes correctly on those CPUs will run correctly on the Pentium processor.

This erratum will happen when both the segment limit and a 16 or 32 bit addressing wrap around boundary falls within the range of the FPU operand, with the segment limit below the wrap boundary. (To use a 16 bit wrap boundary of course, one must be executing code using 16 bit addressing.) The upper endpoint of the FPU operand wraps to near the bottom of the segment, so it passes the initial test. But part way through the data transfer the CPU tries to access memory above the segment limit but below the wrap boundary, causing the GP fault with the FPU state partly copied. This erratum can also happen if the segment limit is at or above a 16 bit addressing wrap boundary, with both straddled by an FPU operand that is **not aligned on an 8 byte boundary.** Test of the upper endpoint wraps and thus passes. When the instruction is actually transferring data, the misalignment forces the CPU to calculate extra addresses for special bus cycles. This special address calculation does not support the 16 bit wrap, so the GP fault is triggered when the segment limit is crossed.

Note that the *Intel Architecture Software Developer's Manual*, Volume 3, Chapter 17 warns in general that the Pentium processor may store only part of operands which generate a memory fault by crossing either a segment or page limit. This erratum is just one case of that general problem, and all cases will be avoided by following the recommended programming practice of never straddling segment or page boundaries with operands. Note also that the handling of operands which straddle such boundaries is processor specific, so code which uses such straddling will behave differently when run on different Intel Architecture processors.

Implication: This erratum can corrupt that state of the FPU and will cause a GP fault. This generally will require that the task using the FPU be restarted, but it will not cause unflagged errors in results. Code written following Intel recommendations, and any code which runs on the Intel486 (or earlier) CPUs, will not cause this erratum. The case where the Pentium processor will experience this erratum is a small subset of the cases in which the Intel486 (and earlier) CPUs will be corrupted.

Workaround:

1. Do not use code in which FPU operands wrap around the top of their segments.
2. If one must use FPU operands which wrap at the top of their segments, make sure that they are aligned on an 8 byte boundary, **and** that the segment limit is not below the 16 or 32 bit wrap boundary.

Status: For the steppings affected see the Summary Table of Changes.

39. FP Exception Inside SMM with Pending NMI Hangs System

Problem: If a previous FPU instruction has caused an unmasked exception, and an FP instruction is executed inside SMM with an NMI pending, the system will hang unless the system is both DOS compatible (CR0.NE=0), **and** external interrupts are enabled.

Implication: For standard PC-AT systems, NMI is typically used (if at all) to indicate a parity error, and the response required is a system reset, to preserve data integrity. So this erratum will only occur when the system has already suffered a parity error; the effect of the erratum is only to force reset inside SMM, instead of after the RSM when the NMI would normally be serviced. In a system where NMI is **not** used for an error that requires shutdown, the workaround should be implemented.

A properly designed system should not experience a hang-up. In such a system the SMM BIOS checks for pending interrupts before issuing an FSAVE or FRSTOR. If an interrupt is pending, the BIOS will exit SMM to handle the interrupt. If an interrupt is not present, the BIOS will disable interrupts (for example, it will disable NMI by writing to the chip set) and only then will issue the FP instruction.

Workaround: If FPU instructions are used in SMM, **and** NMI is used for other than an error that requires shutdown, NMI should be blocked from outside the CPU during SMM.

Status: For the steppings affected see the Summary Table of Changes.

40. Current in Stop Clock State Exceeds Specification

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

41. STPCLK# Buffer Samples Incorrectly During Boundary Scan Testing

Problem: During boundary scan input testing, the boundary scan input path in the STPCLK# buffer is disabled when RESET is high.

Implication: The boundary scan cell in the STPCLK# buffer captures a “1” from the STPCLK# pin regardless of the actual data on that pin when RESET is high. This violates the IEEE Specification 1149.1 which

states that the value driven should always be that in the boundary scan cell regardless of the state of RESET. However, the buffer functions correctly when the EXTEST instruction is used.

Workaround: If testing boundary scan in a system environment this pin can be left untested by marking this pin as “INTERNAL” in the BSDL file.

Status: For the steppings affected see the Summary Table of Changes.

42. Incorrect Decode of Certain 0F Instructions

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

43. Data Breakpoint Deviations

The following three problems are deviations from the data breakpoint specification when a fault occurs during an FP instruction while the data breakpoint is waiting to be serviced. They all share the same workaround. In the first case the breakpoint **is serviced,** incorrectly, before the actual data access that should trigger it takes place; in the other cases the breakpoint is **not serviced** when it should be.

Problem: **PROBLEM A: First**, the debug registers must be set up so that any of the FP instructions which read from memory (except for FRSTOR, FNRSTOR, FLDENV and FNLDENV) will trigger a data breakpoint upon accessing its memory operand. **Second**, there must be an unmasked FP exception pending from a previous FP instruction when the FP load or store instruction enters the execution stage. This so far would cause, per specification, a branch to the FP exception handler. The data breakpoint would not be triggered until/ unless the memory access is made after return from the exception handler. But if **third**, either of the external interrupts INTR or NMI is asserted after the FP instruction enters the execution stage, but before the branch to the FP exception handler occurs, this erratum is generated. In this situation, the processor should branch to the external interrupt handler, but instead it goes to the data breakpoint handler. This is incorrect because the data access that should trigger the breakpoint has not occurred yet.

Problem: **PROBLEM B:** Interrupts are blocked for the instruction after a MOV or POP to SS (to allow a MOV or POP to ESP to complete a stack switch before any interrupt). If the MOV or POP to SS triggers a data breakpoint, it normally is serviced after the following instruction is executed. However, if the following instruction is a FP instruction **and** there is a pending FP error from a preceding FP instruction (even if the error is masked), the delayed data breakpoint is forgotten.

Problem: **PROBLEM C:** If the sequence of memory accesses during execution of FSAVE or FSTENV (or their counterparts FNSAVE and FNSTENV) touches an enabled data breakpoint location, the data breakpoint exception (interrupt 1) occurs at the end of the FP instruction. If however the sequence of memory accesses cross a segment limit after touching the data breakpoint location, the General Protection (GP) fault will occur. This erratum is that as the processor branches to the GP fault handler, the valid data breakpoint is forgotten.

Implication: This erratum will only be seen by software or hardware developers using the data breakpoint feature of the debug registers. It can cause data breakpoints to be both lost, and asserted prematurely, as long as the contributing FP and GP errors remain uncorrected.

Workaround: Use one of the following:

1. General solution: For problems A & B to occur, an FP error must be caused by a preceding FP instruction, and in problem C, the FP operand causes a segment limit violation. These errors are all indicated in the normal way, despite this erratum. Eliminate them and this erratum disappears, allowing the data breakpoint debugging to proceed normally. Since debugging is usually done in successive stages, this workaround is usually performed as part of the debugging process.

2. Problem A may also be handled by blocking NMI and INTR during debugging.

Status: For the steppings affected see the Summary Table of Changes.

44. Event Monitor Counting Discrepancies

Problem: The Pentium processor contains two registers which can count the occurrence of specific events used to measure and monitor various parameters which contribute to the performance of the processor. There are several conditions where the counters do not operate as specified.

In some cases it is possible for the same instruction to cause the “Breakpoint match” (event 100011, 100100, 100101 or 100110) event counter to be incremented multiple times for the same instruction. Instructions which generate FP exceptions may be stalled and restarted several times causing the counter to be incremented every time the instruction is restarted. In addition, if FLUSH# or STPCLK# is asserted during a matched breakpoint or if a data breakpoint is set on a POP SS instruction, the counter will be incremented twice. The counter will (incorrectly) not get incremented if the matched instruction generates an exception and the exception handler does an IRET which sets the resume flag. The counter will also not get incremented for a data breakpoint match on a u-pipe instruction if the paired instruction in the v-pipe generates an exception.

The “Hardware interrupts” (event 100111) event counter counts the number of **taken** INTR and NMIs. In the event that both INTR/NMI and a higher priority interrupt are present on the same instruction boundary, the higher priority interrupt correctly gets processed first. However, the counter prematurely counts the INTR/NMI as taken and the count incorrectly gets incremented.

The “Code breakpoint match” (event 100011, 100100, 100101 or 100110) event counter may also fail to be incremented in some cases. If there is a code breakpoint match on an instruction and there is also a single-step or data breakpoint interrupt pending, the code breakpoint match counter will not be incremented.

The “Non-cacheable memory reads” (event 0111110) event counter is defined to count non-cacheable instruction or data memory read bus cycles. Reads to I/O memory space are not supposed to be counted. However, the counter incorrectly gets incremented for reads to I/O memory space.

The “Instructions executed” (event 010110) and “Instructions executed in the v-pipe” (event 010111) event counters are both supposed to be incremented when any exception is recognized. However, if the instruction in the v-pipe generates an exception and a second exception occurs before execution of the first instruction of the exception handler for the first exception, the counter incorrectly does not get incremented for the first exception.

The “Stall on write to an E or M state line” (event 011011) event counter counts the number of clocks the processor is stalled on a data memory write hit to an E or M state line in the internal data cache while either the write buffers are not empty or EWBE# is not asserted. However, it does not count stalls while the write buffers are not empty, it only counts the number of clocks stalled while EWBE# is not asserted.

The “Code TLB miss” (event 001101) and “Data TLB miss” (event 000010) event counters incorrectly get incremented twice if the instruction that misses the code TLB or the data that misses the data TLB also causes an exception.

The “Data read miss” (event 000011) and “Data write miss” (event 000100) event counters incorrectly get incremented twice if the access to the cache is misaligned.

The “Bank conflicts” (event 001010) event counter may be incremented more than once if a v-pipe access takes more than 1 clock to execute.

The “Misaligned data memory or I/O References” (event 001011) incorrectly gets incremented twice if the access was caused by a FST or FSTP instruction.

The “Pipeline flushes” (event 010101) event counter may incorrectly be incremented for some segment descriptor loads and the VERR instruction.

The “Pipeline stalled waiting for data memory read” (event 011010) event counter incorrectly counts a misaligned access as 2 clocks instead of 3 clocks, unless it misses the TLB.

Implication: The event monitor counters report an inaccurate count for certain events.

Workaround: None identified at this time.

Status: For the steppings affected see the Summary Table of Changes.

45. VERR Type Instructions Causing Page Fault Task Switch with T Bit Set May Corrupt CS:EIP

Problem: This erratum can only occur during debugging with the T bit set in the Page Fault Handler’s TSS. It requires the following very specific sequence of events:

1. The descriptor read caused by a VERR type instruction must trigger a page fault. (These instructions are VERR, VERW, LAR and LSL. They each use a selector to access the selected descriptor and perform some checks on it.)
2. The OS must have the page fault handler set up as a separate task, so the page fault causes a task switch.
3. The T bit in the page fault handler’s TSS must be set, which would normally cause a branch to the interrupt 1 (debug exception) handler.
4. The interrupt 1 handler must be in a not present code segment.

The not present code segment should cause a branch to interrupt 11. However, because of this erratum, execution begins at an invalid location selected by the CS from the page fault handler TSS but with the EIP value pointing to the instruction just beyond the VERR type instruction.

Implication: This erratum will only be seen by software or hardware developers setting the T bit in the page fault handler’s TSS for debugging. It requires that the OS in use has the page fault handler set up as a separate task, which is not done in any standard OS. Even when these conditions are met, the other conditions will cause this erratum to occur only infrequently. When it does occur, the processor will execute invalid or erroneous instructions. Depending on software and system configuration, the developer will typically see an application error message or system reset.

Workaround: If debugging a system in which the page fault handler is a separate task, use one of the following:

1. Do not set the T bit in the page fault handler’s TSS.
2. Ensure that the code segment where the debug exception handler starts is always present in the system memory during debugging.

Status: For the steppings affected see the Summary Table of Changes.

46. BUSCHK# Interrupt Has Wrong Priority

Problem: Section 2.7 of the *Pentium® Processor Family Developer’s Manual* lists the priorities of the external interrupts, with BUSCHK# as the highest (if the BUSCHK# interrupt, AKA the machine check exception, is enabled by setting the MCE bit in CR4), and INTR as the lowest. It is also specified that STPCLK# is the very lowest priority external interrupt for those Pentium processors provided with it (all CPUs with a core frequency of 75 MHz and above). Consistently with this specification, the CPU blocks all other external interrupts once execution of the BUSCHK# exception handler begins.

However this erratum can change the effective priority for a given assertion of BUSCHK# in the following cases:

CASE 1:An additional external interrupt (except INTR) or a debug exception occurs during a narrow window after the CPU begins to transfer control to the BUSCHK# handler, but before the first instruction of the handler begins execution.

In this case, the other interrupt may be serviced before BUSCHK# is serviced. Thus for other interrupts that occur during this narrow window, BUSCHK# is effectively treated as the next to lowest priority interrupt instead of the highest.

CASE 2: The following conditions must all apply for this case to cause an erratum:

1. A machine check request (INT 18) is pending
2. A FLUSH# or SMI# request is pending
3. A single step or data breakpoint exception (INT 1) is pending
4. The IO_Restart feature is enabled (i.e., TR12 bit 9 is set)

Given the above set of conditions, the interrupt priority logic does not recognize the machine check exception as the highest priority. The processor will not service the FLUSH#/SMI# nor the debug exception (INT 1). Instead, it will generate an illegal opcode exception (INT 6).

Implication: Most systems do not use BUSCHK# and thus are unaffected by this erratum. For those that do use BUSCHK#, the pin allows the system to signal an unsuccessful completion of a bus cycle. This would only occur in a defective system. (Since BUSCHK# is an “abort” type exception, it cannot be used to handle a problem from which the OS intends to recover; BUSCHK# always requires a system reset.)

Due to this erratum, the BUSCHK# interrupt would either occasionally be displaced by another interrupt (which incorrectly would be serviced first) or an unexpected illegal opcode exception (INT 6) would be generated and the pending machine check would be skipped.

Depending on the system and also the severity of the defect, this delay of the BUSCHK# interrupt (case #1 above) could cause a system hang or reset before a bus cycle error message is displayed by the BUSCHK# interrupt. In case #2 above where an illegal opcode exception (INT 6) is generated instead of the machine check exception, a properly architected INT 6 handler will usually require a reset since this handler was erroneously entered without an illegal opcode. But in any event, the normal outcome of a bus cycle error is to require a system reset, so the practical result of this erratum is just the occasional loss of the proper error message in a defective system.

Another problem can occur due to this erratum if the system is using the SMM I/O instruction restart feature. This problem requires an improbable coincidence: the SMI# signal caused by an I/O restart event must occur essentially simultaneously with BUSCHK#, such that the SMI# interrupt hits the narrow window (as described above) just before the first instruction of the BUSCHK# handler begins execution. This could happen if the same I/O instruction that triggers SMI# (usually to turn back on a device that’s been turned off to save power) also generates a bus failure due to the system suddenly going defective, thus signaling BUSCHK#. The result is that the SMI# interrupt is serviced after the EIP has already been switched to point to the first instruction of the BUSCHK# handler, instead of the I/O instruction. The SMM code that services the I/O restart feature may well use the image of EIP in the SMRAM state save memory to inspect the I/O instruction, for example to determine what I/O address it’s trying to access. In this case, the I/O restart part of SMM code will not find the correct instruction. If it is well written, it will execute RSM when it determines

there is no valid I/O access to service. Then execution returns to the BUSCHK# handler with no deleterious impact. But less robust code might turn on the wrong I/O device, hang up, or begin executing from a random location.

Workaround: Do not design a system which relies on BUSCHK# as the highest priority interrupt. If using SMM, do not use BUSCHK# at all.

Note that Case 2 does not apply to B1, C1 or D1 steppings of the 60- and 66-MHz Pentium processors.

Status: For the steppings affected see the Summary Table of Changes.

47. BF and CPUTYP Buffers Sample Incorrectly During Boundary Scan Testing

Problem: During boundary scan input testing, the boundary scan input paths in the BF0, BF1 and CPUTYP buffers are disabled when RESET is **low**. (Note that this is different from the BSDL testing problem with STPCLK#, documented as Erratum 41; STPCLK# is "stuck" when RESET is **high**.)

Implication: The boundary scan cells in the BF0, BF1 and CPUTYP buffers capture a "1" from their pins, regardless of the actual data on the pins, when RESET is low. This violates the IEEE specification 1149.1 which states that the value captured should always be that on the pins regardless of the state of RESET. However, the buffer functions correctly when the EXTEST instruction is used.

Workaround: If testing with boundary scan in a system environment these pins can be left untested by marking them "INTERNAL" in the BSDL file.

Status: For the steppings affected see the Summary Table of Changes.

48. Matched But Disabled Data Breakpoint Can Be Lost By STPCLK# Assertion

Problem: Assertion of STPCLK# can interfere with a feature described in the *Intel Architecture Software Developer's Manual*, Volume 3, Section 14.2.3: "The processor sets the DR6 B bits for all breakpoints which match the conditions present at the time the debug exception is generated, whether or not they are enabled." When the debug exception is generated, all breakpoints which match the conditions present at that time are flagged by a bit set in a temporary register. If STPCLK# is asserted after this, but before control is transferred to the debug exception handler (interrupt 1), a matched but disabled data breakpoint may not be transferred from the temporary register. That is, as a result of the STPCLK# assertion, the B bit corresponding to that breakpoint may not get set in DR6.

Implication: This feature (defining disabled breakpoints) can be used in debugging; e.g., one can set a disabled data breakpoint on a memory location and then check the corresponding bit in DR6, to see if the location has been accessed by the most recent (main code) instruction, any time one is in the debug handler for some other reason. This erratum will sometimes cause this debug feature to fail to set its DR6 bit, when STPCLK# is also being used.

Workaround: Use one of the following:

1. Use only *enabled* data breakpoints when STPCLK# may be asserted.
2. Disable the assertion of STPCLK# while this debug feature is being used.

Status: For the steppings affected see the Summary Table of Changes.

49. STPCLK# Ignored In SMM When INIT or NMI Pending

Problem: If an INIT or NMI is pending while in SMM mode, and STPCLK# is asserted, the stop clock interrupt is not serviced. The correct operation is for the stop clock request to be serviced while in SMM, regardless of pending NMI or INIT.

Implication: The stop clock request is blocked until after the processor exits SMM and services the pending NMI or INIT. The processor then services the lower priority stop clock interrupt.

Workaround: None identified at this time.

Status: For the steppings affected see the Summary Table of Changes.

50. STPCLK# Pullup Not Engaged at RESET

Problem: The internal pullup on the STPCLK# pin may not pullup at power on if the pin is floating at a low input level.

Implication: If the STPCLK# pin is floating at a low input level and the pin is left unconnected at bootup, the processor may initiate the stop grant bus cycle in response to the STPCLK# request shortly after completing the reset sequence. This may result in a system hang.

Workaround: Use one of the following:

1. Always drive a valid logic level on STPCLK# (including during RESET).
2. Use an appropriately sized external pullup resistor.

Status: For the steppings affected see the Summary Table of Changes.

51. A Fault Causing a Page Fault Can Cause an Instruction To Execute Twice

Problem: When the processor encounters an exception while trying to begin the handler for a prior exception, it should be able to handle the two serially (i.e., the second fault is handled and then the faulting instruction is restarted, which causes the first fault again, whose handler now should begin properly); if not, it signals the double-fault exception. A “contributory” exception followed by another contributory exception causes the double-fault, but a contributory exception followed by a page fault are both handled. (See the *Intel Architecture Software Developer’s Manual*, Volume 3, Section 5.12, Interrupt 8 for the list of contributory exceptions and other details.) This erratum occurs under the following circumstances:

1. One of these three contributory faults: #12 (stack fault), #13 (General Protection), or #17 (alignment check), is caused by an instruction in the v-pipe.
2. Then a page fault occurs before the first instruction of the contributory fault handler is fetched. (This means that a page fault that occurs because the handler starts in a not present page will *not* cause this erratum.)

The result is that execution correctly branches to the page fault handler, but *an incorrect return address is pushed on the stack*: the address of the (immediately preceding) u-pipe instruction, instead of the v-pipe instruction that caused the faults. This causes the u-pipe instruction to be executed an extra time, after the page fault handler is finished.

Implication: When this erratum occurs, an instruction will be (incorrectly) executed, effectively, twice in a row. For many instructions (e.g., MOV, AND, OR) it will have no effect, but for some instructions it can cause an incorrect answer (e.g., ADD would increase the destination by double the correct amount). However, the page fault (during transfer to the handler for fault #12, #13 or #17) required for this erratum to occur can happen in only three unusual cases:

1. If the alignment check fault handler is placed at privilege level 3, the push of the return address could cause a page fault, thus causing this erratum. (Fault #17 can only be invoked from level 3, so it is legal to have its handler at level 3. Fault 12 and 13 handlers must always be at level 0 since they can be invoked from level 0. The push of a return address on the level 0 stack *must not* cause a page fault, because if the OS allowed that to happen, the push of return address for a regular page fault could cause a second page fault, which causes a double-fault and crashes the OS.)
2. If the descriptor for the fault handler’s code segment (in either the GDT or the current LDT) is in a not present page, a page fault occurs which causes this erratum.

3. If the OS has defined the fault handler as a separate task, and a page fault occurs while bringing in the new LDT or initial segments, this erratum will occur.

Workaround: All of the following steps must be taken (but 2 & 3 are part of normal OS strategy, done in order to optimize speed of access to key OS elements, and minimize chances for bugs): 1). If allowing the alignment fault (#17), place its handler at level 0. 2). Do not allow any of the GDT or current LDT to be "swapped out" during virtual memory management by paging. 3). Do not use a separate task for interrupts 12, 13 or 17.

Status: For the steppings affected see the Summary Table of Changes.

52. Machine Check Exception Pending, then HLT, Can Cause Skipped or Incorrect Instruction, or CPU Hang

Problem: This erratum can occur if a machine check exception is pending when the CPU encounters a HLT instruction, or occurs while the CPU is in the HLT state. (the BUSCHK# error could be caused by executing the previous instruction, or by a code prefetch.) Before checking for pending interrupts, the HLT instruction issues its special bus cycle, and sets an special internal flag to indicate that the CPU is in the HLT state. The machine check exception (MCE) can then be detected, and if it is present the CPU branches to the MCE handler, but *without clearing the special HLT flag - the source of this erratum.* As when other interrupts break into HLT, the return address is that of the next instruction after HLT, so execution continues there after return from the MCE handler.

Except for MCE (and some cases of the debug interrupt), interrupts clear the special HLT flag before executing their handlers. The erratum that causes the MCE logic to not clear the HLT flag in this case can have the following consequences:

1. If NMI, or INTR if enabled, occurs while the HLT flag is set, the CPU logic assumes the instruction immediately following the interrupt is an HLT. So it places the address of the instruction after that on the stack, which means that upon return from the interrupt, the instruction immediately following the interrupt occurrence is skipped over.
2. If FLUSH # is asserted while the HLT flag is set, the CPU flushes the L1 cache and then returns to the HLT state. If the CPU is extracted from the HLT state by NMI or INTR, as in 1), the CPU logic assumes that the current CS:EIP points to an HLT instruction, and pushes the address of the *next* instruction on the stack, so the instruction immediately following the FLUSH# assertion is skipped over.
3. If RSM is executed while the HLT flag is set, again the CPU logic assumes that the CPU must have been interrupted (by SMI, in this case) while in the HLT state. Normally, RSM would cause the CPU to branch back to the instruction that was aborted when entering SMM. But in this case, the CPU branches to the address of the *next* instruction minus one byte. If the aborted instruction is one byte long, this is fine. If it is longer, the CPU executes effectively a random opcode: the last byte of the aborted instruction is interpreted as the first byte of the next instruction.

Implication: In cases 1 and 2, skipping an instruction can have no noticeable effect, or it could cause some obvious error condition signaled by a system exception, or it could cause an error which is not easily detected. In case 3, executing a random opcode is most likely to cause a system exception like #6 (invalid opcode), but it could cause either of the other results as with cases 1 and 2. Case 2 can also cause an indefinite CPU hang, if the problem occurred when INTR was disabled. However, in order to encounter any of these problems, the system has to continue on with program execution after servicing the MCE. Since the MCE is an abort type exception, the handler for it cannot rely on a valid return address. Also MCE usually signals a serious system reliability problem. For both these reasons, the usual protocol is to require a system reset to terminate the MCE handler. If this usual protocol is followed successfully, it will clear the HLT flag and thus always prevent the above problems. However, there is an additional complication: the cases 1, 2 and 3 above can occur *inside* the MCE handler, possibly preventing its completion.

Workaround: The problems caused by this erratum will be prevented if the Machine Check Exception handler (if invoked) always forces a CPU RESET or INIT (which it should do anyway, for reasons given above). Since the problems can occur *inside* the MCE handler, the IF should be left zero to prevent INTR from interrupting. Also, NMI, SMI and FLUSH could be blocked inside the MCE handler. The most secure strategy is to force INIT immediately upon entrance to the MCE handler.

Status: For the steppings affected see the Summary Table of Changes.

53. FBSTP Stores BCD Operand Incorrectly If Address Wrap & FPU Error Both Occur

Problem: This erratum occurs only if a program does all of the following:

1. The program uses 16 bit addressing inside a USE32 segment (requiring the 67H addressing override prefix) in order to wrap addresses at offsets above 64K back to the bottom of the segment.
2. The 10 byte BCD operand written to memory by the FBSTP instruction must actually straddle the 64K boundary. If all 10 bytes are either above or below 64K, the wrap works normally.
3. The FBSTP instruction whose operand straddles the boundary must also generate an FPU exception. (e.g., Overflow if the operand is too big, or Precision if the operand must be rounded, to fit the BCD format.)

The result is that some of the 10 bytes of the stored BCD number will be located incorrectly if there is an FPU exception. They will be nearby, in the same segment, so no protection violation occurs from this erratum.

The erratum is caused by the fact that when an FPU exception occurs due to FBSTP, a different internal logic sequence is used by the CPU, which sends the bytes to memory in different groupings. Normally this does not affect the result, but when address wrap occurs in the middle of the operand, the different groupings can cause different destination addresses to be calculated for some bytes.

Implication: Code which relies on this address wrap with a straddled FBSTP operand may not store the operand correctly if FBSTP also generates an FPU exception. Intel recommends not to straddle segment or addressing boundaries with operands for several reasons, including (see the *Intel Architecture Software Developer's Manual*, Volume 3, Chapter 17) the chance of losing data if a memory fault interrupts an access to the operand. Also there is variation between generations of Intel processors in how straddled operands are handled.

Workaround: Use one of the following:

Do not use 16 bit addressing to cause wraps at 64K inside a USE32 segment.

Follow Intel's recommendation and do not straddle an addressing boundary with an operand.

Status: For the steppings affected see the Summary Table of Changes.

54. V86 Interrupt Routine at Illegal Privilege Level Can Cause Spurious Pushes to Stack

Problem: By architectural definition, V86 mode interrupts must be executed at privilege level 0. If the target CPL (Current Privilege Level) in the interrupt gate in the IDT (Interrupt Descriptor Table) and the DPL (Descriptor Privilege Level) of the selected code segment are not 0 when an interrupt occurs in V86 mode, then interrupt 13 (GP fault) occurs. This is described in the *Intel Architecture Software Developer's Manual*, Volume 3, Section 15.3. The architectural definition says that execution transfers to the GP fault routine (which must be at level 0) with nothing done at the privilege level (call it level N) where the interrupt service routine is illegally located. In fact (this erratum) the Pentium® Processor incorrectly pushes the segment registers GS and FS on the stack at

level N, before correctly transferring to the GP fault routine at level 0 (and pushing GS and FS again, along with all the rest that's specified for a V86 interrupt).

Implication: When this erratum occurs, it will place a few additional bytes on the stack at the level (1, 2 or 3) where the interrupt service routine is illegally located. If the stack is full or does not exist, the erratum will cause an unexpected exception. But this problem will have to be fixed during the development process for a V86 mode OS or application, because otherwise the interrupt service routine can never be accessed by V86 code. Thus this erratum can only be seen during the debugging process, and only if the software violates V86 specifications.

Workaround: Place all code for V86 mode interrupt service routines at privilege level 0, per specification.

Status: For the steppings affected see the Summary Table of Changes.

55. Corrupted HLT Flag Can Cause Skipped or Incorrect Instruction, or CPU Hang

Problem: The Pentium processor sets an internal HLT flag while in the HLT state. There are some specific instances where this HLT flag can be incorrectly set when the CPU is not in the HLT state.

1. A POP SS which generates a data breakpoint, and is immediately followed by a HLT. Any interrupt which is pending during an instruction which changes the SS, is delayed until after the next instruction (to allow atomic modification of SS:ESP). In this case, the breakpoint is therefore correctly delayed until after the HLT instruction is executed. The processor waits until after the HLT cycle to honor the breakpoint, but in this case when the processors branches to the interrupt 1 handler, it fails to clear the HLT flag. The interrupt 1 handler will return to the instruction following the HLT, and execution will proceed, but with the HLT flag erroneously set.
2. A code breakpoint is placed on a HLT instruction, and an SMI# occurs while processor is in the HLT state (after servicing the code breakpoint). The SMI handler usually chooses to RSM to the HLT instruction, rather than the next one, in order to be transparent to the rest of the system. In this case, on returning from the SMI# handler, the code breakpoint is typically re-triggered (SMI# handler does not typically set the RF flag in the EFLAGS image in the SMM save area). The processor branches to the interrupt 1 handler again, but without clearing the HLT flag. The interrupt 1 handler will return to the instruction following the HLT, and execution will proceed, but with the HLT flag erroneously set.
3. A machine check exception just before, or during, a HLT instruction can leave the HLT flag erroneously set. This is described in detail in erratum #52: *Machine Check Exception Pending, then HLT, Can Cause Skipped or Incorrect Instruction, or CPU Hang.*

Implication: For cases 1 and 2, the CPU will proceed with the HLT flag erroneously set. The following problematic conditions may then occur.

a. If NMI, or INTR if enabled, occurs while the HLT flag is set, the CPU logic assumes the instruction immediately following the interrupt is a HLT. It therefore places the address of the instruction after that on the stack, which means that upon return from the interrupt, the instruction immediately following the interrupt occurrence is skipped over.

b. If FLUSH # is asserted while the HLT flag is set, the CPU flushes the L1 cache and then incorrectly returns to the HLT state, which will hang the system if INTR is blocked (IF = 0) and NMI does not occur. If the CPU is extracted from the HLT state by NMI or INTR, as in a), the CPU logic assumes that the current CS:EIP points to an HLT instruction, and pushes the address of the *next* instruction on the stack, so the instruction immediately following the FLUSH# assertion is skipped over.

c. If RSM is executed while the HLT flag is set, again the CPU logic assumes that the CPU must have been interrupted (by SMI#, in this case) while in the HLT state. Normally, RSM would cause the CPU to branch back to the instruction that was aborted when entering

SMM. But in this case, the CPU branches to the address of the *next* instruction minus one byte. If the aborted instruction is one byte long, this is fine. If it is longer, the CPU executes effectively a random opcode: the last byte of the aborted instruction is interpreted as the first byte of the next instruction.

d. If STPCLK# is asserted to the CPU while the HLT flag is incorrectly set, the CPU will hang such that a CPU reset is required to continue execution.

Cases 1 and 2 of this erratum occur only during code development work, and only with the unusual combination of data breakpoint triggered by POP SS followed by HLT or code breakpoint on HLT followed by SMI#.

Workaround: CASE 1: Avoid following POP SS with a HLT instruction. POP SS should always be followed by POP ESP anyway, to finish switching stacks without interruption. Following POP SS with HLT instead would normally be a program logic error (the interrupt that breaks the CPU out of HLT will not have a well defined stack to use).

CASE 2: Do not place code breakpoints on HLT instructions. Or: Modify the SMI# handler slightly for debugging purposes by adding instructions to set the RF flag in the EFLAGS image in the SMM save area.

Status: For the steppings affected see the Summary Table of Changes.

56. Benign Exceptions Can Erroneously Cause Double Fault

Problem: The double-fault counter can be incorrectly incremented in the following cases:

CASE 1: An instruction generates a benign exception (for example, a FP instruction generates an INT 7) and this instruction causes a segment limit violation (or is paired with a v-pipe instruction which causes a segment limit violation)

CASE 2: A machine check exception (INT 18) is generated.

The initial benign exception will be serviced properly. However, if while trying to begin execution of the benign exception handler, the processor gets an additional contributory exception, the processor will trigger a double fault (and start to service the double fault handler) instead of servicing the new contributory fault. (See Table 5-3 in the *Intel Architecture Software Developer's Manual*, Volume 3 for a complete list of benign/contributory exceptions).

Implication: Contributory exceptions generated while servicing benign exceptions can erroneously cause the processor to execute the double fault handler instead of the contributory exception handler.

Workaround: Use benign exception handlers that do not generate additional exceptions. Operating systems designed such that benign exception handlers do not generate additional exceptions will be immune to this erratum. In general, most operating system exception handlers are architected accordingly.

Status: For the steppings affected see the Summary Table of Changes.

57. Double Fault Counter May Not Increment Correctly

Problem: In some cases a double fault exception is not generated when it should have been because the internal double fault counter does not correctly get incremented.

When the processor encounters a contributory exception while attempting to begin execution of the handler for a prior contributory exception (for example, while fetching the interrupt vector from the IDT or accessing the GDT/LDT) it should signal the double fault exception. Due to this erratum, however, the CPU will incorrectly service the new exception instead of going to the double fault handler.

In addition, if the first contributory fault is the result of an instruction executed in the v-pipe, a second contributory fault will cause the processor to push an incorrect EIP onto the stack before entering the second exception handler. Upon completion of the second exception handler, this incorrect EIP gets popped from the stack and the processor resumes execution from the wrong address.

Implication: The processor could incorrectly service a second contributory fault instead of going to the double fault handler. The resulting system behavior will be operating system dependent. Additionally, an inconsistent EIP may be pushed on to the stack.

Robust operating systems should be immune to this erratum because their exception handlers are designed such that they do not generate additional contributory exceptions. This erratum is only of concern during operating system development and debug.

Workaround: Use contributory exception handlers that do not generate additional contributory exceptions. Operating systems which are designed such that their contributory exception handlers do not generate additional contributory exceptions will not be affected by this erratum. In general, most operating system exception handlers are architected accordingly.

Status: For the steppings affected see the Summary Table of Changes.

58. Some Input Pins May Float High Erroneously When Core VCC Powers Up After I/O VCC (Mobile CPU)

Problem: Unused input signals are typically tied to either VCC or VSS. Low inputs can be provided with a hard V_{SS} strap or a pulldown resistor. If any of the input pins AHOLD, KEN#, WB/WT#, NA#, INV, BRDY#, or EWBE# are not driven by system logic, and are tied to ground via a weak pulldown resistor (i.e., >2 KOhms), and CPU I/O power supply (VCC3) ramps before CPU core power supply (VCC2), these input pins may float high and be erroneously latched high by the processor during boot. The effect of this erratum depends on the usage of each pin. For example, if EWBE# gets latched high, the processor may hang indefinitely.

Implication: The input pins AHOLD, KEN#, WB/WT#, NA#, INV, BRDY#, or EWBE# may register a false start up state. In some cases, the processor may erroneously hang while waiting for an input response. For example, the EWBE# being sampled high may cause the system to hang while waiting for the processor to sample EWBE# low.

Workaround: If the signal is not driven by system logic and is pulled low, a pulldown resistor of 2K Ohms or less should be used to guarantee logic level zero.

Status: For the steppings affected see the Summary Table of Changes.

59. Short Form of MOV EAX/ AX/ AL May Not Pair

Problem: The MOV data instruction forms (excluding MOV using Control, Debug or Segment registers) are intended to be pairable, unless there is a register dependency between the two instructions considered for pairing. (e.g., MOV EAX, mem1 followed by MOV mem2, EAX: here the 2nd instruction cannot be completed until after the first has put the new value in EAX.) This pairing for MOV data is documented by the UV symbol in the Pairing column in the table of Pentium processor instruction timings in the *Optimizations for Intel's 32-Bit Processors* application note (Order # 243195). This erratum is that the instruction unit under some conditions fails to pair the special short forms of MOV mem, EAX /AX /AL, when no register dependency exists.

The Intel Architecture includes special instructions to MOV EAX /AX /AL to a memory offset (opcodes 0A2H & 0A3H). These instructions don't have a MOD/RM byte (and so are shortened by one byte). Instead, the opcode is followed immediately by 1/2/4 bytes giving the memory offset (displacement). This erratum occurs specifically when a MOV mem, EAX /AX /AL instruction using opcode 0A2H or 0A3H is followed by an instruction that uses the EAX /AX /AL register as a

source (register source, or as base or index for the address of a memory source) or a destination register. Then the instruction unit detects a (false) dependency and it doesn't allow pairing. For example, the following two instructions are not paired:

```
A340000000   MOV DWORD PTR 40H, EAX ;  memory DS:[40H] <- EAX  [goes into u-pipe ]
A160000000   MOV EAX, DWORD PTR 60H ;  EAX <- memory DS:[60H]  [does NOT go into
v-pipe]
```

Implication: The only result of this erratum is a very small performance impact due to the non-pairing of the above instructions under the specified conditions. The impact was evaluated for SPECint92* and SPECfp92* and was estimated to be much smaller than run-to-run measurement variations.

Workaround: For the Pentium processor, use the normal MOV instructions (with the normal MOD/RM byte) for EAX /AX /AL instead of the short forms, when writing optimizing compilers and assemblers or hand assembling code for maximum speed. However, as documented above, the performance improvement from avoiding this erratum will be quite small for most programs.

Status: For the steppings affected see the Summary Table of Changes.

60. Turning Off Paging May Result In Prefetch To Random Location

Problem: When paging is turned off a small window exists where the BTB has not been flushed and a speculative prefetch to a random location may be performed. The *Intel Architecture Software Developer's Manual*, Volume 3, Section 8.8.2, lists a sequence of nine steps for switching from protected mode to real-address mode. Listed here is step 1.

1. If paging is enabled, perform the following sequence:
   - — Transfer control to linear addresses which have an identity mapping (i.e., linear addresses equal physical addresses). Ensure the GDT and IDT are identity mapped.
   - — Clear the PG bit in the CR0 register.
   - — Move zero into the CR3 register to flush the TLB.

With paging enabled, linear addresses are mapped to physical addresses using the paging system. In step a above the executing code transfers control to code located where the linear addresses are mapped directly to physical addresses. Step b turns off paging followed by step c which writes zero to CR3 which flushes the TLB (and BTB). A small window exists (after clearing the PG bit and before zeroing CR3) where the BTB has not been flushed, and a BTB hit may cause a prefetch to an unintended physical address.

Implication: A prefetch to an unintended physical address could potentially cause a problem if this prefetch was to a memory mapped I/O address. If reading a memory mapped I/O address changes the state of a memory mapped I/O device, this unintended access may cause a system problem.

Workaround: Flush the BTB just before turning paging off. This can be done by reading the contents of CR3 and writing it back to CR3 prior to clearing the PG bit in CR0.

Status: For the steppings affected see the Summary Table of Changes.

61. REVISED ERRATUM: STPCLK#, FLUSH# or SMI# After STI

Problem: The STI specification says that external interrupts are enabled at the end of the next instruction after STI. However, external interrupts may be enabled before the next instruction is executed following STI if a STPCLK#, FLUSH# or SMI# is asserted and serviced before the instruction boundary of this next instruction.

Implication: External interrupts which are assumed blocked until after the instruction following STI may be recognized before this instruction executes.

Workaround: None identified at this time.

Status: For the steppings affected, see the Summary Table of Changes.

62. REP String Instruction Not Interruptible by STPCLK#

Problem: The *Intel Architecture Software Developer's Manual*, Volume 2, Chapter 3 under the REP string instruction, states that any pending interrupts are acknowledged during a string instruction. On the Pentium processor there is one exception. STPCLK# is not able to interrupt a REP string instruction. It is only recognized on an instruction boundary (as stated in Volume 1, Section 21.1.36). However, if any other interrupt is recognized during a REP string instruction, this will allow STPCLK# to be serviced before returning to execution of the REP string instruction.

Implication: A system that uses stop clock frequently can not interrupt the REP string instruction in the middle and must wait until it completes or another interrupt is recognized before STPCLK# is recognized. Note that in standard PC-AT architecture, the real time clock interrupt will interrupt a long string instruction allowing STPCLK# to be recognized.

Workaround: None identified at this time.

Status: For the steppings affected see the Summary Table of Changes.

63. Single Step May Not be Reported on First Instruction After FLUSH#

Problem: The single step trap should cause an exception to occur upon completion of all instructions. However, in some cases when ITR (bit 9 of TR12) = '1', a single step exception may not be reported for the first instruction following FLUSH#. The Single Step exception will be skipped for this instruction. Note that subsequent single step exceptions will be reported correctly.

Implication: A single step breakpoint may be missed when a FLUSH# request is presented to the processor. This erratum will only affect software developers while debugging code.

Workaround: None identified at this time.

Status: For the steppings affected see the Summary Table of Changes.

64. Double Fault May Generate Illegal Bus Cycle

Problem: A double fault condition may generate an illegal bus cycle (a cacheable line-fill with a lock attribute). This scenario is caused by following sequence of events:

1. A contributory fault occurs.
2. The processor begins to service this fault by reading the appropriate trap/interrupt gate from the IDT. However, this gate points to a segment descriptor (in the GDT) whose "accessed" bit is not set.
3. The segment descriptor is modified and marked "not present/not valid" by another processor in the system
4. A locked Read-Modify-Write cycle is generated to update the "accessed" bit.

The erratum condition is encountered if the segment descriptor was modified and marked "not present/not valid" by another processor in the system before the locked read cycle (step #4 above). The processor will begin to execute the locked read. Since the descriptor is marked invalid, the processor should go to the exception handler to service a specific exception and clear the bus-lock (through a write operation). However, since a contributory fault has already occurred, the processor will interpret this condition as a double fault. The double fault logic incorrectly generates a cacheable line-fill with a lock attribute.

Implication: This erratum can only occur in DP and MP systems.

Cacheable line-fills with a lock attribute are "illegal" bus cycles. Exact operation under this condition is chipset dependent. It may cause the system to hang.

Note that this erratum will only occur in the case of a double fault, which are rare events for well architected operating systems. Also, the double fault condition is not generally recoverable, implying that the system will need to be rebooted anyway.

Finally, this erratum can only happen on the first pass through the interrupt handler. After that, the "accessed" bit of the code descriptor will be set, eliminating a prerequisite for occurrence of this erratum.

Workaround: None identified at this time.

Status: For the steppings affected see the Summary Table of Changes.

65. TRST# Not Asynchronous

Problem: TRST# is not an asynchronous input as specified in Section 5.1.67 of the *Pentium® Processor Family Developer's Manual*.

Implication: TRST# will not be recognized in cases where it does not overlap a rising TCK# clock edge. This violates the IEEE 1149.1 specification on Boundary Scan.

Workaround: TRST# should be asserted for a minimum of two TCK periods to ensure recognition by the processor.

Status: For the steppings affected see the Summary Table of Changes.

66. STPCLK# on RSM to HLT Causes Non-Standard Behavior

Problem: This problem will occur if STPCLK# is asserted during the execution of an RSM instruction which is returning from SMM to a HALT instruction (Auto HALT restart must be enabled for this to happen). The RSM instruction will be completed, and then the CPU correctly issues, in response to the STPCLK# assertion, a Stop Grant special cycle, and goes into the Stop Grant state. However, following this, behavior occurs which deviates from the CPU specifications in one of two ways, depending on whether STPCLK# is de-asserted before any (enabled) external interrupt occurs (Case 1), or an (enabled) external interrupt occurs while STPCLK# is still active (Case 2).

CASE 1: After STPCLK# is de-asserted, no HALT special cycle is issued, and the CPU effectively stays in the Stop Grant state until an external interrupt is asserted (to which the CPU responds normally). However, a HALT cycle **should** be issued when STPCLK# is de-asserted, because it is stated that a HALT cycle will be issued upon an RSM to the HALT state.

CASE 2: When the (enabled) external interrupt is asserted while STPCLK# is still active, the CPU should remain in the Stop Grant state. But when the conditions have been met for this erratum, the CPU comes out of the Stop Grant state and starts the internal interrupt service process. This includes issuing the interrupt acknowledge cycles, reading the selected entry from the interrupt descriptor table, and fetching the first instruction of the requested interrupt service routine (I.S.R.). However, before the CPU executes that first instruction, STPCLK# is recognized again, execution halts, and a Stop Grant cycle is issued. The erratum condition is cleared by one of the steps which the CPU performs to prepare for the I.S.R., so any further interrupts (while STPCLK# remains asserted) will not remove the CPU from the Stop Grant state. When STPCLK# is de-asserted, the CPU begins executing the requested I.S.R.

Implication: CASE 1: The absence of the usual HALT special cycle upon a RSM to a HLT instruction in this rare case should have no impact, unless the system is looking for the HALT cycle after RSM and would normally make some response to it. The system will have received the HALT cycle upon initial entry to the HALT state. To expect another HALT cycle after RSM, the system would have to be tracking the fact that the SMI occurred during a HLT.

CASE 2: This case of the erratum means that some cycles preparatory to executing the I.S.R. are issued when the interrupt is received, rather than waiting until after STPCLK# is de-asserted. Also, an extra Stop Grant cycle is issued just after these premature cycles. However, all of the I.S.R. itself is executed at the correct time. This difference in the bus cycles has no known system implications.

Workaround: None required for any known implementations.

Status: For the steppings affected see the Summary Table of Changes.

67. Code Cache Dump May Cause Wrong IERR#

Problem: When using the test registers to read a cache line that is not initialized, the data array may indicate a wrong parity, which may cause IERR# to be asserted. It may also cause a shutdown.

Implication: A code cache dump through test registers may cause a parity check when reading an uninitialized cache entry, resulting in a shutdown.

Workaround: Set TR[1] to 1 to ignore IERR#, so that shutdown during a code cache dump can be avoided, or ensure that all cache lines have been initialized prior to a code cache dump.

68. Asserting TRST# Pin or Issuing JTAG Instructions Does not Exit TAP Hi-Z State

Problem: The *Pentium® Processor Family Developer's Manual*, Section 11.3.2.1 states that the TAP Hi-Z state can be terminated by resetting the TAP with the TRST# pin, by issuing another TAP instruction, or by entering the Test_Logic_Reset state. However, the indication that the processor has entered the TAP Hi-Z state is maintained until the next RESET. Therefore by using the above methods alone, the TAP Hi-Z state can not be terminated.

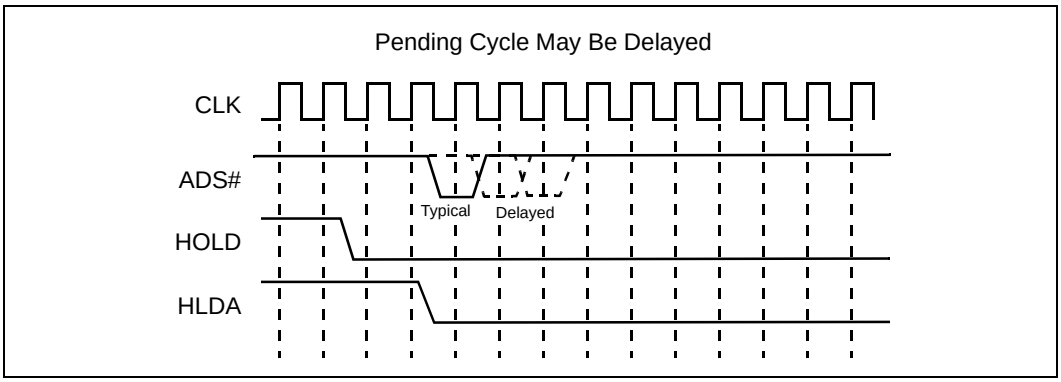
Implication: When the TAP Hi-Z instruction is enabled and executed, the processor may not terminate the Hi-Z state.

Workaround: To exit TAP Hi-Z state, in addition to the methods described above, the processor needs to be RESET as well.

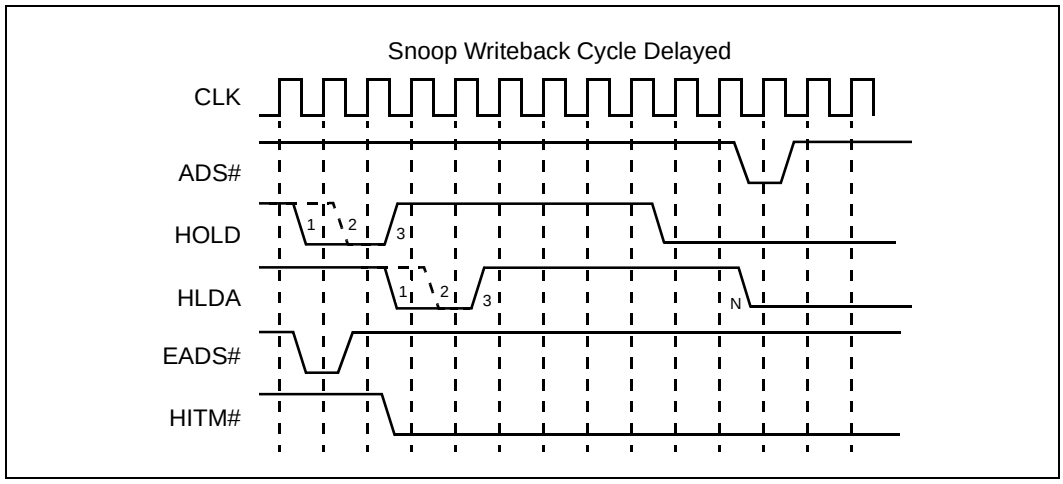
Status: For the steppings affected see the Summary Table of Changes.

69. ADS# May be Delayed After HLDA Deassertion

Problem: The Pentium processor typically starts a pending bus cycle on the same clock that HLDA is deasserted and the Pentium processor with MMX technology typically starts the cycle one clock after HLDA is deasserted. However, in both processors it may be delayed by as many as two clocks. See the diagram below:



In two cases, for example, if HOLD is deasserted for one clock (i.e., clock 2) or two clocks (i.e., clocks 1 & 2) and then reasserted, the window may not be large enough to start a pending snoop writeback cycle. The writeback cycle may be delayed until the HLDA is deasserted again (i.e., clock N). See diagram below.



Implication: If the system expects a cycle, for example a writeback cycle, and depends on this cycle to commence within the HLDA deassertion window, then the system may not complete the handshake and cause a hang.

Workaround:

1. Deassert HOLD for at least 3 clocks (i.e., clocks 1, 2, and 3 shown in figure) before reasserting HOLD again. This ensures that the Pentium processor initiates any pending cycles before reasserting HLDA.
2. If the system is waiting for the snoop writeback cycle to commence, for instance if HITM# is asserted, the system should wait for the ADS# before reasserting HOLD.

Status: For the steppings affected see the Summary Table of Changes.

70. Stack Underflow in IRET Gives #GP, Not #SS

Problem: The general Intel architecture rule about accessing the stack beyond either its top or bottom is that the stack fault error (#SS) will be generated. However, if during the execution of the IRET instruction there are insufficient bytes for an interlevel IRET to pop from the stack (stack underflow), the general protection (#GP) fault is generated instead of #SS.

This can only occur if the stack has been modified since the interrupt stored its return address, flags etc. such that there is no longer room on the stack for all of the stored information when IRET tries to access it. This would constitute a serious programming error that would cause problems more obvious than this erratum, and would normally be corrected during debugging. If this erratum did occur during regular execution of a program, the normal O/S response to a task causing either a #GP or #SS exception is to terminate the task, and so this erratum (#GP instead of #SS) would normally have no effect. If however the O/S is to be programmed to try to correct #GP and #SS problems and allow the task to continue execution, the workaround should be used.

Workaround: In order for the O/S code to correctly analyze this case of stack limit violation, the #GP code must include a test for stack underflow when #GP occurs during the IRET instruction.

Status: For the steppings affected see the Summary Table of Changes.

71. Performance Monitoring Pins PM[1:0] May Count The Events Incorrectly

Problem: The performance monitoring pins PM[1:0] can be used to indicate externally the status of event counters CTR1 and CTR0. While events are generated at the rate of the CPU clock, the PM[1:0] pins toggle at the rate of the I/O bus clock. However in some cases, the PM[1:0] pins may toggle twice when the event counters increment twice in one I/O clock, while in some cases, the PM[1:0] pins may toggle only once even when the event counters increment twice in two consecutive I/O clocks.

Implication: The performance monitoring pins PM[1:0] may not be relied upon to reflect the correct number of events that have occurred.

Workaround: None identified at this time.

Status: For the steppings affected see the Summary Table of Changes.

72. BIST Is Disabled

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

73. Branch Trace Messages May Cause System Hang

Problem: In a system with branch trace messages enabled, certain semaphore signaling sequences may cause the system to hang. Branch trace messages have the highest priority bus cycle in the Pentium processor with MMX technology, unlike previous Pentium processors, and take precedence over any other write cycle. A sequence of code where the processor writes to another processor or a controller, and then locks into a tight loop while waiting for the other processor or the controller to respond to the write, is susceptible to a hang, if branch trace messages are enabled. The problem is that the unending branch trace messages from the loop take priority over the previous write cycle. The write cycle never occurs and the other processor or the controller never responds. However, the processor will be pulled out of the hanging situation if an interrupt occurs.

Implication: This erratum only affects operation of the processor during instruction execution tracing which is normally only done during code development and debug. In addition, this erratum would typically only occur in an MP system, with short code sequences used for message passing. Also since interrupts pull the processor out of the hanging condition and they normally occur frequently, there should not be any noticeable system hang.

Workaround: Disable the branch trace message feature by setting TR12 bit 1 to 0 (the default is disabled).

Status: For the steppings affected see the Summary Table of Changes at the beginning of this section.

74. Enabling RDPMC in CR4 And Also Using SMM May Cause Shutdown

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

75. Event Monitor Counting Discrepancies (Fix)

Problem: The Pentium processor with MMX technology added several performance monitoring events to those defined in the Pentium processor (75/90/100/120/133/166/200). There are several conditions where the counters do not operate as specified.

The “Writes to non-cacheable memory” (event 101110) event counter counts the number of writes to non-cacheable memory including non-cacheable writes caused by MMX™ instructions. In some cases the counter fails to get incremented for a non-cacheable memory write caused by an MMX instruction.

The "Stall on MMX instruction write to an E or M state line" (event 111011) event counter counts the number of clocks the processor is stalled on a data memory write hit to an E or M state line in the internal data cache caused by a MMX instruction while either the write buffers are not empty or EWBE# is not asserted. However, it does not count stalls while the write buffers are not empty, it only counts the number of clocks stalled while EWBE# is not asserted.

Implication: The event monitor counters report an inaccurate count for certain events.

Workaround: None identified at this time. *See also* Errata 82 and 76.

Status: For the steppings affected see the Summary Table of Changes at the beginning of this section.

76. Event Monitor Counting Discrepancies (NoFix)

Problem: The Pentium processor with MMX technology added several performance monitoring events to those defined in the Pentium processor (75/90/100/120/133/166/200). There are several conditions where the counters do not operate as specified.

The "MMX instruction data read misses" (event 110001) and "MMX instruction data write misses" (event 110100) event counters get incorrectly incremented twice if the access to the cache is misaligned. The "Pipeline stalled waiting for MMX instruction data memory read" (event 110110) event counter incorrectly counts a misaligned access as 2 clocks instead of 3 clocks unless it misses the TLB.

The "MMX instruction multiply unit interlock" (event 111011) event counter counts the number of clocks the pipe is stalled because the destination of a previous MMX multiply instruction is not ready. However, if there is a multiply instruction followed by a branch instruction followed by a dependent multiply instruction, the counter incorrectly gets incremented when the branch is taken.

Implication: The event monitor counters report an inaccurate count for certain events.

Workaround: None identified at this time. *See also* Errata 82 and 75.

Status: For the steppings affected see the Summary Table of Changes at the beginning of this section.

77. INVD May Leave Valid Entries In The Cache Due To Snoop Interaction

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

78. TLB Update Is Blocked After A Specific Sequence Of Events With A Misaligned Descriptor

Problem: An obscure sequence of events may cause the TLB replacement mechanism to fail. This specific sequence must contain *all* of the following:

1. A specific setup of the data TLB: all 64 entries must be valid and one entry must contain the page where the IDT is located.
2. A REP-MOVS accesses a string that is at least 62-pages long.
3. A MOVS results in a GP fault in the 62nd page.
4. A gate in the IDT points to a descriptor and the descriptor is misaligned and crosses a page boundary.
5. The descriptor causes a TLB miss.

When the TLB miss discussed in condition 5 above occurs, the processor starts a split locked read-modify-write sequence to update the descriptor access or busy bit. During this split locked cycle, the address of the low bytes of the descriptor is loaded into a slot in the TLB. The address of the

high bytes of the descriptor is then put into the same slot of the TLB causing the address of the low bytes to be overwritten (this is caused by conditions 1-3 above). The address of the low bytes of the descriptor then needs to be re-read from memory. However, since the bus is now locked, this cannot occur and the processor hangs waiting for the sequence to complete.

Implication: If all of the above conditions occur, the processor may hang.

Workaround: Ensure that the base address of the GDT or LDT is aligned. This will prevent the split locked cycle from occurring due to the misaligned descriptor. This is already recommended in the Intel Architecture Software Developer's Manual, Volume 3, Section 3.5.1 for performance reasons.

Status: For the steppings affected see the Summary Table of Changes at the beginning of this section.

79. Erroneous Debug Exception on POPF/IRET Instructions with a GP Fault

Problem: An erroneous debug exception can occur due to execution of a POPF or IRET instruction in virtual 8086 mode, if there is a data breakpoint set on the address pointed to by SS:ESP, and the POPF or IRET triggers a general protection fault. This occurs in virtual 8086 mode when the IOPL < 3, causing POPF and IRET to trap to the GP fault without accessing the stack. The data breakpoint set on the stack should not be triggered, but in fact it is incorrectly triggered as soon as the GP fault handler is entered.

Implication: This results in an invalid debug exception where the saved state (CS:EIP in the stack, or in the TSS in the case of a task-switch for interrupt 1) points to the first instruction of the GP Fault handler. This may confuse the debug monitor which expects to find a pointer to an instruction accessing the stack. Note this erratum only occurs during debugging and does not affect normal execution.

Workaround: The debug monitor could be revised to detect this erratum, and to only perform an IRET when this erratum is detected as the cause of entry into the debugger.

Status: For the steppings affected see the Summary Table of Changes.

80. CR2 and CR4 Content upon Return from SMM

Problem: Control registers CR2 and CR4 should maintain values across breakpoints or interrupts. CR2 contains the page fault linear address. CR4 is used in protected mode to control operations such as virtual-8086 support, enabling I/O breakpoints, page size extension and machine check exceptions. If CR2 or CR4 are modified in SMM, the original contents of CR2 and CR4 prior to entering SMM are not restored when exiting SMM.

Implication: If either CR2 or CR4 is modified during the execution of the SMM handler, the modified values will remain after a resume from the SMM handler. The new values in CR2 or CR4 may be unexpected.

Workaround: If the SMM handler needs to modify CR2 or CR4, the handler should store the values of CR2 and CR4 upon entering the SMM handler and restore the values prior to the RSM instruction.

Status: For the steppings affected, see the Summary Table of Changes at the beginning of this section.

81. Invalid Operand with Locked CMPXCHG8B Instruction

Problem: The CMPXCHG8B instruction compares an 8 byte value in EDX and EAX with an 8 byte value in memory (the destination operand). The only valid destination operands for this instruction are memory operands. If the destination operand is a register the processor should generate an invalid opcode exception, execution of the CMPXCHG8B instruction should be halted and the processor should execute the invalid opcode exception handler. This erratum occurs if the LOCK prefix is used with the CMPXCHG8B instruction with an (invalid) register destination operand. In this case, the processor may not start execution of the invalid opcode exception handler because the bus is locked. This results in a system hang.

Implication: If an (invalid) register destination operand is used with the CMPXCHG8B instruction and the LOCK prefix, the system may hang. No memory data is corrupted and the user can perform a system reset to return to normal operation. Note that the specific invalid code sequence necessary for this erratum to occur is not normally generated in the course of programming nor is such a sequence known by Intel to be generated by commercially available software.

This erratum only applies to Pentium processors, Pentium processors with MMX technology, Pentium OverDrive® processors and Pentium OverDrive processors with MMX technology. Pentium Pro processors, Pentium II processors and i486™ and earlier processors are not affected.

Workaround: There are two workarounds for this erratum for protected mode operating systems. Both workarounds generate a page fault when the invalid opcode exception occurs. In both cases, the page fault will be serviced before the invalid opcode exception and thus prevent the lock condition from occurring. The implementation details will differ depending on the operating system. Use one of the following:

1. The first part of this workaround sets the first 7 entries (0-6) of the Interrupt Descriptor Table (IDT) in a non-writeable page. When the invalid opcode exception (exception 6) occurs due to the locked CMPXCHG8B instruction with an invalid register destination (and only then), the processor will generate a page fault if it does not have write access to the page containing entry 6 of the IDT. The second part of this workaround modifies the page fault handler to recognize and correctly dispatch the invalid opcode exceptions that are now routed through the page fault handler.

   Part I, IDT Page Access:

   a. Mark the page containing the first seven entries (0-6) of the IDT as read only by setting bit 1 of the page table entry to zero. Also set CR0.WP (bit 16) to 1. Now when the invalid opcode exception occurs on the locked CMPXCHG8B instruction, the processor will check for write access due to the lock prefix and trigger a page fault since it does not have write access to the page containing entry 6 of the IDT. This page fault prevents the bus lock condition and gives the OS complete control to process the invalid operand exception as appropriate. Note that exception 6 is the invalid opcode exception, so with this scheme an OS has complete control of any program executing an invalid CMPXCHG8B instruction.

   b. Optional: If updates to entries 7-255 of the IDT occur during the course of normal operation, page faults should be avoided on writes to these IDT entries. These page faults can be avoided by aligning the IDT across a 4KB page boundary such that the first seven entries (0-6) of the IDT are on the first read only page and the remaining entries are on a read/writeable page.

   Part II, Page Fault Handler Modifications:

   a. Modify the page fault handler to calculate which exception caused the page fault using the fault address in CR2. If the error code on the stack indicates the exception occurred from ring 0 and if the address corresponds to the invalid opcode exception, then pop the error code off the stack and jump to the invalid opcode exception handler. Otherwise continue with the normal page fault handler.

   OR

2. This workaround has two parts. First, the Interrupt Descriptor Table (IDT) is aligned such that any invalid opcode exception will cause a page fault (due to the page not being present). Second, the page fault handler is modified to recognize and correctly dispatch the invalid opcode exception and certain other exceptions that are now routed through the page fault handler.

Part I, IDT Alignment:

a. Align the Interrupt Descriptor Table (IDT) such that it spans a 4KB page boundary by placing the first entry starting 56 bytes from the end of the first 4KB page. This places the first seven entries (0-6) on the first 4KB page, and the remaining entries on the second page.

b. The page containing the first seven entries of the IDT must not have a mapping in the OS page tables. This will cause any of exceptions 0-6 to generate a page not present fault. A page fault prevents the bus lock condition and gives the OS complete control to process these exceptions as appropriate. Note that exception 6 is the invalid opcode exception, so with this scheme an OS has complete control of any program executing an invalid CMPXCHG8B instruction.

Part II, Page Fault Handler Modifications:

a. Recognize accesses to the first page of the IDT by testing the fault address in CR2. Page not present faults on other addresses can be processed normally.

b. For page not present faults on the first page of the IDT, the OS must recognize and dispatch the exception which caused the page not present fault. Before proceeding, test the fault address in CR2 to determine if it is in the address range corresponding to exceptions 0-6.

c. Calculate which exception caused the page not present fault from the fault address in CR2.

d. Depending on the operating system, certain privilege level checks and adjustments to the interrupt stack may be required before jumping to the normal exception handler in Step e below. If you are an operating system vendor, please contact your local Intel representative for more information.

e. Jump to the normal handler for the appropriate exception.

Both workarounds should only be implemented on Intel processors that return Family=5 via the CPUID instruction.

82. Event Monitor Counting Discrepancy

Problem: The Pentium processor contains two registers which can count the occurrence of specific events used to measure and monitor various parameters that contribute to the performance of the processor. There is one condition where the counter does not operate as specified:

The “Stall on write to E or M state line” (event 011011) event counts the number of clocks the processor is stalled on a memory write to an E or M state line, while the write buffers are not empty, or EWBE# is negated. In order for event 011011 to accurately count stalled clocks cycles, it must ignore all other stall cases, such as TLB-miss. However, if data resides in the top 4 Kbyte of the physical address space, some stalls due to TLB-miss were also counted.

Implication: The event monitor counters report an inaccurate count for event 011011.

Workaround: Avoid mapping to the top 4 Kbytes of the address space, or physical page ‘0xFFFFF. *See also* Errata 75 and 76.

Status: For the steppings affected see the Summary Table of Changes.

83. FBSTP Instruction Incorrectly Sets Accessed and Dirty Bits of Page Table Entry

Problem: This erratum occurs only if a program does all of the following:

1. Paging is enabled.

2. The program uses 16-bit addressing inside a USE32 segment (requiring the 67H addressing override prefix) in order to wrap addresses at offsets above 64K back to the bottom of the segment.
3. The 10-byte BCD operand written to memory by the FBSTP instruction must actually straddle the 64K boundary. If all 10 bytes are either above or below 64K, the wrap works normally.

The result is that Accessed and Dirty bits of a Page Table entry are sometimes incorrectly set by the FBSTP instruction in case of a 16-bit address wraparound (Prefix 67H) within a 32-bit code segment. FBSTP does the wraparound, but the attributes of the next sequential Page Table entry are also set as if the page was accessed and written to.

Implication: An incorrectly set Accessed bit may cause a non-used page to be kept in memory. An incorrectly set Dirty bit may cause unnecessary disk write-back cycles.

Workaround: None.

1DP. Problem with External Snooping while Two Cycles Are Pending on the Bus

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

2DP. STPCLK# Assertion and the Stop Grant Bus Cycle

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

3DP. External Snooping with AHOLD Asserted May Cause Processor to Hang

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

4DP. Address Parity Check Not Supported in Dual Processing Mode

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

5DP. Inconsistent Cache State May Result from Interprocessor Pipelined READ into a WRITE

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

6DP. Processors Hang During Zero WS, Pipelined Bus Cycles

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

7DP. Bus Lock-up Problem in a Specific Dual Processing Mode Sequence

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

8DP. Incorrect Assertion of PHITM# without PHIT#

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

9DP. Double Issuance of Read Cycles

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

10DP. Line Invalidation May Occur On Read or Prefetch Cycles

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

11DP. EADS# or Floating ADS# May Cause Extra Invalidates

Problem: This erratum only occurs in a dual processing environment. Extra invalidates may occur into the L1 cache due to assertions of EADS#. If EADS# is asserted while the processor is driving the bus, an invalidate into the processor's L1 cache may occur.

Implication: The specification states that EADS# is ignored while the processor is driving the bus. Occurrence of this erratum means that unnecessary invalidations and writeback cycles may be performed resulting in sub-optimal performance.

Workaround: The system should not assert EADS# while the CPU owns the bus. Designs based on the 82430NX PCIset and other chip sets which do not assert EADS# while the CPU owns the bus are therefore not affected.

Status: For the steppings affected see the Summary Table of Changes.

12DP. HOLD and BOFF# During APIC Cycle May Cause Dual Processor Arbitration Problem

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

13DP. System Hang After Hold During Local APIC 2nd INTA Cycle

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

14DP. External Snoop Can Be Incorrectly Invalidated

Problem: An external snoop with INV pin = 0 (non invalidating snoop) can be incorrectly treated as an invalidating snoop under the following conditions:

1. The system must use Dual Processors, operating in Intel's DP mode.
2. An external snoop occurs via EADS#, with INV = 0.
3. The previous bus master must have driven CACHE# = 1, M/IO# = 1, D/C# = 1, (W/R# = 1 or LOCK# = 0), or the pins must float to this value by the time EADS# is asserted. (This corresponds to an immediately preceding bus cycle that was non cacheable, memory, data and write or locked read.)

Implication: A small fraction of non-invalidating external snoops will be invalidated incorrectly, which in turn will cause unnecessary write back cycles, resulting in a degradation of performance if the system uses non invalidating external snoops frequently. The degree of performance degradation will depend on the details of system hardware and software, and most importantly, on the amount of non invalidating external snoops.

Workaround: None identified at this time.

Status: For the steppings affected see the Summary Table of Changes.

15DP. STPCLK# Re-assertion Recognition Constraint With DP

Problem: The *Pentium® Processor Family Developer's Manual,* Section 14.4.2.1 describes how to assure that each assertion and de-assertion of STPCLK# is recognized. However, it is not possible to guarantee that all changes on STPCLK# will be recognized in a DP system. This is because snoops between the dual processors triggered by the PHITM# signal can delay the processor's entry into the Stop Grant state until well after the end of the Stop Grant cycle. It is specified that de-assertion of STPCLK# must be held for at least 5 clocks after the beginning of the processor's entry into the Stop Grant state to be guaranteed to be recognized by the processor. As it is not practical for the system to monitor the PHITM# signal, there is no practical way to guarantee that deassertion of STPCLK# will be recognized.

A DP system should not be designed to depend on every STPCLK# assertion being recognized and thus generating a Stop Grant bus cycle response, and/or on every STPCLK# de-assertion allowing execution of at least one instruction. If a system design (such as typical usage of STPCLK# for thermal control and/or power usage reduction) does not depend on either of these features, this erratum will have no effect. Aside from sometimes not displaying these two features, a Pentium Processor system will never hang or otherwise malfunction because of random assertion and de-assertion of STPCLK#.

Workaround: Do not design DP systems to depend on every STPCLK# assertion being recognized and thus generating a Stop Grant bus cycle response, or to depend on every STPCLK# de-assertion allowing execution of at least one instruction.

Status: For the steppings affected see the Summary Table of Changes.

16DP. Second Assertion of FLUSH# During Flush Acknowledge Cycle May Cause Hang

Problem: The *Pentium® Processor Family Developer's Manual,* Section 3.5.1.2 states that in a DP system the FLUSH# signal must not be asserted again until the FLUSH ACK cycle is generated. The erratum occurs when the dual processor hasn't been initialized (with the IPI), and a FLUSH# is asserted during a FLUSH ACK cycle (anytime from ADS# to 1 clock after BRDY# of FLUSH ACK cycle).

Implication: Asserting FLUSH# in a DP system with the dual processor un-initialized by an IPI during a FLUSH ACK cycle may cause a hang.

Workaround: Initialize the dual processor by sending an IPI, or do not assert FLUSH# during the FLUSH ACK cycle. The FLUSH# can safely be asserted two clocks after the completion (i.e., BRDY#) of the FLUSH ACK cycle.

Status: For the steppings affected see the Summary Table of Changes.

17DP. Asserting FLUSH# May Cause a Processor Deadlock in a DP System with a 2/7 Bus Fraction

Problem: In a Dual Processing (DP) system, when FLUSH# is asserted by the system, the bus ownership is first transferred from the primary to the secondary processor. With ownership of the bus, the secondary processor starts to flush its L1 cache. Upon completion of the flush, the secondary processor then returns the bus ownership to the primary processor. Next, the secondary processor will go into a waiting loop until the primary processor finishes flushing its cache and generates the flush acknowledge special cycle. Finally, upon sampling the flush acknowledge special cycle from the primary processor, the secondary processor exits the waiting loop and both processors return to the normal DP mode.

However, FLUSH# may not function correctly due to an internal cycle alignment issue in DP mode when operating with a 2/7 bus fraction. Due to this erratum, the secondary processor fails to wait for the primary processor to release the bus, and initiates a new bus request before the primary

processor finishes flushing its L1 cache. In this case, the primary processor does not have access to the bus and therefore cannot finish flushing its cache and cannot generate the flush acknowledgment special cycle; at the same time, the secondary processor has the bus but cannot resume operation until the primary processor issues the flush acknowledgement special cycle.

Implication: In a DP system which operates at 2/7 bus fraction, the usage of FLUSH# may cause primary and secondary processors to end up in a dead-lock situation.

Workaround: While using the processors in 2/7 bus fractions and DP mode, do not use FLUSH#.

1AP. Remote Read Message Shows Valid Status After a Checksum Error

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

2AP. Chance of Clearing an Unread Error in the Error Register

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

3AP. Writes to Error Register Clears Register

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

4AP. Three Interrupts of the Same Priority Causes Lost Local Interrupt

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

5AP. APIC Bus Synchronization Lost Due to Checksum Error on a Remote Read Message

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

6AP. HOLD During a READ from Local APIC Register May Cause Incorrect PCHK#

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

7AP. HOLD During an Outstanding Interprocessor Pipelined APIC Cycle Hangs Processor

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

8AP. PICCLK Reflection May Cause an APIC Checksum Error

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

9AP. Spurious Interrupt in APIC Through Local Mode

Problem: This erratum affects APIC in through Local (virtual wire) mode. The system can be a uniprocessing or dual processing system that is using a Pentium processor with the APIC in through-local (virtual wire) mode. This mode is supposed to cause the processor to respond to interrupts identically to a Level triggered 8259 interrupt controller, and is typically used to provide AT

compatibility mode for existing drivers. Currently it acts as an edge triggered mode interrupt controller, latching any interrupts that may be quickly asserted and then deasserted based on driver interception. The result is that some operating systems (i.e., Novell*) will report the spurious interrupt and it may impact the performance or operation of certain debug hooks for the operating system or network. Software disabling of the APIC by clearing bit 8 of the SVR (spurious vector interrupt register) will not prevent this from occurring.

Implication: Reports of the a spurious interrupt or lost interrupt message may continuously be output to the terminal connection and fill the screen of a monitoring host.

Workaround: Use one of the following:

1. Ignore/disable the spurious interrupt reports. This may impact other debug hooks normally associated with the network or operating system.
2. Rewrite drivers such that they disable interrupt processing during the driver execution, and then re-enable the interrupts at the end of the procedure.
3. Disable APIC instead of running it in through Local mode.

   By Hardware: By deasserting the APICEN pin prior to the falling edge of reset.

   By Software: This can be done on the B1, B3, B5, and C2-step components by using a reserved bit (bit 4) in the TR12 test register set to ‘1’. The use of a reserved bit is only for these steppings (B1, B3, B5, and C2) and the function of this bit may change in future steppings. When implementing this workaround ensure that the BIOS does a CPUID check looking for a specific stepping of the device. CPUIDs for the following components are B1 = 0521H, B3= 0522H, B5= 0524H and C2=0525H. If the TR12 register is used, the APIC is fully disabled. To re-enable APIC, bit 4 must be cleared to ‘0’ and then a warm reset of the part performed prior to APIC use of any kind.
4. For cB1, cC0 and E0 steppings, a software fix can be enabled by setting bit 14 of TR12 to ‘1’. By enabling this bit, an interrupt that is asserted and deasserted during the window that interrupts are disabled (after CLI and before STI) is ignored. If the interrupt is asserted during this window and deasserted after interrupts are enabled (after STI sets IF), the interrupt is latched and serviced. By setting bit 14 to ‘0’, the processor will behave as in earlier steppings.

Status: For the steppings affected see the Summary Table of Changes.

10AP. Potential for Lost Interrupts while Using APIC in Through Local Mode

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

11AP. Back to Back Assertions of HOLD or BOFF# May Cause Lost APIC Write Cycle

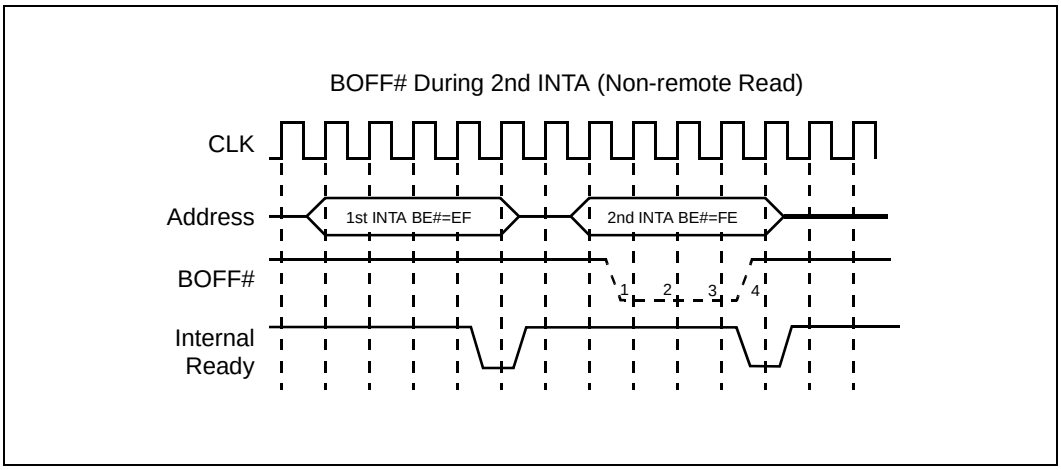
This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

12AP. System May Hang When BOFF# Is Asserted During the Second Internal APIC INTA Cycle

Problem: The processor will hang if BOFF# is asserted and deasserted during the second internal APIC INTA cycle. The erratum will occur if BOFF# is sampled during the interval (i.e., clocks 1, 2, or 3) shown in the figure below, although keeping BOFF# asserted through the last cycle (i.e., cycle 4) of the second INTA will prevent this erratum from occurring

If these conditions occur while a remote read message is being sent or received, the second INTA cycle may take up to 8 clocks to complete (counting from clock 1). BOFF# is not latched and must remain asserted until after the 2nd INTA completes (e.g., for 8 clocks).

Similarly, if HOLD is asserted anytime during the second INTA cycle, and during a remote read, the processor will hang.



Implication: If the system does not assert BOFF# when the processor bus is idle, then this problem will not occur. Internal APIC INTA cycles are run only when the bus is idle, thus asserting BOFF# during an external bus cycle (e.g., started by an ADS#) will avoid these circumstances. In systems that can assert BOFF# when the bus is idle, asserting BOFF# for a least 4 clocks will avoid the problem for non-remote read cases. Note that there may be other implementations that guarantee BOFF# is not asserted in the problematic window. For example, if BOFF# is only asserted with AHOLD active and AHOLD always precedes BOFF# by at least 4 clocks, the erratum is avoided.

If a remote read is occurring (e.g., between two processors) this will delay completion of the second INTA cycle by up to 8 clocks (i.e., bus clocks), and BOFF# asserted during this time may hang the system. Remote reads are typically performed during system debug and not in normal operation. Not performing remote reads will avert this case.

Asserting HOLD anytime during the second INTA cycle during a remote read cycle will cause the system to hang. Not asserting HOLD or not performing APIC remote read cycles will avert this case.

Workaround: Use one of the following to avoid the BOFF# case:

1. For non-remote read case, do not assert BOFF# when the bus is idle, or assert BOFF# for at least 4 clocks during idle bus cycles. For the remote read case, assert BOFF# for at least 8 clocks.
2. Use APIC in through local mode.

Status: For the steppings affected see the Summary Table of Changes.

13AP. APIC Pipeline Cycle During Cache Linefill Causes Restarted Cycle to Lose Its Attribute

Problem: When a read or write cycle to an APIC register is pipelined into a cache linefill and both cycles get backed off (by assertion of BOFF# for **one clock** only), the cache linefill that is restarted loses its attributes. When the cache linefill is restarted, although the processor does assert CACHE#, the processor loses track of the cacheability of the cycle and treats the burst linefill as a single cycle read.

Implication: The processor reads only the first quad-word (indicated by the first BRDY#), but ignores the following three transfers of the burst linefill. However, the internal APIC cycle is allowed to restart after the first BRDY#. When the APIC cycle completes and another bus cycle is started by the processor (indicated by an ADS#) before the last BRDY# from the burst linefill is returned, the leftover BRDY#s could incorrectly terminate the new cycle and the processor could lose synchronization with the bus, causing the processor to hang or get corrupted data.

It is unlikely this erratum will occur for systems using zero wait states (i.e., 2111 burst read) or one wait state lead off (i.e., 3111 burst read). A high-latency memory subsystem or I/O subsystem would increase the exposure of the new bus cycle to a leftover BRDY# (i.e., 3222 burst read).

Workaround: Use one of the following:

1. Always assert BOFF# for more than one clock.
2. Disable pipelining when using the APIC.
3. Avoid asserting BOFF# during pipelined linefill cycles when using the APIC.

Status: For the steppings affected see the Summary Table of Changes.

14AP. INIT and SMI# Via the APIC Three-Wire Bus May Be Lost

Problem: If the INIT and SMI# pins are kept asserted once they are recognized and then another INIT or SMI# is asserted to the processor via the APIC three-wire bus, the processor will not recognize this second assertion of INIT or SMI#.

INIT and SMI# are edge triggered interrupts and are only recognized on the rising edge (falling edge for SMI#). Since the processor only detects the edges on these pins, it is possible to hold the levels on these pins in the asserted state (logic 1 for INIT and logic 0 for SMI#). When another INIT or SMI# is required, the levels at these pins can be deasserted for several clocks and reasserted to generate the edge which triggers the interrupt. However, if the levels on these pins are kept asserted, and the APIC three-wire bus is also used to assert INIT and SMI# to the processor, the INIT and SMI# interrupts via the APIC three-wire bus are lost.

Implication: If the above conditions are met, INIT and SMI# interrupts via the APIC three-wire bus will be lost. Designs which do not use the APIC three-wire bus to assert INIT and SMI# will not be affected by this erratum.

Workaround: To avoid this erratum, use one of the following:

1. Assert INIT or SMI# to trigger the interrupt and then deassert the INIT or SMI# thereafter to avoid conflict with the APIC serial bus INIT or SMI# messages.
2. Do not send an INIT or SMI# message via the APIC three-wire bus.

Status: For the steppings affected see the Summary Table of Changes.

15AP. IERR# in FRC Lock-Step Mode During APIC Write

Problem: When an APIC write is pipelined into a memory write, IERR# is incorrectly asserted one clock after the BRDY# of the memory write for a duration of one clock. (Note that APIC write cycles are not driven on the external bus). This problem is a subset of the problem described in Erratum 29.

Implication: This will cause an inadvertent IERR# to occur for one clock.

Workaround: Disable pipelining in FRC lock-step mode.

Status: For the steppings affected see the Summary Table of Changes.

16AP. Inadvertent BRDY# During External INTA Cycle With BOFF#

Problem: The *Pentium® Processor Family Developer's Manual* states that BRDY# is ignored during assertion of ADS#. There are two cases when using the APIC in through local mode where an inadvertent BRDY# asserted during the ADS# of an external INTA cycle, in combination with a subsequent BOFF#, can cause the processor to hang.

1. If during the first INTA cycle an inadvertent BRDY# is asserted with ADS#, followed by the real BRDY# for that cycle, and then the 2nd INTA cycle is backed off, the processor loses synchronization. Instead of restarting the 2nd INTA cycle externally, the processor ends the cycle internally without reading a valid interrupt number (0-255) which hangs the interrupt handler. The window that BOFF# can cause this erratum is after (the valid BRDY#) completion of the 1st INTA cycle and before completion of the 2nd INTA cycle.
2. If the 1st INTA cycle completes correctly (with only one valid BRDY#), and an inadvertent BRDY# is asserted during the ADS# of the 2nd INTA cycle, and then the 2nd INTA cycle is backed off before its completion, the processor again loses synchronization and hangs.

Implication: The interrupt will not be serviced and the system hangs waiting for the processor to complete its 2nd INTA cycle.

Workaround: Use one of the following:

1. Do not assert BRDY# during ADS#.
2. Do not assert BOFF# during an external INTA cycle.

Status: For the steppings affected see the Summary Table of Changes.

17AP. APIC Read Cycle Doesn't Complete Upon Assertion of BOFF# and HOLD

Problem: During an APIC cycle (read or second INTA), if BOFF# is asserted for one clock, and HOLD is asserted in the following cycle (at the rising edge of BOFF#) for one clock, the APIC cycle may either not complete or not complete correctly. Either the processor is waiting for the APIC cycle to complete before issuing any new cycles and the processor hangs (APIC read), or the cycle does complete but with the incorrect Interrupt Vector being recognized (APIC second INTA). Note that this erratum does not occur when BOFF# and HOLD are asserted simultaneously for one clock.

Implication: Systems typically use either BOFF# or HOLD (but not both) to gain control of the bus. If a system were to assert this sequence of BOFF# and HOLD for one clock each, the system may be susceptible to a hang.

Workaround: Do not assert BOFF# for one clock immediately followed by HOLD for one clock. If HOLD must follow BOFF# by one clock, assert one of the signals (BOFF# or HOLD) for more than one clock.

Status: For the steppings affected see the Summary Table of Changes.

18AP. PICCLK Must Toggle For at Least Twenty Cycles Before RESET

Problem: In order for the internal circuitry of the local APIC to initialize properly, PICCLK must toggle at least twenty times (1.2 µS – 10 µS depending on the PICCLK frequency) before the falling edge of RESET.

Implication: An improper initialization of the internal APIC circuits may cause, for example, the APICEN/ PICD1 pin to be erroneously driven low; thus, the on-chip APIC would not be enabled. In such a scenario, the second-processor in DP systems and all messages sent on the serial APIC bus would not be recognized.

Workaround: Ensure that PICCLK toggles for at least twenty cycles before the falling edge of RESET.

Status: For the steppings affected see the Summary Table of Changes.

19AP. APIC ID Can Not Be Changed

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

1TCP. CPU May Not Reset Correctly Due to Floating FRCMC# Pin

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

2TCP. BRDY# Does Not Have Buffer Selection Capability

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

Specification Clarifications

1. Pentium® Processor's Response to Startup and Init IPIs

The Pentium processor when used as a dual processor upgrade component, will require a STARTUP IPI to wake up this part after the following two situations:

- After any assertion of RESET.
  Or
- After any assertion of INIT.

(The assertion of INIT could come from toggling the INIT pin or though an APIC IPI.)

In either case, the dual processor upgrade component will not jump to the RESET Vector, it will instead go into a halt state. If an INIT IPI is then sent to the halted upgrade component, it will be latched and kept pending until a STARTUP IPI is received. From the time the STARTUP IPI is received the CPU will respond to further INIT IPIs but will ignore any STARTUP IPIs. It will not respond to future STARTUP IPIs until a RESET assertion or an INIT assertion (INIT Pin or INIT IPI) happens again.

The Pentium processor when used as a primary processor, will never respond to a STARTUP IPI at any time. It will ignore the STARTUP IPI with no effects.

To shutdown the processors the operating system should only use the INIT IPI, STARTUP IPIs should never be used once the processors are running.

The following pseudo-code shows the generic algorithm for waking up Pentium processors, including 82489DX based systems, dual processor systems and multi-processor systems.

```
BSP sends AP an INIT IPI
BSP DELAYs (10mSec)
If (APIC VERSION is not an 82489DX)
{
    BSP sends AP a STARTUP IPI
    BSP DELAYs (200uSec)
    BSP sends AP a STARTUP IPI
    BSP DELAYs (200uSec)
}
BSP verifies synchronization with executing AP
```

For additional information please refer to the *Intel Multiprocessor Specification*, Version 1.4 (Order Number 242016).

2. APIC Timer Use Clarification

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

3. PICCLK Reflection May Cause APIC Checksum Errors and Dropped IPIs

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

4. Boundary Scan RUNBIST Register Requires Initialization Prior to Use

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

5. Only One SMI# Can Be Latched During SMM

Section 20.1.4.2 of Volume 3 of the *Pentium® Processor Family Developer's Manual* correctly states that only one SMI# can be latched by the CPU while it is in SMM (end of second paragraph). However, Section 5.1.50 of Volume 1 of the manual in the SMI# pin definition incorrectly implies by the use of the plural that more than one SMI# request may be held pending during SMM. Thus the following changes will be implemented in the next revision of the Manual:

Section 20.1.4.2 of Volume 3, next to last sentence in the second paragraph, will have the underlined phrase added: "The first SMI# interrupt request that occurs while the processor is in SMM (i.e., after SMIACT# has been asserted) is latched, and serviced when the processor exits SMM with the RSM instruction."

Section 5.1.50 of Volume 1: The second paragraph of the Signal Description, that refers to SMI# requests held pending during SMM, will be replaced with the entire second paragraph of Section 20.1.4.2 of Volume 3.

6. APIC 8-Bit Access

The following should be added to the *Pentium® Processor Family Developer's Manual*, Volume 3, Section 19.3.1.4. The APIC supports 32 bit sized, 32 bit aligned, read and write cycles to its registers. Therefore, all APIC registers should be accessed using 32-bit loads and stores. If a 32-bit APIC register is accessed with an 8 or 16 bit write cycle the result may be unpredictable. This implies that to modify a field, the entire 32-bit register should be read, the field modified, and the entire 32 bits written back.

7. LOCK Prefix Excludes APIC Memory Space

In the *Pentium® Processor Family Developer's Manual*, Volume 3, page 25-216, the LOCK prefix is described. A line should be added at the end of the description as follows: The LOCK prefix has no effect on instructions that address the APIC memory space. Therefore, LOCK# is not asserted.

8. SMI# Activation May Cause a Nested NMI Handling

In the *Pentium® Processor Family Developer's Manual*, Volume 3, Section 20.1.4.4, the following note should be added just before the last paragraph.

During NMI interrupt handling NMI interrupts are disabled. NMI interrupts are serviced and completed with IRET one at a time. When the processor enters SMM from the NMI interrupt handler, the processor saves the SMRAM State Save Map (e.g., contents of status registers) but does not save the attribute to keep NMI interrupts disabled. Potentially a NMI could be latched (while in SMM or upon exit) and serviced upon exit of SMM even though the previous NMI handler has still not completed. One or more NMIs could be nested in the first NMI handler. The interrupt handler should take this into consideration.

9. Code Breakpoints Set on Meaningless Prefixes Not Guaranteed to be Recognized

The following should be added to the *Pentium® Processor Family Developer's Manual*, Volume 3, Section 17.3.1.1 (Instruction-Breakpoint Fault).

Code breakpoints set on meaningless instruction prefixes (a prefix which has no logical meaning for that instruction, e.g., a segment override prefix on an instruction that does not access memory) are not guaranteed to be recognized.

Code breakpoints should be set on the instruction opcode, not on a meaningless prefix.

In the *Pentium® Processor Family Developer's Manual*, Volume 3, Sections 3-4 (Instruction Format) and 25.2 (Instruction Format), after "For each instruction one prefix may be used from each group. The effect of redundant prefixes (more than on prefix from a group) is undefined and may vary from processor to processor." The following should be added:

Some prefixes when attached to specific instructions have no logical meaning (e.g., a segment override prefix on an instruction that does not access memory). The effect of attaching meaningless prefixes to instructions is undefined and may vary from processor to processor.

10. Resume Flag Should Be Set by Software

The lead-in sentences and first bullet of Section 14.3.3 in the *Pentium® Processor Family Developer's Manual*, Volume 3 should be replaced with the following:

The RF (Resume Flag) in the EFLAGS register should be used during debugging to avoid servicing an instruction breakpoint fault multiple times. RF works as follows:

- The debug handler (interrupt #1) should set the RF bit in the EFLAGS image on the stack whenever it is servicing an instruction breakpoint fault (rather than a data breakpoint trap), and the breakpoint is being left in place. If this is not done, the CPU will return to execute the instruction, fault on the breakpoint again to interrupt #1, and so on.

The following should be added as fifth and sixth bullets:

- If a fault type breakpoint coincides with another fault (the instruction accesses a not present page, violates a general protection rule, etc.) one spurious repetition of the breakpoint will occur after the second fault is handled, even though the debug handler sets RF. As an optional debugging convenience, to avoid this occasional confusion, all interrupt handlers that could interact during debugging in this way can be modified by having them also set the RF bit in the EFLAGS image on their stack.
- The CPU, in branching to fault handlers under some circumstances, will set the RF bit in the EFLAGS image on the stack by hardware action. Exactly when the CPU does this is implementation specific and should not be relied upon by software. No problem is caused by setting this bit again if it is already set.

11. Data Breakpoints on INS Delayed One Iteration

The *Pentium® Processor Family Developer's Manual*, Volume 3, last paragraph and sentence of Section 17.3.1.2 states, "Repeated INS and OUTS instructions generate a memory breakpoint debug exception trap after the iteration in which the memory address breakpoint location is accessed."

The sentence should read, "Repeated OUTS instructions generate a memory breakpoint debug exception trap after the iteration in which the memory address breakpoint location is accessed. Repeated INS instructions generate the memory breakpoint debug exception trap one iteration later."

12. When L1 Cache Disabled, Inquire Cycles are Blocked

The last line in Table 18-2 in the *Pentium® Processor Family Developer's Manual,* Volume 3 presently reads "Invalidation is inhibited". This is part of the description of L1 cache behavior when it is "disabled" by setting CR0 bits CD = NW = 1. This line will be clarified to read "Inquire cycles (triggered by EADS# active) and resulting invalidation and any APCHK# assertions are inhibited."

13. Serializing Operation Required When One CPU Modifies Another CPU's Code

A new subsection, 19.2.1, will be added to the *Pentium® Processor Family Developer's Manual,* Volume 3, titled *Processor Modifying Another Processor's Code,* and it will be referenced in the current subsection 18.2.3 on self modifying code.

A particular problem in memory access ordering occurs in a multiprocessing system if one processor (CPU1) modifies the code of another (CPU2). This obviously requires a semaphore check by CPU2 before executing in the area being modified, to assure that CPU1 is finished with the changes before CPU2 begins executing the changed code. In addition, it is necessary for CPU2 to execute a serializing operation after the semaphore allows access but before the modified code is executed. This is needed because the external snoops into CPU2 caused by the code modification by CPU1 will invalidate any matching lines in CPU2's code cache, but not in its prefetch buffers or execution pipeline. Note that this is different from the situation described in Section 18.2.3 on self-modifying code. When the CPU modifies its own code, the prefetch buffers and pipeline as well as the code cache are checked and invalidated if necessary.

14. For Correct Translations, the TLB Should be Flushed After the PSE Bit in CR4 Is Set

Memory mapping tables may be changed by setting the page size extension bit in CR4 (bit 4). However if the TLB is not flushed after the CR4.PSE bit is set, it may provide an erroneous 4 Kbyte page translation rather than a new 4 Mbyte page translation, or the other way around. Therefore for correct translations, the TLB should be flushed by writing to CR3 after the CR4.PSE bit is set.

This will be added to the *Pentium® Processor Family Developer's Manual,* Volume 3, Sections 10.1.3 and 11.3.5.

15. When APIC Enabled, its 4K Block Should Not be Used in Regular Memory

When the local APIC is enabled, it uses a 4 Kbyte memory mapped address block starting at 0FEE00000H for its control and status registers. Obviously one can't use the 4K block at 0FEE00000H in regular memory for data, because reads and writes would always go to the APIC registers instead. Not obviously, code placed in this location in memory usually is fetched correctly, because the bus unit normally distinguishes code fetches from APIC reads and puts the code fetches on the external bus. Nonetheless, this 4K block should not be used for code either, because in a case when the code fetch is backed off, the bus unit directs the recovered code fetch cycle to the APIC, resulting in interrupt 6, or unpredictable execution.

The following NOTE will be added as the last text in Section 19.3.1.4 in the *Pentium® Processor Family Developer's Manual,* Volume 3: "When the APIC is enabled, the 4K page in regular memory that overlays the 4K block assigned to the APIC should not be used, for either code or data."

16. Extra Code Break Can Occur on I/O or HLT Instruction if SMI Coincides

If a code breakpoint is set on an I/O instruction, as usual the breakpoint will be taken before the I/O instruction is executed. If the I/O instruction is also used as part of an I/O restart protocol, I/O restart is enabled, and executing the instruction triggers SMI, RSM from the SMI handler will return to the start of the I/O instruction, and the code breakpoint will be taken again before the I/O instruction is executed a second time.

Similarly, if a code breakpoint is set on an HLT instruction, the breakpoint will be taken before the processor enters the HLT state. If SMI occurs during this state, and the SMI handler chooses to RSM to the HLT instruction (the usual choice, for SMI to be transparent), the code breakpoint will be taken again before the HLT state is re-entered. In this case, other problems can occur, because an internal HLT flag remains set incorrectly. These problems are documented in Erratum # 55, case 2.

This information will be added to the end of Section 17.3.1.1, on "Instruction-Breakpoint Faults," in the *Pentium® Processor Family Developer's Manual,* Volume 3.

17. LRU May Be Updated for Non-cacheable Cycles

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

18. FYL2XP1 Does Not Generate Exceptions for X Out of Range

The FYL2XP1 instruction is intended to be used only for taking the log of numbers very close to one, to provide improved accuracy. For X values outside of the FYL2XP1 instruction's valid range, the FYL2X instruction should be used instead. The present documentation of what happens when X is outside of the FYL2XP1 instruction's valid range is inconsistent. For FYL2XP1, out of range behavior will be replaced by "If the ST operand is outside of its acceptable range, the result is undefined, and software should not rely on an exception being generated. Under some circumstances exceptions may be generated when ST is out of range, but this behavior is implementation specific and not guaranteed." The information on pages 7-15 and 25-161 of the *Pentium® Processor Family Developer's Manual*, Volume *3* will be clarified.

19. Enabling NMI Inside SMM

Page 20-11 of the *Pentium® Processor Family Developer's Manual* Volume 3 states "Although NMI requests are blocked when the CPU enters SMM, they may be enabled through software by invoking a dummy interrupt and vectoring to an Interrupt Service Routine."This will be changed to: "Although NMI requests are blocked when the CPU enters SMM, they may be enabled by first enabling interrupts through INTR by setting the IF flag, and then by triggering INTR. Also, for the Pentium processor, exceptions that invoke a trap or fault handler will enable NMI inside of SMM. This behavior of exceptions enabling NMI within SMM is not part of the Intel Architecture, and is implementation specific".

20. BF[1:0] Must Not Change Values While RESET is Active

Table 4-3 and page 2-49 of the *Pentium® Processor Family Developer's Manual* states that BF[1:0] must not change values while RESET is active. Page 5-18 of the *Pentium® Processor Family Developer's Manual* also states that BF[1:0] must meet a 1 mS setup time to the falling edge of RESET. Since RESET has to be active for at least 1ms, the setup time spec of 1mS is a subset of the specification in Table 4-3 and will be removed. t43a (BF[1:0] 1 mS setup time to the falling edge of RESET) will also be removed from Tables 7-8, 7-10, 7-12 and page 2-49.

The following will also be added to the "When Sampled/Driven" section on page 5-18:

Additionally, BF[1:0] must not change values while RESET is active.

The following will be added to the end of the first paragraph on page 5-17 and to note 22 on page 7-27:

In order to override the internal defaults and guarantee that the BF[1:0] inputs remain stable while RESET is active, these pins should be strapped directly to or through a pullup/pulldown resistor to V_{CC3} or ground. Driving these pins with active logic is not recommended unless stability during RESET can be guaranteed.

On pages 3-1 and 5-80, the sentence starting with "During power up, RESET must be asserted while V_{CC}..." will be modified as follows:

During power up, RESET should be asserted prior to or ramped simultaneously with the core voltage supply to the processor.

21. Active A20M# During SMM

Section 14.3.3. of the 1997 *Pentium® Processor Family Developer's Manual* describes two considerations when using the A20M# input and SMM when SMRAM is relocated above 1 Megabyte. The system designer must ensure that A20M# is de-asserted on entry into SMM. A20M# must be driven inactive before the first cycle of the SMM state save, and must be returned to its original level after the last cycle of the SMM state restore. This can be done by blocking the assertion of A20M# whenever SMIACT# is active.

The following will be added to the end of Section 14.3.3:

In addition to blocking the assertion of A20M# whenever SMIACT# is active, the system must also guarantee that A20M# is de-asserted at least one I/O clock prior to the assertion of SMIACT#. The processor may start the SMM state save as soon as SMIACT# is asserted. Processors faster than 200 MHz may not have enough time to recognize the de-assertion of A20M# before starting the SMM state save. As a result, this may cause the processor to start the first few cycles of the SMM state save with A20M# asserted. To avoid this, the system designer can use either of the following:

1. When relocating the SMRAM above 1 Megabyte, ensure that the SMRAM does not coincide with any odd megabyte addresses. (Note that systems which use A20M# and SMM but do not relocate SMRAM above 1 Megabyte are not affected.)
2. Use external logic to prevent the assertion of SMI to the processor until A20M# is de-asserted (and guarantee that A20M# remains de-asserted while in SMM).

Note that the A20M# input must also meet setup and hold times in order to be recognized in a specific clock.

22. POP[ESP] with 16-bit Stack Size

In the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual* and the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, the section regarding "POP–Pop a Value from the Stack", the following note in incomplete:

"If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register."

It should read as follows:

"If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register. In the case of a 16-bit stack where ESP wraps to 0h as a result of the POP instruction, the resulting location of the memory write is processor family specific."

In Section 15.12.1. of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide* and Section 17.23.1. of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, add a new section:

A POP-to-memory instruction, which uses the stack pointer (ESP) as a base register.

For a POP-to-memory instruction that meets the following conditions:

1. The stack segment size is 16-bit,
2. Any 32-bit addressing form with the SIB byte specifying ESP as the base register, and
3. The initial stack pointer is FFFCh (32-bit operand) or FFFEh (16-bit operand) and will wrap around to 0h as a result of the POP operation,

The result of the memory write is processor family specific. For example, in Pentium II and Pentium Pro processors, the result of the memory write is to SS:0h plus any scaled index and displacement. In Pentium and Intel486 processors, the result of the memory write may be either a stack fault (real mode or protected mode with a stack segment size of 64 Kbytes), or write to SS:10000h plus any scaled index and displacement (protected mode and stack segment size exceeds 64 Kbytes).

23. Pin #11 and Pin #190 (TCP Package) Connection

This item does not apply to Pentium processors for embedded applications. For the full text of this item, refer to the *Pentium® Processor Specification Update*, order number 242480.

24. Line Fill Order Optimization Revision

In Section 3.5.3 of the *Intel Architecture Optimization Manual*, the following passage is incomplete:

"For Pentium processors with MMX technology, Pentium Pro and Pentium II processors, data is available for use in the order that it arrives from memory. If an array of data is being read serially, it is preferable to access it in sequential order so that each data item will be used as it arrives from memory. On Pentium processors, the first 8-byte section is available immediately, but the rest of the cache line is not available until the entire line is read from memory."

Instead, it should read as follows:

"For Pentium processors with MMX technology, Pentium Pro and Pentium II processors, data is available for use in the order that it arrives from memory. On these processors, if an array of data is being read serially, it is preferable to access it in sequential order so that each data item will be used as it arrives from memory. On Pentium processors with MMX technology, the first 8-byte section is available immediately after it arrives from memory (after 5 I/O cycles), but the rest of the cache line is not available until the entire line is read from memory. If an array of data is being read serially, it is often preferable to pre-fetch portions of the array mapping to other cache lines. The first 8-byte section of the other cache line will be available for use immediately after it arrives from memory. This technique can be used to effectively hide the latency associated with accessing the last three 8-byte sections of the cache line.access it in sequential order so that each data item will be

used as it arrives from memory. If a request of reading the next cache line is made after the first quad-word of the first cache line is available, memory will provide the first 8-byte section of the second cache line, while the processor loads rest of the three quad-words of the first cache line from the in-line buffer. It take three I/O cycles to load the first section of the second cache line instead of five I/O cycles. Notice the order for each data item for these two cache line to arrive from memory will still be in a sequential order, that is, the data items for the second cache line will not be available until the first cache line is completely loaded. This technique is frequently used in modern compiler technology.

25. Test Parity Check Mechanism Clarification

In the 1997 *Pentium® Processor Family Developer's Manual*, Section 16.2.1.4 on Test Parity Check, the sentence, "For the microcode, bad parity may be forced on a read by setting the PRR.MC bit to 1." is incorrect. The correct wording should be as follows:

"For the microcode, bad parity may be forced on a read by a transition of the PRR.MC bit from 0 to 1. No bad parity will be forced by setting the PRR.MC bit to 1 if the bit was already set as 1."

Documentation Changes

The Documentation Changes listed in this section apply to the documents listed in the Preface of this Specification Update. Documentation Changes that are incorporated into a future version of the appropriate Pentium processor documentation are removed from the specification update upon publication of the amended document.

1. JMP Cannot Do a Nested Task Switch, Volume 3, Page 13-12

In the *Pentium® Processor Family Developer's Manual*, Volume 3, Section 13.6, the sentence "When an interrupt, exception, jump, or call causes a task switch..." incorrectly includes the **jump** in the list of actions that can cause a **nested** task switch. The word "jump" will be removed from the sentence. The Table 13-2 correctly shows the effects of task switches via jumps vs. Task switches via CALL's or interrupts, on the NT flag and the Link field of the TSS.

2. Interrupt Sampling Window, Volume 3, Page 23-39

In the *Pentium® Processor Family Developer's Manual*, Volume 3, Section 23.3.7 the first sentence of the second paragraph "The Pentium processor... asserts the FERR# pin." Should be replaced with the following:

The Pentium processor and the Intel486 processor implement the "No-Wait" Floating-Point instructions (See Section 6.3.7) in the DOS-Compatibility mode (CR0.NE = 0) in the following manner:

In the event of a pending unmasked numeric exception, the "No-Wait" class of instructions asserts the FERR# pin.

3. FSETPM Is Like NOP, Not Like FNOP

In the *Pentium® Processor Family Developer's Manual*, Volume 3, page 23-37 in the Section 23.3.4 on Instructions, the 80287 instruction FSETPM is described as being equivalent to an FNOP when executed in the Intel387 math coprocessor and the Intel486 and Pentium processors. In fact, FSETPM is treated as a NOP in these processors, as is correctly explained (along with the difference between FNOP and NOP) on the next page in Section 23.3.6. "FNOP" will be changed to "NOP" in the FSETPM description.

4. Errors in Three Tables of Special Descriptor Types

In the *Pentium® Processor Family Developer's Manual*, Volume 3 on page 25-199 and on page 25-222, in the descriptions of the **LAR** and **LSL** instructions respectively, tables are given of the special segment and gate descriptor types and names, with indication of which ones are valid with the given instruction. The same two pairs of descriptor types are interchanged in these two tables. Descriptor type 6 is the 16-bit interrupt gate, not trap gate, and type 7 is the 16-bit trap gate, not interrupt gate. Similarly, descriptor type 0Eh is the 32-bit interrupt gate, and 0Fh is the 32-bit trap gate. Table 12-1 gives a completely correct listing of the special descriptor types, but in the same chapter, Table 12-3 (page 12-22) incorrectly indicates that the 16-bit gates are not valid for the **LSL** instruction (this table *does* have the correct types for the interrupt and trap gates that it shows).

5. Invalid Arithmetic Operations and Masked Responses to Them Relative to FIST/FISTP Instruction

The *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, Table 7-20 and the *Intel Architecture Software Developer's Manual, Volume 1*, Table 7-20 show "Invalid Arithmetic Operations and the Masked Responses to Them." The table entry corresponding to the FIST/FISTP condition is missing, and is shown below:

Condition	Masked Response
FIST/FISTP instruction when input operand <> MAXINT for destination operand size.	Return MAXNEG to destination operand.

6. Incorrect Sequence of Registers Stored in PUSHA/PUSHAD

In the *Intel Architecture Software Developer's Manual, Volume 2*, the section on PUSHA/ PUSHAD–Push All General Purpose Registers, the sentence, "The registers are stored on the stack in the following order: EAX, ECX, EDX, EBX, EBP, ESP (original value), EBP, ESI and EDI (if the current operand-size attribute is 32)" incorrectly states the sequence of the registers stored on the stack. The sentence should state, "The registers are stored on the stack in the following order: EAX, ECX, EDX, EBX, ESP (original value), EBP, ESI and EDI (if the current operand-size attribute is 32)".

7. One-Byte Opcode Map Correction

In the *Intel Architecture Software Developer's Manual, Volume 2*, there are two corrections that need to be made to the Opcode Map:

1. In Appendix A, Opcode Map, Table A-1, Row 9, Column 8, only CBW is indicated. It should indicate both CBW and CWDE.
2. In Appendix A, Opcode Map, Table A-1, Row 9, Column A, the operand is defined as "aP". It should be defined as "Ap". Operand "Ap" selects a pointer, 32-/48-bits in size without specifying a MOD R/M byte.

Appendix A Pentium® Processor Related Technical Collateral

Unless otherwise noted, the following documentation may be obtained by visiting Intel's website at http:\\www.intel.com or by contacting Intel's Literature Center at 1-800-879-4683 in the U.S. and Canada. In other geographies, please contact your local sales office.

Document Title	Order Number
Embedded Pentium® Processor Family Developer's Manual (1998)	273204
Embedded Pentium® Processor Datasheet	273202
Embedded Pentium® Processor with Voltage Reduction Technology Datasheet	273203
Embedded Pentium® Processor with MMX™ Technology Datasheet	273214
Low-Power Embedded Pentium® Processor with MMX™ Technology Datasheet	273184
Embedded Pentium® Processor with MMX™ Technology Flexible Motherboard Design Guidelines	273206
Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture	243190
Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference	243191
Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide	243192
Pentium® Processor Specification Update	242480
MultiProcessor Specification	242016
Pentium® Processor Family Product Brief	241561
Pentium® Processor Performance Brief	241557
Pentium® Processor Technical Overview	241610
AP-579: *Pentium® Processor Flexible Motherboard Design Guidelines*	243187
AP-479: *Pentium® Processor Clock Design*	241574
AP-480: *Pentium® Processor Thermal Design Guidelines*	241575
AP-485: *Intel Processor Identification with the CPUID Instruction*	241618
AP-500: *Optimizations for Intel's 32-Bit Processors*	241799
AP-577: *An Introduction to PPGA Packaging*	243103
AP-522: *Implementation Guidelines for 3.3V Pentium® Processors with VRE Specifications*	242687
AP-578: *Software and Hardware Considerations in Handling FPU Exceptions*	242415