Pentium® Processor Specification Update

Release Date: October, 1996

Order Number: 242480-022

The Pentium processor may contain design defects or errors known as errata. Characterized errata that may cause the Pentium processor's behavior to deviate from published specifications are documented in this specification update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
February, 1995	-001	This document consolidates information previously contained in various versions of stepping information, notably the B-, C- and D-stepping of Pentium® processor at iCOMP® index (510\60, 567\66) and the B- and C-stepping of Pentium processor at iCOMP index (610\75, 735\90, 815\100).
March, 1995	-002	Added Errata 22-25 and Spec Clarification 15 to Part I. Added Spec Change 12, Errata 24-27, 8DP-12DP, 11AP, Spec Clarification 3, and Doc Change 6 to Part II.
April, 1995	-003	Part II: Added new markings for 120 MHz; added Spec Changes 13, 14, 15, 16, 17, 2TCP, 3TCP; revised Spec Change 7, 11, 1TCP; added Errata 28 and 29; revised Errata 2TCP.
May, 1995	-004	Part I: Added Errata 26 and 27; added Spec Clarification 17 and 18; added Doc Change 3. Part II: Revised Spec Change 12 and 17; added Spec Change 18; revised Errata 25 and 4DP and added Errata 30-33; added Spec Clarifications 9-13; added Doc Changes 7 and 8. Part III: New Section — Pentium® OverDrive® Processor.
June, 1995	-005	Part I: Added Spec Change 5; added Erratum 28; added Spec Clarification 19. Part II: Added Pentium Processor with Voltage Reduction Technology; added 133 MHz; added spec changes 19-23; revised Spec Changes 12-17; added S-Specs 1 and 2; added Errata 34, 35 and 13DP; revised Errata 4DP and 9AP; added Spec Clarification 14-17; revised Spec Clarification 8. Part III: Added Spec Change 3; added Erratum 22; added Spec Clarification 8-10
July, 1995	-006	Part I: Added Erratum 29. Part II: Added Spec Change 24; revised Spec Change 7 and 17; added Erratum 36; added Spec Clarification 18. Part III: Added Spec Change 4; added Errata 23 and 24.
August, 1995	-007	Part I: Added Errata 30-32; added Spec Clarification 20 and 21. Part II: Revised Spec Change 21; added S-Spec 3; added Errata 37-41; revised Erratum 9; added Spec Clarification 19-26. Part III: Added Errata 25-29; added Spec Clarification 11, 12.

September, 1995	-008	Removed Spec Changes, Spec Clarifications and Doc Changes which were incorporated into the 1995 *Pentium® Processor Family Developer's Manual* (Order Number 241563-004). Part I: Added Errata 33 and 34; added Doc Changes 1 and 2. Part II: Added Errata 42, 43 and 12AP; added Spec Clarification 13; added Doc Changes 1-6. Part III: Added Erratum 30; added Doc Changes 1 and 2.
October, 1995	-009	Part I: Added Spec Changes 1 and 2; added Errata 35-37; added Spec Clarification 4-7; added Doc Changes 3 and 4. Part II: Added Spec Changes 12 and 13; added Errata 44-48; revised Erratum 11; added Spec Clarification 14-17; revised Spec Clarification 1; added Doc Changes 7-9; revised Doc Change 3. Part III: Added 83 MHz Pentium OverDrive processor; added Spec Changes 4-6; added Errata 31-33; revised erratum 7; added Spec Clarification 6 and 7; added Doc Changes 3 and 4.
November, 1995	-010	Part I: Added Errata 38-40; added Spec Clarifications 8-10; added Doc Change 5. Part II: Added 120 MHz Pentium processor with Voltage Reduction Technology; added Spec Changes 14-15; added S-Spec 4; added Errata 49-53, 13AP and 14AP; revised Erratum 10; added Spec Clarifications 18-21; added Doc Change 10. Part III: Added Errata 34-36; revised Erratum 10; added Spec Clarification 8-9; added Doc Change 5.
December, 1995	-011	Part I: Added Errata 41-44; revised errata 35 and 37; added Spec Clarifications 11-12; revised Spec Clarification 5 and 10; added Doc Change 6. Part II: Added Spec Change 16; added Errata 54-58; revised Errata 44 and 46; added Spec Clarifications 22 and 23; revised Spec Clarifications 5, 15 and 20; added Doc Change 11. Part III: Added Errata 37-40; revised Erratum 31; added Spec Clarification 10-12; revised Spec Clarification 6; added Doc Change 6.
January, 1996	-012	Part I: Added Errata 45-46; revised Errata 37 and 43; added Spec Clarification 13; added Doc Changes 7 and 8. Part II: Added Pentium Processor 150-MHz and 166-MHz, added Spec Changes 17-20; added Errata 59-64, 14DP, 15AP and 16AP; revised Errata 46, 56, 9AP and 11AP; added Spec Clarifications 24 and 25; revised Spec Clarification 9; added Doc Change 12-13. Part III: Added Spec Change 7; added Errata 41-45; revised Erratum 39; added Spec Clarification 13; added Doc Change 7 and 8.

February, 1996	-013	Part II: Added Erratum 17AP.
March, 1996	-014	Part I: Added Erratum 47. Part II: Added S-Spec 5 and 6; added Errata 65 and 66. Part III: Added Erratum 46.
April, 1996	-016*	Part I: Revised Erratum 35; added Spec Clarification 14; added Doc Change 9. Part II: Added S-Spec 7; revised Erratum 44; added Spec Clarifications 26-29; added Doc Change 14. Part III: Added 120/133-, 125-, 150- and 166-MHz OverDrive processors; added Spec Change 6; revised Erratum 31; added Errata 47-55; added Spec Clarification 14-16; added Doc Changes 9 and 10.
May, 1996	-017	Part I: Added Erratum 48; revised Erratum 35; added Spec Clarification 15-17; revised Spec Clarification 5; added Doc Change 10. Part II: Added Spec change 21; added S-Spec 8; revised S-Spec 4; added Errata 67-68; revised Errata 28 and 44; added Spec Clarifications 30-35; revised Spec Clarification 15; added Doc Change 15-16. Part III: Added Errata 56-57; revised Errata 19 and 31; added Spec Clarifications 17-19; revised Spec Clarification 6; added Doc Changes 11 and 12.
June, 1996	-018	Part I: Added Spec Clarification 18. Part II: Revised Spec Changes 19 and 21; added Errata 15DP, 16DP and 18AP; revised Erratum 17AP; added Spec Clarification 36. Part III: Revised Spec Change 7; added Erratum 58; revised Erratum 57; added Spec Clarification 20.
July, 1996	-019	Part I: Added Spec Change 3; added Erratum 49; added Spec Clarification 19. Part II: Added Spec Change 22; revised Spec Change 20; added Errata 69; added Spec Clarifications 37 and 38; revised Spec Clarification 5 and 25. Part III: Added Spec Change 8; added Erratum 59; added Spec Clarification 21 and 22.

Note:

* Revision -015 was issued as revision -016.

August, 1996	-020	Part I: Added Erratum 50; revised Erratum 49; revised Spec Clarification 10. Part II: Added Spec Change 23; added Erratum 70; revised Erratum 69; revised Spec Clarification 20. Part III: Added Erratum 60; revised Erratum 59; revised Spec Clarification 10.
September, 1996	-021	Part I: Added Erratum 51; added Spec Clarification 20. Part II: Added Erratum 71; added Spec Clarification 39. Part III: Added Erratum 61; added Spec Clarification 23.
October, 1996	-022	Part I: Added Spec Clarification 21; revised Spec Clarification 5. Part II: Added Spec Clarification 40; revised Spec Clarification 15. Part III: Added Spec Clarification 24; revised Spec Clarification 6.

PREFACE

This document is an update to the specifications contained in the *Pentium Processor Family Developer's Manual*, (Order Number 241563-004). 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, and is divided into the following three parts:

- Part I: Specification Update for 60- and 66-MHz Pentium® Processors
- Part II: Specification Update for 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processors
- Part III: Specification Update for 63- and 83-MHz Pentium® OverDrive® Processors

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:

Family[1]	60- and 66-MHz Model 1[2]	75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Model 2[2]	63- and 83-MHz Model 3[2]
05h	01h (described in Part I)	02h (described in Part II)	03h (described in Part III)

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.

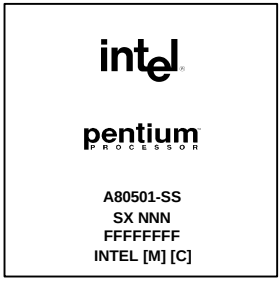
Part I:

Specification Update for 60- and 66-MHz Pentium® Processors

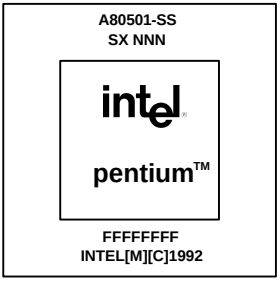
GENERAL INFORMATION

Top Markings

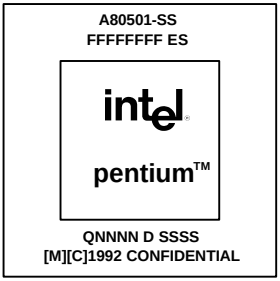
B1 Production Units – Top:



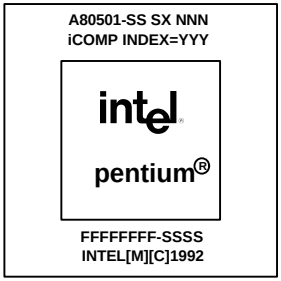
C1 Production Units – Top:



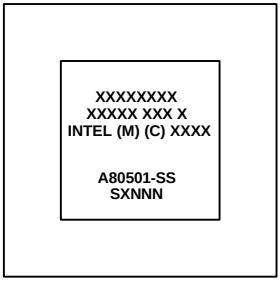
D1 Sample Units – Top:



D1 Production Units – Top:



D1 Production Units – Bottom:



NOTES:

- SS = Speed (MHz).
- SX NNN = S-Spec number.
- FFFFFFFF = FPO # (Test Lot Traceability #).
- For packages with heat spreaders, the inner line box defines the spreader edge.
- Ink Mark = All logo information on the heat spreader.
- Laser Mark = The two lines of information above and below the heat spreader. All bottomside information is laser mark.
- ES = Engineering Sample.
- QNNNN = Sample Specification number.
- SSSS = Serialization code.
- YYY = iCOMP® index (510 for 60-MHz and 567 for 66-MHz product).
- On the bottomside, the inner line box defines the edge of the metallic package lid.
- Bottomside, first two lines = Reserved for Intel internal use.
- Bottomside, third line = Copyright information; the last four digits of this line are reserved for Intel internal use

Basic 60- and 66-MHz Pentium® Processor Identification Information

CPUID									
Type	Family	Model	Stepping	Mfg. Stepping	Speed (MHz) Core/Bus	S-Spec	V_{CC}	T_{CASE}	Notes
0	5	1	3	B1'	50/50	Q0399	4.75V-5.25V	0°C-85°C	1, 2
0	5	1	3	B1'	60/60	Q0352	4.75V-5.25V	0°C-85°C	1
0	5	1	3	B1'	60/60	Q0400	4.75V-5.25V	0°C-75°C	1, 2
0	5	1	3	B1'	60/60	Q0394	4.75V-5.25V	0°C-80°C	2, 3
0	5	1	3	B1'	66/66	Q0353	4.90V-5.25V	0°C-75°C	1
0	5	1	3	B1'	66/66	Q0395	4.90V-5.25V	0°C-70°C	2, 3
0	5	1	3	B1"	60/60	Q0412	4.75V-5.25V	0°C-85°C	1
0	5	1	3	B1"	60/60	SX753	4.75V-5.25V	0°C-85°C	1
0	5	1	3	B1"	66/66	Q0413	4.90V-5.40V	0°C-75°C	1
0	5	1	3	B1"	66/66	SX754	4.90V-5.40V	0°C-75°C	1, 4
0	5	1	5	C1	60/60	Q0466	4.75V-5.25V	0°C-80°C	3
0	5	1	5	C1	60/60	SX835	4.75V-5.25V	0°C-80°C	3
0	5	1	5	C1	60/60	SZ949	4.75V-5.25V	0°C-80°C	3,5
0	5	1	5	C1	66/66	Q0467	4.90V-5.40V	0°C-70°C	3
0	5	1	5	C1	66/66	SX837	4.90V-5.40V	0°C-70°C	3
0	5	1	5	C1	66/66	SZ950	4.90V-5.40V	0°C-70°C	3,5
0	5	1	7	D1	60/60	Q0625	4.75V-5.25V	0°C-80°C	3
0	5	1	7	D1	60/60	SX948	4.75V-5.25V	0°C-80°C	3
0	5	1	7	D1	60/60	SX974	5.15V-5.40V	0°C-70°C	3
0	5	1	7	D1	60/60	---	4.75V-5.25V	0°C-80°C	3, 5, 6
0	5	1	7	D1	66/66	Q0626	4.90V-5.40V	0°C-70°C	3
0	5	1	7	D1	66/66	SX950	4.90V-5.40V	0°C-70°C	3
0	5	1	7	D1	66/66	Q0627	5.15V-5.40V	0°C-70°C	3
0	5	1	7	D1	66/66	SX949	5.15V-5.40V	0°C-70°C	3
0	5	1	7	D1	66/66	---	4.90V-5.40V	0°C-70°C	3, 5, 6

NOTES:

1. Non-heat spreader package.
2. These are engineering samples only.
3. Heat spreader package (see Specification Change #2).
4. 66 MHz B1" shipped after work week 34 of 1993 were tested to V_{CC} = 4.90V - 5.40V
5. This is a boxed Pentium processor
6. This part is not marked with a S-Spec number

Summary Table of Changes

The following table indicates the Specification Changes, Errata, Specification Clarifications, or Documentation Changes which apply to the 60- and 66-MHz Pentium® processor. 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.
PP##:	Please refer to Part II: Specification Update for 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processors.
Shaded:	This item is either new or modified from the previous version of the document.

NO.	B1	C1	D1	Refer to:	Plans	SPECIFICATION CHANGES
1	X	X	X	PP12	Doc	STI followed by FP instruction doesn't delay interrupt window
2	X	X	X	PP13	Doc	IDT limit violation causes GP fault, not interrupt 8
3	X	X	X	PP22	Doc	Leakage current I_{LI}/I_{LO} applies only in the valid logic states
NO.	**B1**	**C1**	**D1**	**Refer to:**	**Plans**	**ERRATA**
1	X				Fixed	BOFF# hold timing
2	X				Fixed	Incomplete initialization may flush the internal pipeline
3	X				Fixed	IV pin may not be asserted under certain conditions
4	X				Fixed	Testability writes to data TLB may store wrong parity
5	X				Fixed	LRU bits in the data cache TLBs are updated incorrectly
6	X				Fixed	A replacement writeback cycle may invade a locked sequence
7	X				Fixed	RUNBIST instruction generates incorrect BIST signature
8	X	X			Fixed	Data breakpoint mistakenly remembered on a faulty instruction
9	X	X	X		NoFix	RESET affects RUNBIST instruction execution in boundary scan
10	X	X			Fixed	Locked operation during instruction execution tracing may hang the processor

NO.	B1	C1	D1	Refer to:	Plans	ERRATA (Cont'd)
11	X	X			Fixed	Breakpoint or single-step may be missed for one instruction following STI
12	X	X			Fixed	Internal snoop problem due to reflection on address bus
13	X	X			Fixed	Internal parity error on uninitialized data cache entry
14	X	X			Fixed	Missing shutdown after an IERR#
15	X	X			Fixed	Processor core may not serialize on bus idle
16	X	X			Fixed	SMIACT# assertion during replacement writeback cycle
17	X	X	X	PP20	NoFix	Overflow undetected on some numbers on FIST
18	X	X	X	PP21	NoFix	Six operands result in unexpected FIST operation
19	X	X			Fixed	Snoop with table-walk violation may not invalidate snooped line
20	X	X		PP23	Fixed	Slight precision loss for floating-point divides on specific operand pairs
21	X	X	X		NoFix	Power-up BIST failure
22	X	X	X	PP24	NoFix	FLUSH#, INIT or Machine Check dropped due to floating-point exception
23	X	X	X	PP25	NoFix	Floating-point operations may clear Alignment Check bit
24	X	X	X	PP26	NoFix	CMPXCHG8B across page boundary may cause invalid opcode exception
25	X	X	X	PP27	NoFix	Single step debug exception breaks out of HALT
26	X	X	X	PP32	NoFix	EIP altered after specific FP operations followed by MOV Sreg, Reg
27	X	X	X	PP33	NoFix	WRMSR into illegal MSR does not generate GP fault
28	X	X	X	PP34	NoFix	Inconsistent data cache state from concurrent snoop and memory write
29	X	X	X	PP36	NoFix	Incorrect FIP after RESET
30	X	X	X	PP37	NoFix	Second assertion of FLUSH# not ignored
31	X	X	X	PP38	NoFix	Segment limit violation by FPU operand may corrupt FPU state
32	X	X	X	PP39	NoFix	FP exception inside SMM with pending NMI hangs system
33	X	X	X	PP42	NoFix	Incorrect decode of certain 0F instructions
34	X	X	X	PP43	NoFix	Data breakpoint deviations
35	X	X	X	PP44	NoFix	Event monitor countng discrepancies
36	X	X	X	PP45	NoFix	VERR type instructions causing page fault task switch with T bit set may corrupt CS:EIP
37	X	X	X	PP46	NoFix	BUSCHK# interrupt has wrong priority

NO.	B1	C1	D1	Refer to:	Plans	ERRATA (Cont'd)
38	X	X	X	PP51	NoFix	A fault causing a page fault can cause an instruction to execute twice
39	X	X	X	PP52	NoFix	Machine check exception pending, then HLT, can cause skipped or incorrect instruction, or CPU hang
40	X	X	X	PP53	NoFix	FBSTP stores BCD operand incorrectly If address wrap & FPU error both occur
41	X	X	X	PP54	NoFix	V86 interrupt routine at illegal privilege level can cause spurious pushes to stack
42	X	X	X	PP55	NoFix	Corrupted HLT flag can cause skipped or incorrect instruction, or CPU hang
43	X	X	X	PP56	NoFix	Benign exceptions can erroneously cause double fault
44	X	X	X	PP57	NoFix	Double fault counter may not increment correctly
45	X	X	X	PP59	NoFix	Short form of mov EAX/ AX/ AL may not pair
46	X	X	X	PP60	NoFix	Turning off paging may result in prefetch to random location
47	X	X	X	PP65	NoFix	TRST# not asynchronous
48	X	X	X	PP68	NoFix	Asserting TRST# pin or issuing JTAG instructions does not exit TAP Hi-Z state
49	X	X	X	PP69	NoFix	ADS# may be delayed after HLDA deassertion
50	X	X	X	PP70	NoFix	Stack underflow in IRET gives #GP, not #SS
51	X	X	X	PP71	NoFix	Performance monitoring pins PM[1:0] may count the events incorrectly
NO.	**B1**	**C1**	**D1**	**Refer to:**	**Plans**	**SPECIFICATION CLARIFICATIONS**
1	X	X	X		Doc	I/O trap restart in system management mode
2	X	X	X	PP6	Doc	Only one SMI# can be latched during SMM
3	X	X	X	PP11	Doc	SMIACT# handling during snoop writeback
4	X	X	X	PP14	Doc	EADS# recognition
5	X	X	X	PP15	Doc	Event monitor counters
6	X	X	X	PP16	Doc	KEN# sets cacheability for restarted cycles
7	X	X	X	PP17	Doc	NMI Signal Description
8	X	X	X	PP18	Doc	BTB behavior when entering SMM
9	X	X	X	PP19	Doc	SMI# Activation May Cause a Nested NMI Handling

NO.	B1	C1	D1	Refer to:	Plans	SPECIFICATION CLARIFICATIONS (Cont'd)
10	X	X	X	PP20	Doc	Exit from shutdown
11	X	X	X	PP21	Doc	Code breakpoints set on meaningless prefixes not guaranteed to be recognized
12	X	X	X	PP23	Doc	Resume flag should be set by software
13	X	X	X	PP24	Doc	Data breakpoints on INS delayed one iteration
14	X	X	X	PP26	Doc	CPUID feature flags
15	X	X	X	PP30	Doc	When L1 cache disabled, inquire cycles are blocked
16	X	X	X	PP31	Doc	Serializing operation required when one CPU modifies another CPU's code.
17	X	X	X	PP32	Doc	Cache test registers are modified during FLUSH#
18	X	X	X	PP36	Doc	Extra code break can occur on I/O or HLT instruction if SMI coincides
19	X	X	X	PP37	Doc	4-MByte page extensions
20	X	X	X	PP39	Doc	Recognizing INTR after its INTA cycle
21	X	X	X	PP40	Doc	Multiple interrupt delaying instructions may not delay more than one instruction
NO.	**B1**	**C1**	**D1**	**Refer to:**	**Plans**	**DOCUMENTATION CHANGES**
1	X	X	X	PP4	Doc	JMP cannot do a nested task switch, Volume 3, page 13-12
2	X	X	X	PP6	Doc	Interrupt sampling window, Volume 3, page 23-39
3	X	X	X		Doc	Flatness specification, Volume 1 Table 9-2
4	X	X	X	PP7	Doc	Errors in the detailed descriptions of FSUB, FSUBR, FDIV, FDIVR and related instructions
5	X	X	X	PP10	Doc	PUSHA, PUSHF, POPA & POPF can cause alignment faults
6	X	X	X	PP11	Doc	Corrections to BT, BTC, BTR and BTS instruction descriptions
7	X	X	X	PP12	Doc	Corrections for description of a stack overflow on interrupt to inner privilege case
8	X	X	X	PP13	Doc	FSETPM is like NOP, not like FNOP
9	X	X	X	PP14	Doc	Corrections in the pseudocode descriptions of the instructions CALL, IRET(D) & RET
10	X	X	X	PP15	Doc	Errors in 3 tables of special descriptor types

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the *Pentium Processor Family Developer's Manual,* Volume 1 and Volume 3, (Order Numbers 241428 and 241430, respectively), or the *Pentium Processor* datasheet, (Order Number 241595). All Specification Changes will be incorporated into future versions of the appropriate document(s).

1-3 Refer to Summary Table of Changes

ERRATA

1. BOFF# Hold Timing

PROBLEM: Silicon characterization indicates that the processor does not meet the specified hold time, 1.5nS, for the BOFF# signal. Data indicates that a minimum hold time of 2.0nS is required.

IMPLICATION: If a minimum hold time of 2.0nS is not met for the BOFF# input, the processor may drive undefined or incorrect cycles on to the external bus.

WORKAROUND: System must guarantee a minimum hold time of 2.0nS at the BOFF# input to the Pentium processor.

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

2. Incomplete Initialization May Flush the Internal Pipeline

PROBLEM: If a memory write occurs before the first branch instruction immediately after RESET, then the internal pipeline may get flushed unexpectedly. Assuming normal distribution of code space, this unexpected flush has the probability of 1 in 2^{26} of occurring.

IMPLICATION: The probability of this unexpected flush occurring is very low. When it happens, its only effect is to flush the internal pipeline and re-fetch the correct opcodes again. The instructions will still be executed correctly.

In the FRC (Functional Redundancy Checking) environment, where the clock-by-clock behavior of the processor needs to be checked deterministically, it may cause the system to report an error.

WORKAROUND: After RESET, ensure that the first write to memory occurs after a branch instruction.

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

3. IV Pin May Not Be Asserted Under Certain Conditions

PROBLEM: The IV pin is driven active by the Pentium processor to indicate that an instruction in the v-pipe has completed execution. In the following case, the IV pin may not get asserted:

When a mispredicted instruction (pair) reaches the execution stage, it will cause a pipeline flush. If in this clock, a fault is detected on the instruction in the u-pipe, the IV pin will not be asserted for a v-pipe instruction of the next instruction pair which is executed next.

IMPLICATION: The fact that the IV pin is not asserted under certain conditions will affect the reliability of the execution tracing data. It will also affect the performance monitoring event count for instructions executed in the v-pipe.

WORKAROUND: Disabling the v-pipe will allow execution tracing to work properly.

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

4. Testability Writes to Data TLB May Store Wrong Parity

PROBLEM: During testability writes to the data TLB, an incorrect tag parity may be computed and stored with the tag address. Subsequently, when this entry is read during a normal (non-testability) cycle, an internal parity error (IERR#) may be generated.

IMPLICATION: The internal parity error may occur only if a non-testability access is made to the same data TLB entry which had been previously written on a testability write. This problem will not show up if the data TLB is flushed after having been used for testability purposes using the TLB test registers.

WORKAROUND: Ensure that the data TLB is flushed after having been used for testability writes and before being used for normal operation.

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

5. LRU Bits in the Data Cache TLBs Are Updated Incorrectly

PROBLEM: Due to a circuit problem, the LRU bits for the data cache TLBs are updated incorrectly when both the u and v pipes access the same set. As the TLBs are organized as 4-Way set associative, more specifically this problem occurs when a u-pipe match is found on Way 0 and a v-pipe match is found on Way 1, or the u-pipe match is found on Way 2 and the v-pipe match is found on Way 3 of the same set.

IMPLICATION: The LRU bits are used to handle replacements in the TLB. In this specific case, the pseudo LRU mechanism is not strictly adhered to. Any performance degradation resulting from this is expected to be negligible.

WORKAROUND: None identified at this time.

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

6. A Replacement Writeback Cycle May Invade a Locked Sequence

PROBLEM: During a locked read-modify-write (RMW) sequence, if BOFF# or AHOLD is asserted before the write portion of the RMW sequence is completed, then the write cycle will be held off in the internal write buffer until the BOFF# or AHOLD signal is deasserted. During the time that the bus is backed off, if another locked instruction (i.e., with a LOCK prefix) enters the instruction pipeline and initiates a replacement writeback in the data cache, then as soon as the bus is freed, the writeback cycle due to the replacement writeback may be issued in front of the locked write cycle pending in the write buffer. After completion of the writeback cycle, the processor will issue the write cycle to complete the RMW sequence.

IMPLICATION: This problem will only affect those systems which do not expect a replacement writeback cycle in the middle of a locked RMW sequence. Furthermore, the timing of events needed for the above problem to manifest itself has a low probability of occurrence. Note that even though the bus cycles are reordered in this case, the correct bus cycles are run and should not cause any data coherency problems.

WORKAROUND: Do not assert BOFF# or AHOLD in between the read and the write portion of a locked read-modify-write sequence.

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

7. RUNBIST Instruction Generates Incorrect BIST Signature

PROBLEM: The Pentium processor TAP instruction, RUNBIST, generates incorrect BIST signature if issued while the processor is in Probe mode. The BIST result is also incorrect if the RUNBIST instruction is issued during a repeated MOVS (move data from string to string) instruction.

IMPLICATION: The BIST may report an incorrect signature indicating self-test failure even though the processor may not be faulty.

WORKAROUND: Do not issue the (TAP) RUNBIST instruction when the processor is in Probe mode, or during a repeated MOVS instruction .

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

8. Data Breakpoint Mistakenly Remembered on a Faulty Instruction

PROBLEM: In the following two cases, an instruction that has data breakpoints enabled and also generates a fault before completing execution may cause an unexpected data breakpoint to occur later in the code or may cause the software to hang:

CASE 1: For the first failing case to occur, the data breakpoint must be set on an instruction that is **not** a simple instruction (NOTE: simple instructions are those that are entirely hardwired and do not require any microcode control) and includes multiple memory references (e.g., a read operation followed by a write operation). In addition, this instruction also generates a fault before completing execution. The data breakpoint must be set on one memory operation (e.g., a data read portion), and the fault occurs on a subsequent memory operation (e.g., a data write portion). If later in the code, a fault during a repeated string operation is encountered, then a spurious debug exception will be reported first, followed by the fault for the string operation. This unexpected debug exception during the string operation may only occur if no other debug exception is taken before the string operation is executed.

CASE 2: For the second failing case, the data breakpoint must be set on the data read of iteration X of a repeated string instruction, and a fault must occur on the data write of the same iteration X. In this case, the Pentium processor takes the debug exception first, before handling the fault. When the faulting iteration is restarted after the debug exception is handled, the data breakpoint is again detected when the fault is encountered, and the processor returns to the debug exception handler. This will cause repeated entries into the debug exception handler for the same iteration. This loop may occur forever, unless the debug exception handler modifies the data breakpoints or the return instruction pointer.

IMPLICATION: In the first case, a data breakpoint may occur when it is not expected. In the second case, a repeated string instruction has a data breakpoint set on the read portion and a fault occurs on the corresponding write. This may cause the software to hang.

WORKAROUND: When setting data breakpoints, be aware of the above failure cases. Note that code and I/O breakpoints can be set properly and are not affected by this erratum.

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

9. RESET Affects RUNBIST Instruction Execution in Boundary Scan

PROBLEM: The Boundary Scan TAP instruction, RUNBIST, is affected by the assertion of the RESET pin. If the RESET pin is asserted while the processor is executing the TAP instruction, RUNBIST (TAP command field = 0111, and the TAP controller is in the Run-Test-Idle state), then the processor indicates a BIST failure.

IMPLICATION: The IEEE 1149.1-1990 specification states "the design of the component shall ensure that results of the self-tests executed in response to the RUNBIST instruction are not affected by signals received at the non-clock system input pins". The Pentium processor does not meet this requirement.

WORKAROUND: Ensure that the RESET pin is deasserted while the RUNBIST (TAP) instruction is executing.

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

10. Locked Operation During Instruction Execution Tracing May Hang the Processor

PROBLEM: During instruction execution tracing (TR12.TR bit set to '1') the processor can internally buffer up to two Branch Trace messages. If there is a possibility of a third Branch Trace message being delivered from the instruction being executed, the machine will stall in order to avoid overwriting either of the two messages that are already buffered. If this instruction is performing a "locked read-modify-write" operation, the processor can hang up due to internal service contention for the bus controller logic.

IMPLICATION: This problem does not affect normal operation (TR12.TR bit is not set). It only affects operation while the instruction execution tracing feature is enabled. A hardware RESET will be required to get the processor out of the deadlock condition if it occurs.

WORKAROUND: Do not enable instruction execution tracing.

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

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

PROBLEM: If the next instruction following STI is the target of a mispredicted branch, the processor may shut off the interrupt window for one instruction following STI. This will prevent breakpoints, single-step or other external interrupts from being recognized during this time.

IMPLICATION: The processor may not recognize NMI, SMI#, INIT, FLUSH#, BUSCHK#, R/S#, code/data breakpoint and single-step for one instruction after executing STI. This is not a problem unless breakpoints or single stepping is used. The only possible effect is that they may be missed.

WORKAROUND: Do not set a breakpoint on the next sequential instruction after STI. Alternatively, disabling branch prediction will prevent this problem from occurring.

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

12. Internal Snoop Problem Due to Reflection on Address Bus

PROBLEM: An internal snoop occurs in the following three cases:

1. An access is made to the code cache, and that access is a miss.
2. An access is made to the data cache, and that access is a miss or a writethrough.
3. There is an access to the page table/directory entries.

In all of these cases, the address used for the internal snoop is obtained from the input buffer of the address I/O buffers inside the chip. If there is signal reflection on the address lines A[31:5] which causes the setup/hold time inside the chip to be violated, then the internal snoop may fail.

When the reflection on the address I/O buffer is above (or below) the trip point of an input buffer (i.e., 1.5V for a TTL input) for a high to low (or low to high) transition, then the internal snoop address will not be valid until after the address reflection falls below (or transitions above) the trip point in the clock ADS# and address is driven. If the reflection causes the wrong snoop address A[31:5] to be latched inside the chip due to setup/hold time violation, then an incorrect internal snoop may occur.

IMPLICATION: When the failure occurs, a wrong cache line may be snooped causing either a line to remain valid when it should not, a valid line to become invalid when it should not, or an invalid line to be snooped.

In the first case, when a valid line should be invalidated and is not, a cache coherency problem between the code and data cache is possible (e.g., self-modifying code). In the second case, when a valid cache line is incorrectly made invalid, an unexpected writeback cycle may occur if the line was in the modified state, and an unexpected bus cycle may occur to re-allocate the cache line. The third case, where an invalid line is snooped, will not cause any detectable failure.

An additional failure mechanism can be seen if address lines A[11:5] are transitioning while being sampled. In this case, the internal snoop may fail, causing a tag parity error during the snoop resulting in an IERR# assertion.

The magnitude of the reflection is dependent upon the I/O buffer vs. transmission line impedance mismatch and the length/layout of trace (transmission line). There is sufficient margin built into the external timing specification for the address bus and I/O buffer models, and it is not expected that any failures will be seen on existing systems. In a lab environment, a failure was forced by adding a 12 inch coaxial cable (with no termination) along with a 330 Ohm pullup resistor on the address line to induce excessive reflection. Removing either the pullup resistor or adding termination to the coaxial cable eliminated the failure.

WORKAROUND: To avoid any internal snoop failures, ensure that the address A[31:5] setup and hold times are not violated at the processor's input due to reflection in the clock in which the processor drives the address bus and ADS# is asserted. NOTE: Meeting the setup and hold times at the processor's input is not necessary in the clocks where ADS# is not asserted.

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

13. Internal Parity Error on Uninitialized Data Cache Entry

PROBLEM: In the following case, an incorrect internal parity error (IERR#) may be reported due to an uninitialized entry in the data cache during a special qword (64 bit) read. This may occur if the qword read issued to the u-pipe is also followed by a v-pipe read, and the v-pipe read is to a different odd bank and different way than the u-pipe qword read. In addition, the u-pipe address corresponding to this different way must have invalid and uninitialized data. Under these conditions, the processor may check the invalid data for parity errors and incorrectly assert the IERR# pin.

IMPLICATION: After power-on reset, before the data cache is completely initialized, the processor may incorrectly report an IERR# and shutdown.

WORKAROUND: After power-on reset, initialize all entries in the data cache before the cache is enabled. The easiest way to do this is to invoke BIST (built-in self test) after reset. Alternatively, software can initialize the data cache by reading data from memory appropriately, or writing into all its locations through the cache test registers.

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

14. Missing Shutdown After an IERR#

PROBLEM: If an internal parity error is reported to the IERR# pin and a mispredicted branch or a trap/fault/interrupt with a higher priority than shutdown occurs, then the processor may not shutdown.

IMPLICATIONS: During the reporting of an internal parity error, the IERR# pin may go active without a processor shutdown. Note that IERR# due to an internal parity error will not occur unless the parity error is induced through parity reversal testing or if the chip is defective.

WORKAROUND: The system can latch an IERR# assertion at the processor clock edge and force a shutdown by asserting NMI or initializing the processor through RESET or INIT.

NOTE: IERR# is a glitch free signal, so no spurious assertions of IERR# will occur.

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

15. Processor Core May Not Serialize on Bus Idle

PROBLEM: Under rare circumstances, the processor may not serialize with the bus when the processor core is waiting for the bus to finish pending cycles and BOFF# is asserted. The processor will not reorder bus cycles; it may only start with the next event (fetching and executing subsequent instructions) before waiting for all pending bus cycles to complete. The following cases have been identified that may be affected by this:

SMI# pending	If BOFF# is used to back off a bus cycle while an SMI# is pending, the processor may assert SMIACT# before re-starting the aborted bus cycles.
Serializing instruction	If BOFF# is used to back off a bus cycle due to a serializing instruction, the processor may start executing the next instruction before restarting or completing the previous bus cycle. The processor, however, will not reorder any bus cycles for the new instruction in front of bus cycles for the previous instruction.
Invalidation during cache line fill	If BOFF# is used to back off a cache line fill and BOFF# occurs after the data has been returned to the processor but before the end of the line fill, an invalidation request during this time may result in the cache invalidation to occur before the line fill has completed, This may cause the cache line to remain in a valid state after the invalidation has completed. Note that if the invalidation request comes in via WBINVD or FLUSH#, the line fill would have to be backed off at least twice (or once for INVD) in order for the cache line to remain in a valid state after the invalidation has completed.
OUT instruction	If BOFF# is used to back off a bus cycle due to an OUT instruction, the processor may start executing the next instruction before the bus cycle due to OUT has completed (NOTE: The OUT instruction is similar to the serializing instructions except that it does not stop the prefetch of the subsequent instruction.) The processor, however, will not reorder any bus cycles for the new instruction in front of the OUT bus cycle.

IMPLICATION: This problem has only been observed in internal test vehicles. The events described above have different possible implications as follows:

SMI# pending	The processor may enter SMM before restarting the aborted bus cycle. The SMIACT# assertion may cause the restarted bus cycle to run to SMRAM space.
Serializing Instruction	Since the cycles are not reordered, a system should not encounter any problems unless it depends on the serializing instruction to cause an external event prior to execution of the next instruction.
Invalidation during cache line fill	In a rare instance, a cache line may remain in the valid state (E or S state) after the cache invalidation has completed.
OUT instruction	Since the cycles are not reordered, a system should not encounter any problems unless it depends on the OUT instruction to cause an external

event prior to execution of the next instruction. For example, an OUT instruction may be used to assert the A20M# signal prior to the next instruction. In this case, observed code has followed the OUT with an I/O read (IN) to ensure the signal is properly asserted. A second case, could be using an OUT instruction to configure/initialize and interrupt controller and follow it with STI to enable interrupts. Once again no failure would be observed. The controller would respond with the spurious interrupt vector.

WORKAROUND: Restrict the use of BOFF# for the described events. In addition, the SMI# pending event can be eliminated by locating SMRAM so that it does not shadow standard memory and does not require SMIACT# for memory decode. The OUT or serializing instruction events are eliminated if the next instruction does not depend on the result of the event before executing the instruction.

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

16. SMIACT# Assertion During Replacement Writeback Cycle

PROBLEM: If a data read cycle triggers a replacement writeback cycle and the SMI# signal is asserted prior to the first BRDY# of the read cycle, the processor may assert the SMIACT# signal prematurely.

Before the processor asserts SMIACT# in response to an SMI# request, it should complete all pending write cycles (including emptying the write buffers). However, if the appropriate conditions occur, the SMIACT# signal may get asserted during the replacement writeback cycle a few clocks after the last BRDY# of the read cycle.

IMPLICATION: If the SMIACT# signal is used by the system logic to decode SMRAM (e.g., SMRAM is located in an area that is overlaid on top of normal cacheable memory space), then the replacement writeback cycle with SMIACT# asserted could occur to SMM space. Systems that locate SMRAM in its own distinct memory space (non-overlaid) should not be affected by this.

WORKAROUND: Asserting FLUSH# simultaneously with SMI# will prevent this problem from occurring. In systems that overlay SMRAM over normal cacheable memory space, this is already a necessary requirement to maintain cache coherency.

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

17-18. Refer to Summary Table of Changes

19. Snoop with Table-Walk Violation May Not Invalidate Snooped Line

PROBLEM: If an internal snoop (as a result of ADS#) or external snoop (EADS#) with invalidation (INV) occurs coincident with a page table walk violation, the snoop may fail to invalidate the entry in the instruction cache. A page table walk violation occurs when the processor speculatively prefetches across a page boundary and this page is not accessible or not present. This violation results in a page fault if this code is executed. A page fault does not occur if the code is not executed.

For this erratum to occur, all the following conditions must be met:

1. A snoop with invalidation is run in the code cache. The snoop may be internal or external.
2. The Pentium processor is performing a page table walk to service an instruction TLB miss.
3. The page table walk results in a violation (this may or may not lead to a page or other fault due to a speculative fetch).

4. The EADS# of the external snoop or ADS# of the table update occur within the window of failure.

   The window is defined by:

   a. 2-4 clocks after BRDY# is returned for the page directory or table read.

   b. 2-n clocks after BRDY# is returned for the page directory or table read if the set address of a buffered write matches that of the instruction cache lookup. “n” is determined by the time to complete two new data write bus cycles from the data cache.

IMPLICATION: This erratum has not been observed on any system. It was found only through investigation of component schematics, and Intel has only duplicated it on a proprietary test system by forcing failure conditions using the internal test registers. The low frequency of occurrence is due to the way most systems operate; DMA devices snoop code 4 bytes at a time so that each line would get snooped and invalidated multiple times.

If this erratum occurs and a line is not invalidated in the instruction cache, then the instruction cache may have a coherency problem. As a result the processor may execute incorrect instructions leading to a GPF or an application error. This erratum affects only self-modifying code and bus masters/DMA devices. Due to necessary conditions, this erratum is expected to have an extremely low frequency of occurrence.

WORKAROUND: There are two workarounds. Because of the rarity of occurrence of this erratum, many OEMs may choose not to implement either workaround.

1. Rewrite the device driver for the DMA devices such that after DMA is complete, the instruction cache is invalidated using the TR5.cntl=11(flush) and CD=0 (code cache) bits.
2. Disable the L1 cache.

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

20. Refer to Summary Table of Changes

21. Power-Up BIST Failure

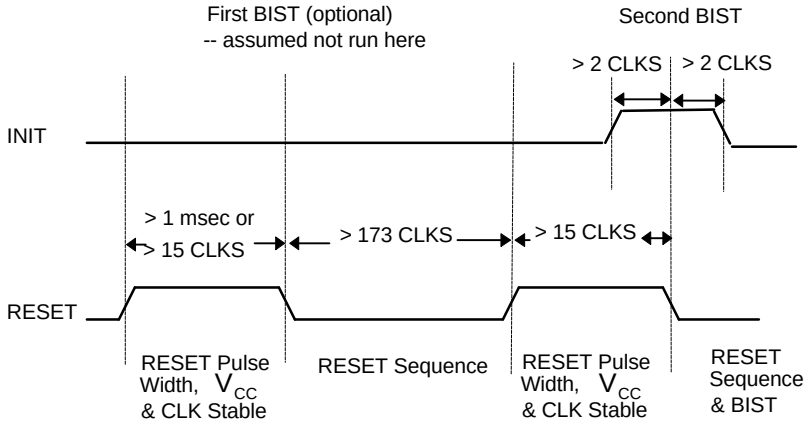
PROBLEM: The BIST (Built In Self Test) feature may fail.

IMPLICATION: Upon completion of BIST, a false failure may be reported. That is, a non-zero value may be returned in the EAX register, even though the part is operating correctly. As such, this erratum has no other product or system implications and is fully functional. To date, all observed false failures have returned a value of 20000H in EAX.

WORKAROUND: The root cause of the failure is not yet fully understood. As a result, workarounds suggested in this document are not guaranteed to fix the problem. To Intel’s knowledge, however, no BIST failures have been observed with use of either of the workarounds.

There are two workarounds:

1. Invoke a double RESET — All false BIST failures have occurred during cold RESET. In order to accurately execute BIST, a double RESET sequence may be performed as shown in the figure below:



The first RESET pulse is used to ensure that all logic used in the BIST test is correctly initialized; optionally, INIT may be driven high to enter BIST at the falling edge of the first RESET. In this case the results of the first BIST should be ignored.

After deassertion of the first RESET pulse, a minimum of 173 clocks must elapse before assertion of the second RESET. (Note that if BIST was entered at the end of the first RESET, this 173-clock requirement is satisfied by the time required to complete the first BIST.) During the second RESET, INIT is toggled high in order to run the 'real' BIST. This BIST correctly returns results in EAX.

The RESET and INIT pulses must meet normal specifications; RESET must be active at least 15 clocks, as described in the AC specifications (Tables 7-3 and 7-4 of the *Pentium Processor Family Developer's Manual*, Volume 1 and in the case of cold RESET must remain asserted for a minimum of 1 millisecond after V_{CC} and CLK have reached their proper AC/DC specifications. INIT must meet setup and hold times around the falling edge of RESET, and asynchronous INIT must be driven high at least 2 clocks before and held for at least 2 clocks after the falling edge of RESET.

2. Use RUNBIST — BIST may be run through the RUNBIST instruction, available through the Test Access Port as documented. In this fashion, BIST will return correct values.

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

22-51. Refer to Summary Table of Changes

SPECIFICATION CLARIFICATIONS

1. I/O Trap Restart in System Management Mode

The SMM revision identifier, as part of the state dump record in the SMRAM area, specifies the version of SMM and the extensions that are available on the processor. Bit 16 of the SMM revision identifier, if set, indicates that the processor supports I/O Trap Restart. Currently on the Pentium processor, bit 16 of the SMM revision identifier is '0' to indicate that this feature is not supported.

2-21. Refer to Summary Table of Changes

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the *Pentium Processor Family Developer's Manual*, Volumes 1 and 3. All Documentation Changes will be incorporated into a future version of the appropriate Pentium processor documentation.

1-2. Refer to Summary Table of Changes

3. Flatness Specification, Volume 1, Table 9-2

The flatness specification F of 0.127 millimeters is incorrectly located in the Min column in Table 9-2 of Volume 1 of the *Pentium Processor Family Developer's Manual*. It should be moved from the Min column to the Max column.

4-10. Refer to Summary Table of Changes

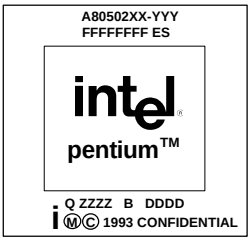
Part II:

Specification Update for 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processors

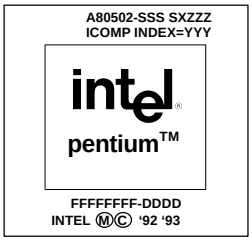
GENERAL INFORMATION

Top Markings

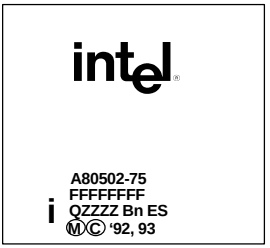
B-Step Engineering Samples:



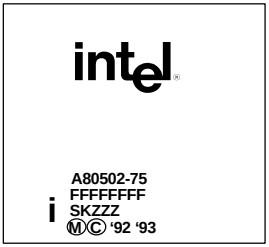
B-Step Production Units:



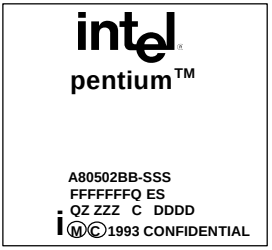
B-Step TCP Engineering Samples:



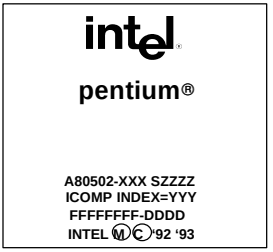
B-Step TCP Production Units:



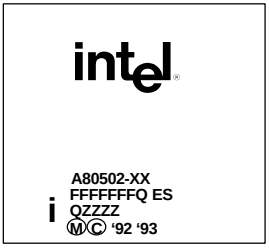
C-Step Engineering Samples:



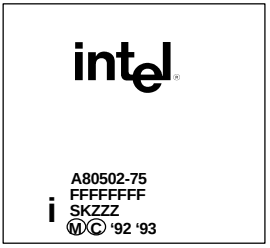
C and cB1-Step Production Units:



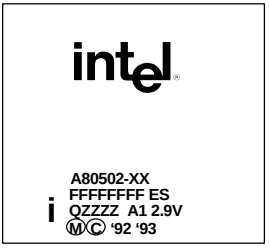
C-Step Engineering Sample TCP Units:



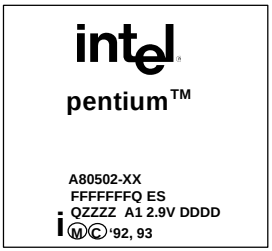
C-Step Production TCP Units:



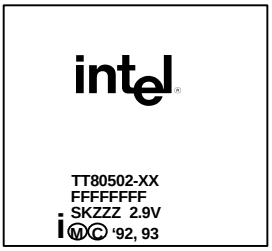
mA1-Step Engineering Samples TCP Units:



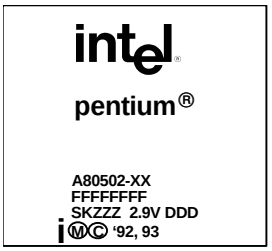
mA1-Step Engineering Sample SPGA Units:



mA1-Step Production TCP Units:



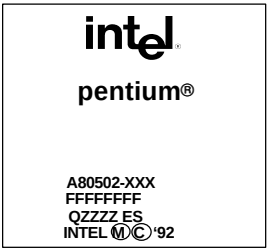
mA1-Step Production SPGA Units:



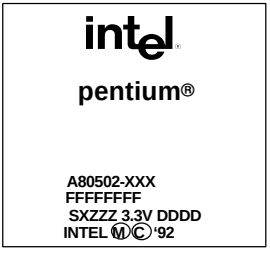
mcB1-Step Engineering Sample TCP Units:

mcB1-Step Production TCP Units:

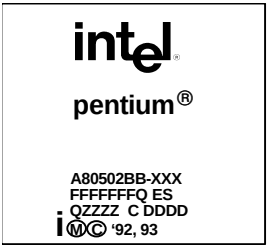
mcB1-Step Engineering Sample SPGA Units:



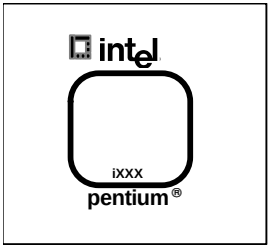
mcB1-Step Production SPGA Units:



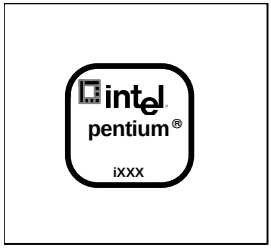
cC0-Step Engineering Sample Units:



cC0-Step Engineering Sample PPGA Units:

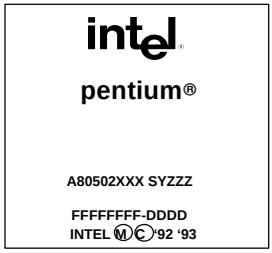


cC0-Step 200-MHz Engineering Sample PPGA Units:

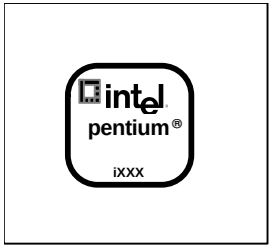


133 MHz cC0-Step Production Units:

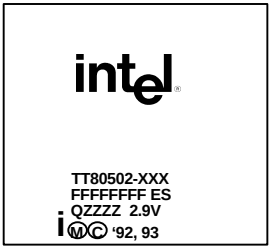
150 and 166-MHz cC0-Step Production Units:



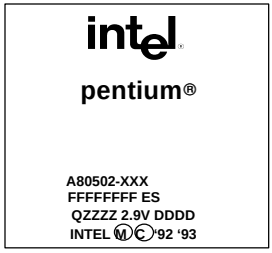
cC0-Step Production PPGA Units:



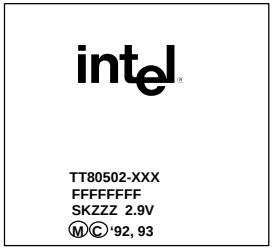
mA4-Step Engineering Sample TCP Units:



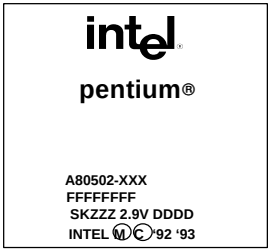
mA4-Step Engineering Sample SPGA Units:



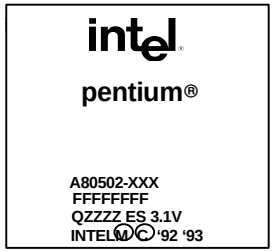
mA4-Step Production TCP Units:



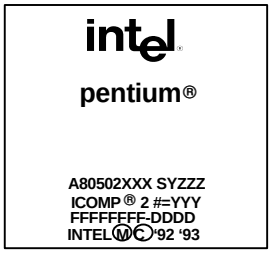
mA4-Step Production SPGA Units:



mcC0-Step Engineering Sample SPGA Units



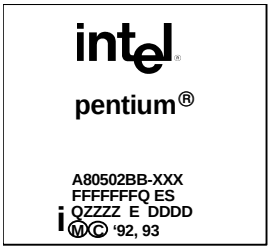
mcC0-Production SPGA Steppings:



mcC0-Step Engineering Sample TCP

mcC0-Step Production TCP Units:

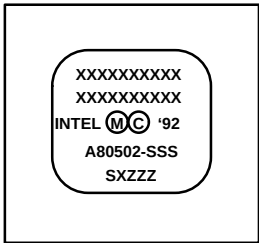
E0-Step Engineering Sample Units:



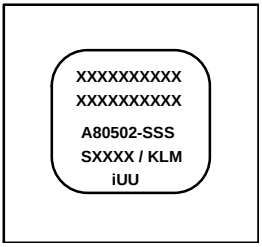
E0-Step Production Units:

Bottom Markings

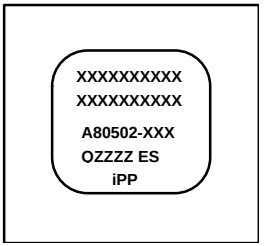
C-Step Production Units:
(before 7/95)



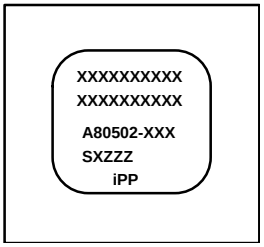
C and cB1-Step Production Units:
(after 7/95)



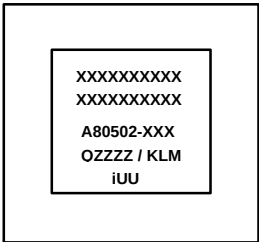
mcB1-Step Engineering Sample SPGA Units:



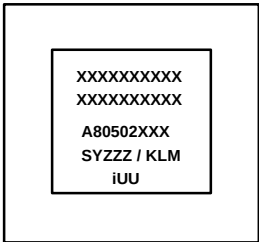
mcB1-Step Production SPGA Units:



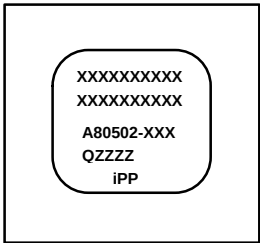
cC0-Step Engineering Sample Units:



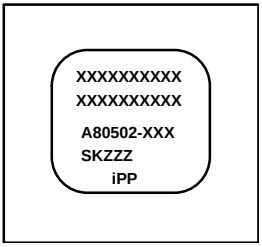
cC0-Step Production Units:



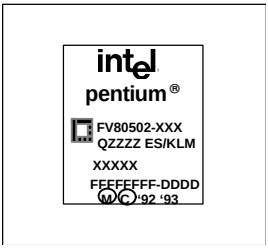
mA4-Step Engineering Sample SPGA Units:



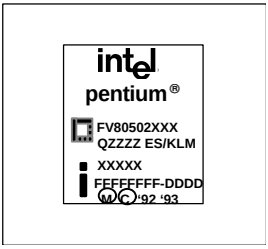
mA4-Step Production SPGA Units:



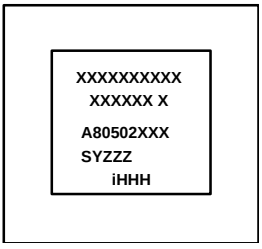
cC0-Step Engineering Sample PPGA Units:



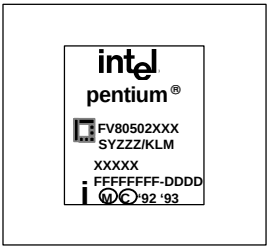
cC0-Step 200 MHz Engineering Sample PPGA Units:



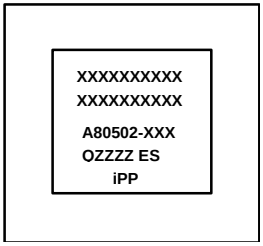
mcC0-Step Production SPGA Units:



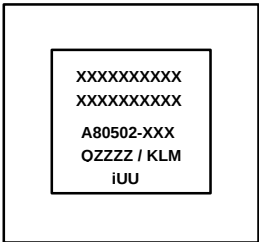
cC0-Step Production PPGA Units:



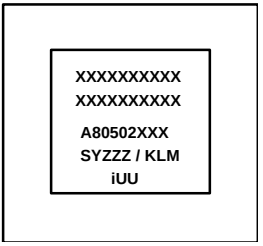
mcC0-Step Engineering Sample SPGA Units



E0-Step Engineering Sample Units

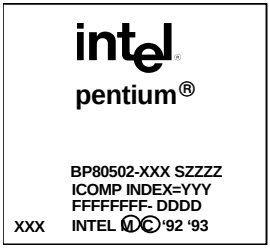


E0-Step Production Units:

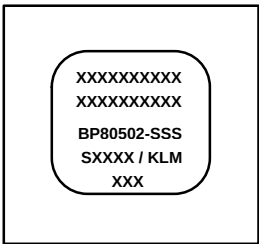


Boxed Pentium Processors with Attached Fan Heatsink

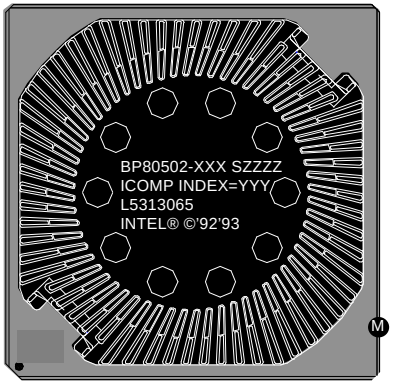
Top Side Marking:



Bottom Side Marking:



Fan Heatsink Base Marking:



NOTES:

- XX or XXX = Core Speed (MHz).
- BB = Bus speed (MHz)
- SXZZZ/SYZZZ/SZZZZ = Product S-Spec number
- FFFFFFFF = FPO # (Test Lot Traceability #).
- For packages with heat spreaders, the inner line box defines the spreader edge.
- Ink Mark = All logo information on the heat spreader.
- Laser Mark = The two lines of information above and below the heat spreader. All bottomside information is laser mark.
- ES = Engineering Sample.
- QZZZZ = Sample Specification number.
- DDDD = Serialization code.
- YYY = iCOMP® index (610 for 75-MHz and 735 for 90-MHz, 815 for 100-MHz, 1000 for 120-MHz, 1110 for 133-MHz, 1176 for 150-MHz and 1308 for 166-MHz parts.)
- TT = TCP Package, A = SPGA Package
- The bottom markings on the C and cB1-step production units will replace the existing bottom marking on C-step parts effective 7/95.
- UU = 75 or 133 for 75- or 133-MHz Pentium Processors, PP for all other speeds and MPP for mobile Pentium Processors
- K = V for VRE voltage range and S for standard voltage range
- L = M for min valid MD timings and S for min valid standard timings
- M = U is not tested for DP, is tested for UP and MP and S is tested for DP, UP and MP

Basic 75-, 90-,100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processor Identification Information

CPUID				Manufacturing Stepping	Speed (MHz) Core / Bus	S-Spec	Comments
Type	Family	Model	Stepping				
0	5	2	1	B1	75/50	Q0540	ES
2	5	2	1	B1	75/50	Q0541	ES
0	5	2	1	B1	90/60	Q0542	STD
0	5	2	1	B1	90/60	Q0613	VR
2	5	2	1	B1	90/60	Q0543	DP
0	5	2	1	B1	100/66	Q0563	STD
0	5	2	1	B1	100/66	Q0587	VR
0	5	2	1	B1	100/66	Q0614	VR
0	5	2	1	B1	75/50	Q0601	TCP Mobile
0	5	2	1	B1	90/60	SX879	STD
0	5	2	1	B1	90/60	SX885	MD
0	5	2	1	B1	90/60	SX909	VR
2	5	2	1	B1	90/60	SX874	DP, STD
0	5	2	1	B1	100/66	SX886	MD
0	5	2	1	B1	100/66	SX910	VR, MD
0	5	2	2	B3	90/60	Q0628	STD
0 or 2	5	2	2	B3	90/60	Q0611	STD
0 or 2	5	2	2	B3	90/60	Q0612	VR
0	5	2	2	B3	100/66	Q0677	VRE/MD
0	5	2	2	B3	75/50	Q0606	TCP Mobile
0	5	2	2	B3	75/50	SX951	TCP Mobile
0	5	2	2	B3	90/60	SX923	STD
0	5	2	2	B3	90/60	SX922	VR
0	5	2	2	B3	90/60	SX921	MD
2	5	2	2	B3	90/60	SX942	DP, STD
2	5	2	2	B3	90/60	SX943	DP, VR
2	5	2	2	B3	90/60	SX944	DP, MD
0	5	2	2	B3	90/60	SZ951[5]	STD
0	5	2	2	B3	100/66	SX960	VRE/MD

Basic 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processor Identification Information (Cont'd)

CPUID				Manufacturing Stepping	Speed (MHz) Core / Bus	S-Spec	Comments
Type	Family	Model	Stepping				
0 or 2	5	2	4	B5	75/50	Q0704	TCP Mobile
0 or 2	5	2	4	B5	75/50	Q0666	STD
0 or 2	5	2	4	B5	90/60	Q0653	STD
0 or 2	5	2	4	B5	90/60	Q0654	VR
0 or 2	5	2	4	B5	90/60	Q0655	MD
0 or 2	5	2	4	B5	100/66	Q0656	MD
0 or 2	5	2	4	B5	100/66	Q0657	VR, MD
0 or 2	5	2	4	B5	100/66	Q0658	VRE/MD
0	5	2	4	B5	120/60	Q0707	VRE/MD[1]
0	5	2	4	B5	120/60	Q0708	STD[1]
0	5	2	4	B5	75/50	SX975	TCP Mobile
0 or 2	5	2	4	B5	75/50	SX961	STD
0 or 2	5	2	4	B5	75/50	SZ977[5]	STD
0 or 2	5	2	4	B5	90/60	SX957	STD
0 or 2	5	2	4	B5	90/60	SX958	VR
0 or 2	5	2	4	B5	90/60	SX959	MD
0 or 2	5	2	4	B5	90/60	SZ978[5]	STD
0 or 2	5	2	4	B5	100/66	SX962	VRE/MD
0	5	2	5	C2	75/50	Q0725	TCP Mobile
0 or 2	5	2	5	C2	75/50	Q0700	STD
0 or 2	5	2	5	C2	75/50	Q0749	MD
0 or 2	5	2	5	C2	90/60	Q0699	STD
0 or 2	5	2	5	C2	100/50 or 66	Q0698	VRE/MD
0 or 2	5	2	5	C2	100/50 or 66	Q0697	STD
0	5	2	5	C2	120/60	Q0711	VRE/MD
0	5	2	5	C2	120/60	Q0732	VRE/MD
0	5	2	5	C2	133/66	Q0733	MD
0	5	2	5	C2	133/66	Q0751	MD
0	5	2	5	C2	133/66	Q0775	VRE/MD

Basic 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processor Identification Information (Cont'd)

CPUID				Manufacturing Stepping	Speed (MHz) Core / Bus	S-Spec	Comments
Type	Family	Model	Stepping				
0	5	2	5	C2	75/50	SK079	TCP Mobile
0 or 2	5	2	5	C2	75/50	SX969	STD
0 or 2	5	2	5	C2	75/50	SX998	MD
0 or 2	5	2	5	C2	75/50	SZ994[5]	STD
0 or 2	5	2	5	C2	75/50	SU070[6]	STD
0 or 2	5	2	5	C2	90/60	SX968	STD
0 or 2	5	2	5	C2	90/60	SZ995[5]	STD
0 or 2	5	2	5	C2	90/60	SU031[6]	STD
0 or 2	5	2	5	C2	100/50 or 66	SX970	VRE/MD
0 or 2	5	2	5	C2	100/50 or 66	SX963	STD
0 or 2	5	2	5	C2	100/50 or 66	SZ996[5]	STD
0 or 2	5	2	5	C2	100/50 or 66	SU032[6]	STD
0	5	2	5	C2	120/60	SK086	VRE/MD
0	5	2	5	C2	120/60	SX994	VRE/MD
0	5	2	5	C2	120/60	SU033[6]	VRE/MD
0	5	2	5	C2	133/66	SK098	MD
0	5	2	5	mA1[4]	75/50	Q0686	VRT[2], TCP
0	5	2	5	mA1[4]	75/50	Q0689	VRT[2], SPGA
0	5	2	5	mA1[4]	90/60	Q0694	VRT[2], TCP
0	5	2	5	mA1[4]	90/60	Q0695	VRT[2], SPGA
0	5	2	5	mA1[4]	75/50	SK089	VRT[2], TCP
0	5	2	5	mA1[4]	75/50	SK091	VRT[2], SPGA
0	5	2	5	mA1[4]	90/60	SK090	VRT[2], TCP
0	5	2	5	mA1[4]	90/60	SK092	VRT[2], SPGA
0 or 2	5	2	B	cB1[4]	120/60	Q0776	STD/no Kit[3]
0 or 2	5	2	B	cB1[4]	133/66	Q0772	STD/no Kit[3]
0 or 2	5	2	B	cB1[4]	133/66	Q0773	STD
0 or 2	5	2	B	cB1[4]	133/66	Q0774	VRE/MD, no Kit[3]
0 or 2	5	2	B	cB1[4]	120/60	SK110	STD/no Kit[3]
0 or 2	5	2	B	cB1[4]	133/66	SK106	STD/no Kit[3]

Basic 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processor Identification Information (Cont'd)

CPUID				Manufacturing Stepping	Speed (MHz) Core / Bus	S-Spec	Comments
Type	Family	Model	Stepping				
0 or 2	5	2	B	cB1[4]	133/66	S106J[7]	STD/no Kit[3]
0 or 2	5	2	B	cB1[4]	133/66	SK107	STD
0 or 2	5	2	B	cB1[4]	133/66	SU038[6]	STD/no Kit[3]
0	5	2	B	mcB1[4]	100/66	Q0884	VRT[2], TCP
0	5	2	B	mcB1[4]	120/60	Q0779	VRT[2], TCP
0	5	2	B	mcB1[4]	120/60	Q0808	3.3V, SPGA
0	5	2	B	mcB1[4]	100/66	SY029	VRT[2], TCP
0	5	2	B	mcB1[4]	120/60	SK113	VRT[2], TCP
0	5	2	B	mcB1[4]	120/60	SK118[7]	VRT[2], TCP
0	5	2	B	mcB1[4]	120/60	SX999	3.3V, SPGA
0 or 2	5	2	C	cC0	133/66	Q0843	STD/No Kit[3]
0 or 2	5	2	C	cC0	133/66	Q0844	STD
0 or 2	5	2	C	cC0	150/60	Q0835	STD
0 or 2	5	2	C	cC0	150/60	Q0878	STD, PPGA[9]
0 or 2	5	2	C	cC0	166/66	Q0836	VRE/No Kit[3]
0 or 2	5	2	C	cC0	166/66	Q0841	VRE
0 or 2	5	2	C	cC0	166/66	Q0886	VRE, PPGA[9]
0 or 2	5	2	C	cC0	166/66	Q0890	VRE, PPGA[9]
0	5	2	C	cC0	166/66	Q0949[8]	VRE, PPGA[9]
0 or 2	5	2	C	cC0	133/66	SY022	STD/ No Kit[3]
0 or 2	5	2	C	cC0	133/66	SY023	STD
0 or 2	5	2	C	cC0	133/66	SU073[6]	STD/ No Kit[3]
0 or 2	5	2	C	cC0	150/60	SY015	STD
0 or 2	5	2	C	cC0	150/60	SU071[6]	STD
0 or 2	5	2	C	cC0	166/66	SY016	VRE/ No Kit[3]
0 or 2	5	2	C	cC0	166/66	SY017	VRE
0 or 2	5	2	C	cC0	166/66	SU072[6]	VRE/ No Kit[3]
0	5	2	C	cC0	166/66	SY037[8]	VRE, PPGA[9]
0 or 2	5	2	C	cC0	200/66	Q0951F[10]	VRE, PPGA[9]
0	5	2	C	cC0	200/66	Q0951[8]	VRE, PPGA[9]

Basic 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processor Identification Information (Cont'd)

CPUID				Manufacturing Stepping	Speed (MHz) Core / Bus	S-Spec	Comments
Type	Family	Model	Stepping				
0 or 2	5	2	C	cC0	200/66	SY044	VRE, PPGA[9]
0	5	2	C	cC0	200/66	SY045[8]	VRE, PPGA[9]
0	5	7	0	mA4[4]	75/50	Q0848	VRT[2], TCP
0	5	7	0	mA4[4]	75/50	Q0851	VRT[2], SPGA
0	5	7	0	mA4[4]	90/60	Q0849	VRT[2], TCP
0	5	7	0	mA4[4]	90/60	Q0852	VRT[2], SPGA
0	5	7	0	mA4[4]	100/66	Q0850	VRT[2], TCP
0	5	7	0	mA4[4]	100/66	Q0853	VRT[2], SPGA
0	5	7	0	mA4[4]	75/50	SK119	VRT[2], TCP
0	5	7	0	mA4[4]	75/50	SK122	VRT[2], SPGA
0	5	7	0	mA4[4]	90/60	SK120	VRT[2], TCP
0	5	7	0	mA4[4]	90/60	SK123	VRT[2], SPGA
0	5	7	0	mA4[4]	100/66	SK121	VRT[2], TCP
0	5	7	0	mA4[4]	100/66	SK124	VRT[2], SPGA
0	5	2	C	mcC0[4]	100/66	Q0887	TCP/VRT[2]
0	5	2	C	mcC0[4]	120/60	Q0879	TCP/VRT[2]
0	5	2	C	mcC0[4]	120/60	Q0880	SPGA 3.1V
0	5	2	C	mcC0[4]	133/66	Q0881	TCP/VRT[2]
0	5	2	C	mcC0[4]	133/66	Q0882	SPGA 3.1V
0	5	2	C	mcC0[4]	150/60	Q024	TCP/VRT[2]
0	5	2	C	mcC0[4]	150/60	Q0906	TCP 3.1V
0	5	2	C	mcC0[4]	75/50	SY056	TCP/VRT[2]
0	5	2	C	mcC0[4]	100/66	SY020	TCP/VRT[2]
0	5	2	C	mcC0[4]	100/66	SY046	SPGA 3.1V
0	5	2	C	mcC0[4]	120/60	SY021	TCP/VRT[2]
0	5	2	C	mcC0[4]	120/60	SY027	SPGA 3.1V
0	5	2	C	mcC0[4]	120/60	SY030	SPGA 3.3V
0	5	2	C	mcC0[4]	133/66	SY019	TCP/VRT[2]
0	5	2	C	mcC0[4]	133/66	SY028	SPGA 3.1V
0	5	2	C	mcC0[4]	150/60	SY061	TCP/VRT[2]
0	5	2	C	mcC0[4]	150/60	SY043	TCP 3.1V

Basic 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processor Identification Information (Cont'd)

CPUID				Manufacturing Stepping	Speed (MHz) Core / Bus	S-Spec	Comments
Type	Family	Model	Stepping				
0	5	2	6	E0	75/50	Q0846	TCP Mobile
0 or 2	5	2	6	E0	75/50	Q0837	STD
0 or 2	5	2	6	E0	90/60	Q0783	STD
0 or 2	5	2	6	E0	100/50 or 66	Q0784	STD
0 or 2	5	2	6	E0	120/60	Q0785	VRE
0	5	2	6	E0	75/50	SY009	TCP Mobile
0 or 2	5	2	6	E0	75/50	SY005	STD
0 or 2	5	2	6	E0	75/50	SU097[5]	STD
0 or 2	5	2	6	E0	75/50	SU098[6]	STD
0 or 2	5	2	6	E0	90/60	SY006	STD
0 or 2	5	2	6	E0	100/50 or 66	SY007	STD
0 or 2	5	2	6	E0	100/50 or 66	SU110[5]	STD
0 or 2	5	2	6	E0	100/50 or 66	SU099[6]	STD
0 or 2	5	2	6	E0	120/60	SY033	STD
0 or 2	5	2	6	E0	120/60	SU100[6]	STD

NOTES:

- For a definition of STD, VR, VRE, MD, VRE/MD, refer to Specification Changes 2 and 18 in this document. ES refers to Engineering Samples. DP indicates that this part can only be used as a dual processor. 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 2 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.

1. T_{CASE} = 60°C.
2. VRT Intel's Voltage Reduction Technology: The V_{CC} for I/O is 3.3V, but the core V_{CC}, accounting for about 90% of power usage, is reduced to 2.9V, to reduce power consumption and heating.
3. No Kit means that part meets the specifications but is not tested to support 82498/82493 and 82497/82492 cache timings
4. STEPPING The cB1 stepping is logically equivalent to the C2-step, but on a different manufacturing process. The mcB1 step is logically equivalent to the cB1 step (except it does not support DP, APIC or FRC). The mcB1, mA1, mA4 and mcC0-steps also use Intel's VRT (Voltage Reduction Technology, see note 2 above) and are available in the TCP and/or SPGA package, primarily to support mobile applications. All mobile steppings are distinguished by an additional "m" prefix, for "mobile".
5. This is a boxed Pentium processor without the attached fan heatsink.
6. This is a boxed Pentium processor with an attached fan heatsink.
7. These parts do not support boundary scan. S106J was previously marked (and is the same as) SK106J.
8. DP, FRC and APIC features are not supported on these parts.
9. These parts are packaged in the Plastic Pin Grid Array (PPGA) package. For additional specifications of this package, see specification clarifications 27 and 28.
10. Some Q0951F units are marked on the bottom side with spec number Q0951 and with an additional line immediately underneath spelling out "Full Feature" to properly identify the unit.

Summary Table of Changes

The following table indicates the Specification Changes, S-Specs, Errata, Specification Clarifications or Documentation Changes, which apply to the listed 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-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 only to B- and C-step 75-MHz Pentium processor mobile-specific errata in a TCP package.
Shaded:	This item is either new or modified from the previous version of the document.

NO.	B1	B3	B5	C2	mA1	cB1	mcB 1	cC 0	mA 4	mcC 0	E0	Plans	SPECIFICATION CHANGES
1	X	X	X	X		X						Doc	Mixing steppings in dual processing mode
2	X	X	X	X		X		X				Doc	MD/VR/VRE specifications
3			X	X		X		X			X	Doc	Lock Step APIC operation specification
4				X								Doc	120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0732, Q0733, Q0751, Q0775, SK086, SX994, SK098, SU033) do not support dual processing
5				X								Doc	120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) do not support FRC
6				X								Doc	120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) V_{CC} to CLK startup specification
7				X								Doc	120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) current leakage on PICD1 pin

NO.	B1	B3	B5	C2	mA1	cB1	mcB1	cC0	mA4	mcC0	E0	Plans	SPECIFICATION CHANGES (Cont'd)
8					X		X		X	X		Doc	Pentium® processors with Voltage Reduction Technology do not support the APIC, dual processing, upgrade, or FRC
9					X		X		X	X		Doc	SPGA T_{CASE} specification is improved to 85 °C
10	Incorporated in the latest revision of the datasheet											Doc	Maximum power specification for mA1 and mA4 parts
11	X	X	X	X	X	X	X	X	X	X	X	Doc	V_{CC} measurement specification
12	X	X	X	X	X	X	X	X	X	X	X	Doc	STI followed by FP instruction doesn't delay interrupt window
13	X	X	X	X	X	X	X	X	X	X	X	Doc	IDT limit violation causes GP fault, not interrupt 8
14	X	X	X	X		X		X			X	Doc	DP not supported in FRC systems
15	Incorporated in the latest revision of the datasheet											Doc	Decreased maximum power specification for mcB1 parts
16	X	X	X	X		X		X			X	Doc	PICD0-1 valid delay (resistor pull-up & V_{OL})
17								X			X	Doc	PHITM# timing change
18								X			X	Doc	Min delay timings
19	X	X	X	X	X	X	X	X	X	X	X	Doc	Requirements for STPCLK# re-assertion recognition
20								X				Doc	Power specifications for 150, 166 and 200 MHz parts
21								X				Doc	Boundary scan min valid-delay timings (t_{53} & t_{55})
22	X	X	X	X	X	X	X	X	X	X	X	Doc	Leakage current I_{LI}/I_{L0} applies only in the valid logic states
23										X		Doc	150 MHz active power dissipation (typical) change
NO.	**B1**	**B3**	**B5**	**C2**	**mA1**	**cB1**	**mcB1**	**cC0**	**mA4**	**mcC0**	**E0**	**Plans**	**S-SPECS**
1					X				X			Fix	t_{6a}, t_{6b}, max valid delay A31-A3, BE7#-BE0#, ADS#, LOCK#
2					X				X			Fix	Minimum required voltage separation between V_{CC3} and V_{CC2}
3						X	X					Fixed	V_{IH} for TRST#
4							X					Fix	V_{IL} for BF and BF1 is reduced
5										X			Boundary scan timing changes
6										X			SPGA V_{CC2} supply voltage change
7					X				X			Doc	AC specifications for the Pentium processor with Voltage Reduction Technology
8							X			X		Fix	Reduced V_{IL} for TCK
NO.	**B1**	**B3**	**B5**	**C2**	**mA1**	**cB1**	**mcB1**	**cC0**	**mA4**	**mcC0**	**E0**	**Plans**	**ERRATA**
1	X	X	X									Fixed	Branch trace messages during lock cycles

2	X	X	X									Fixed	Breakpoint or single-step may be missed for one instruction following STI
3	X	X	X									Fixed	I/O restart does not function during single-stepping or data breakpoint exceptions

NO.	B1	B3	B5	C2	mA1	cB1	mcB 1	cC 0	mA 4	mcC 0	E0	Plans	ERRATA (Cont'd)
4	X	X	X									Fixed	NMI or INIT in SMM with I/O restart during single-stepping
5	X	X	X									Fixed	SMI# and FLUSH# during shutdown
6	X	X	X									Fixed	No shutdown after IERR#
7	X	X	X									Fixed	FLUSH# with a breakpoint pending causes false DR6 values
8	X											Fixed	Processor core may not serialize on bus idle
9	X	X	X	X	X	X						Fixed	SMIACT# premature assertion during replacement writeback cycle
10	Replaced by Specification Change 19											NoFix	STPCLK# deassertion not recognized for 5 CLKs after BRDY# returned
11	X	X	X									Fixed	Future Pentium® OverDrive® Processor FERR# Contention in Two-Socket Systems
12	X											Fixed	Code cache lines are not invalidated if snooped during AutoHALT or stop grant states
13	X											Fixed	STPCLK# assertion during execution of the HALT instruction hangs system
14	X	X	X	X	X	X	X	X	X	X	X	NoFix	NMI or INIT during HALT within an SMM may cause large amount of bus activity
15	X	X	X	X	X	X	X	X	X	X	X	NoFix	RUNBIST restrictions when run through boundary scan circuitry
16	X	X	X	X		X		X			X	NoFix	FRC mode miscompare due to uninitialized internal register
17	Replaced by Specification Change 19											NoFix	STPCLK# restrictions during EWBE#
18	X	X	X									Fixed	Multiple allocations into branch target buffer
19	X	X	X									Fixed	100-MHz REP MOVS speed path
20	X	X	X									Fixed	Overflow undetected on some numbers on FIST
21	X	X	X									Fixed	Six operands result in unexpected FIST operation
22	X											Fixed	Snoop with table-walk violation may not invalidate snooped line
23	X	X										Fixed	Slight precision loss for floating-point divides on specific operand pairs
24	X	X	X									Fixed	FLUSH#, INIT or machine check dropped due to floating-point exception
25	X	X	X	X	X	X	X		X			Fixed	Floating-point operations may clear alignment check bit
26	X	X	X	X	X	X	X		X			Fixed	CMPXCHG8B across page boundary may cause invalid opcode exception
27	X	X	X									Fixed	Single-step debug exception breaks out of HALT

28	X	X	X	X	X	X	X	X	X	X	X	NoFix	Branch trace message corruption
29	X	X	X	X		X						Fixed	FRC lock-step failure during APIC write

NO.	B1	B3	B5	C2	mA1	cB1	mcB 1	cC 0	mA 4	mcC 0	E0	Plans	ERRATA (Cont'd)
30	X	X	X	X	X	X	X		X			Fixed	BE4#-BE0# sampled incorrectly at Min Vih
31	X	X	X	X		X						Fixed	Incorrect PCHK# output during boundary scan if in DP mode
32	X	X	X	X	X	X	X		X			Fixed	EIP altered after specific FP operations followed by MOV Sreg, Reg
33	X	X	X	X	X	X	X	X	X	X	X	NoFix	WRMSR into illegal MSR does not generate GP Fault
34	X	X	X									Fixed	Inconsistent data cache state from concurrent snoop and memory write
35	X	X	X									Fixed	BE3#-BE0# not driven during boundary scan if RESET high
36	X	X	X	X	X	X	X	X	X	X	X	NoFix	Incorrect FIP after RESET
37	X	X	X	X	X	X	X	X	X	X	X	NoFix	Second assertion of FLUSH# not ignored
38	X	X	X	X	X	X	X	X	X	X	X	NoFix	Segment limit violation by FPU operand may corrupt FPU state
39	X	X	X	X	X	X	X	X	X	X	X	NoFix	FP exception inside SMM with pending NMI hangs system
40	X	X	X	X	X	X	X					Fixed	Current in Stop Clock state exceeds specification
41	X	X	X	X	X	X	X		X		X	Fixed	STPCLK# buffer samples incorrectly during boundary scan testing
42	X	X	X	X	X	X	X					Fixed	Incorrect decode of certain 0F instructions
43	X	X	X	X	X	X	X	X	X	X	X	NoFix	Data breakpoint deviations
44	X	X	X	X	X	X	X	X	X	X	X	NoFix	Event monitor counting discrepancies
45	X	X	X	X	X	X	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	X	X	X	X	X	X	NoFix	BUSCHK# interrupt has wrong priority
47	X	X	X	X	X				X		X	Fixed	BF and CPUTYP buffers sample incorrectly during boundary scan testing
48	X	X	X	X	X	X	X	X	X	X	X	NoFix	Matched but disabled data breakpoint can be lost by STPCLK# assertion.
49	X	X	X	X	X	X	X	X	X	X	X	NoFix	STPCLK# ignored in SMM when INIT or NMI pending
50	X	X	X	X	X	X	X	X	X	X	X	NoFix	STPCLK# pullup not engaged at RESET
51	X	X	X	X	X	X	X	X	X	X	X	Fix	A fault causing a page fault can cause an instruction to execute twice

52	X	X	X	X	X	X	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	X	X	X	X	X	X	NoFix	FBSTP stores BCD operand incorrectly If address wrap & FPU error both occur
54	X	X	X	X	X	X	X	X	X	X	X	NoFix	V86 interrupt routine at illegal privilege level can cause spurious pushes to stack

NO.	B1	B3	B5	C2	mA1	cB1	mcB 1	cC 0	mA 4	mcC 0	E0	Plans	ERRATA (Cont'd)
55	X	X	X	X	X	X	X	X	X	X	X	NoFix	Corrupted HLT flag can cause skipped or incorrect instruction, or CPU hang
56	X	X	X	X	X	X	X	X	X	X	X	NoFix	Benign exceptions can erroneously cause double fault
57	X	X	X	X	X	X	X	X	X	X	X	NoFix	Double fault counter may not increment correctly
58					X		X		X	X		NoFix	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	X	X	X	X	X	X	NoFix	Short form of mov EAX/ AX/ AL may not pair
60	X	X	X	X	X	X	X	X	X	X	X	NoFix	Turning off paging may result in prefetch to random location
61	X	X	X	X	X	X	X	X	X	X	X	NoFix	STPCLK# or FLUSH# after STI
62	X	X	X	X	X	X	X	X	X	X	X	NoFix	REP string instruction not interruptable by STPCLK#
63	X	X	X	X	X	X	X	X	X	X	X	NoFix	Single step may not be reported on first instruction after FLUSH#
64	X	X	X	X		X		X			X	NoFix	Double fault may generate illegal bus cycle
65	X	X	X	X	X	X	X	X	X	X	X	NoFix	TRST# not asynchronous
66	X	X	X	X	X	X	X	X	X	X	X	NoFix	STPCLK# on RSM to HLT causes non-standard behavior
67	X	X	X	X	X	X	X	X	X	X	X	NoFix	Code cache dump may cause wrong IERR#
68	X	X	X	X	X	X	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	X	X	X	X	X	X	NoFix	ADS# may be delayed after HLDA deassertion
70	X	X	X	X	X	X	X	X	X	X	X	NoFix	Stack underflow in IRET gives #GP, not #SS
71	X	X	X	X	X	X	X	X	X	X	X	NoFix	Performance monitoring pins PM[1:0] may count the events incorrectly
1DP	X	X	X									Fixed	Problem with external snooping while two cycles are pending on the bus
2DP	X	X	X									Fixed	STPCLK# assertion and the stop grant bus cycle
3DP	X	X	X									Fixed	External snooping with AHOLD asserted may cause processor to hang
4DP	X	X	X									Fixed	Address parity check not supported in dual processing mode

5DP	X	X										Fixed	Inconsistent cache state may result from interprocessor pipelined READ into a WRITE
6DP	X	X	X									Fixed	Processors hang during Zero WS, pipelined bus cycles
7DP	X	X	X									Fixed	Bus lock-up problem in a specific dual processing mode sequence
8DP	X	X	X	X		X						Fixed	Incorrect assertion of PHITM# without PHIT#
9DP	X	X	X	X		X						Fixed	Double issuance of read cycles
10DP	X	X	X	X		X						Fixed	Line invalidation may occur on read or prefetch cycles

NO.	B1	B3	B5	C2	mA1	cB1	mcB 1	cC 0	mA 4	mcC 0	E0	Plans	ERRATA (Cont'd)
11DP	X	X	X	X		X		X			X	NoFix	EADS# or floating ADS# may cause extra invalidates
12DP	X	X	X	X		X						Fixed	HOLD and BOFF# during APIC cycle may cause dual processor arbitration problem
13DP	X	X	X	X		X						Fixed	System hang after hold during local APIC 2nd INTA cycle
14DP	X	X	X	X		X		X			X	NoFix	External snoop can be incorrectly invalidated
15DP	X	X	X	X		X		X			X	NoFix	STPCLK# re-assertion recognition constraint with DP
16DP	X	X	X	X		X		X			X	NoFix	Second assertion of FLUSH# during flush acknowledge cycle may cause hang
1AP	X	X	X									Fixed	Remote read message shows valid status after a checksum error
2AP	X	X	X									Fixed	Chance of clearing an unread error in the error register
3AP	X	X	X									Fixed	Writes to error register clears register
4AP	X	X	X									Fixed	Three interrupts of the same priority causes lost local interrupt
5AP	X	X	X									Fixed	APIC bus synchronization lost due to checksum error on a remote read message
6AP	X	X	X									Fixed	HOLD during a READ from local APIC register may cause incorrect PCHK#
7AP	X	X	X									Fixed	HOLD during an outstanding interprocessor pipelined APIC cycle hangs processor
8AP	X	X	X									Fixed	PICCLK reflection may cause an APIC checksum error
9AP	X	X	X	X		X		X			X	NoFix	Spurious interrupt in APIC through local mode
10AP	X	X	X									Fixed	Potential for lost interrupts while using APIC in through Local mode
11AP	X	X	X	X		X						Fixed	Back to back assertions of HOLD or BOFF# may cause lost APIC write cycle
12AP	X	X	X	X		X		X				Fix	System hangs when BOFF# is asserted during second internal INTA cycle
13AP	X	X	X	X		X		X			X	Fix	APIC pipeline cycle during cache linefill causes restarted cycle to lose its attribute
14AP	X	X	X	X		X		X			X	NoFix	INIT and SMI via the APIC three-wire bus may be lost
15AP								X			X	Fix	IERR# in FRC lock-step mode during APIC write
16AP	X	X	X	X	X	X	X	X			X	Fix	Inadvertent BRDY# during external INTA cycle with BOFF#

17AP	X	X	X	X		X		X			X	Fix	APIC read cycle may not complete upon assertion of BOFF# and HOLD
18AP	X	X	X	X		X		X			X	NoFix	PICCLK must toggle for at least twenty cycles before RESET
1TCP	X											Fixed	CPU may not reset correctly due to floating FRCMC# pin
2TCP	X			X	X		X		X	X	X	Fix	BRDY# does not have buffer selection capability

NO.	B1	B3	B5	C2	mA1	cB1	mcB1	cC0	mA4	mcC0	E0	Plans	ERRATA (Cont'd)
NO.	B1	B3	B5	C2	mA1	cB1	mcB1	cC0	mA4	mcC0	E0	Plans	SPECIFICATION CLARIFICATIONS
1	X	X	X	X		X		X			X	Doc	Pentium processor's response to startup and init IPIs
2	X	X	X	X		X		X			X	Doc	APIC timer use clarification
3	X	X	X									Fixed	PICCLK reflection may cause APIC checksum errors and dropped IPIs
4	X	X	X									Fixed	Boundary scan RUNBIST register requires initialization prior to use
5	X	X	X	X	X	X	X	X	X	X	X	Doc	Pullup/pulldown resistor values for some configuration pins
6	X	X	X	X	X	X	X	X	X	X	X	Doc	Only one SMI# can be latched during SMM
7	X	X	X	X		X		X			X	Doc	The stop clock state is not supported in FRC mode
8	X	X	X	X	X	X	X	X	X	X	X	Doc	Nominal V_{CC} for V_{IH3}
9	X	X	X	X	X	X	X	X	X	X	X	Doc	Branch trace message discrepancies
10	X	X	X	X		X		X			X	Doc	APIC 8-bit access
11	X	X	X	X	X	X	X	X	X	X	X	Doc	SMIACT# handling during snoop writeback
12	X	X	X	X							X	Doc	Package thermal specifications
13	X	X	X	X		X		X			X	Doc	LOCK prefix excludes APIC memory space
14	X	X	X	X	X	X	X	X	X	X	X	Doc	EADS# recognition
15	X	X	X	X	X	X	X	X	X	X	X	Doc	Event monitor counters
16	X	X	X	X	X	X	X	X	X	X	X	Doc	KEN# sets cacheability for restarted cycles
17	X	X	X	X	X	X	X	X	X	X	X	Doc	NMI signal description
18	X	X	X	X	X	X	X	X	X	X	X	Doc	BTB behavior when entering SMM
19	X	X	X	X	X	X	X	X	X	X	X	Doc	SMI# activation may cause a nested NMI handling
20	X	X	X	X	X	X	X	X	X	X	X	Doc	Exit from shutdown
21	X	X	X	X		X		X			X	Doc	Power dissipation (Max) clarification
22	X	X	X	X	X	X	X	X	X	X	X	Doc	Code breakpoints set on meaningless prefixes not guaranteed to be recognized
23	X	X	X	X	X	X	X	X	X	X	X	Doc	Resume flag should be set by software
24	X	X	X	X	X	X	X	X	X	X	X	Doc	Data breakpoints on INS delayed one iteration
25	X	X	X	X	X	X	X	X	X	X	X	Doc	Supported bus fractions and frequency ranges
26	X	X	X	X	X	X	X	X	X	X	X	Doc	CPUID feature flags

NO.	B1	B3	B5	C2	mA1	cB1	mcB 1	cC 0	mA 4	mcC 0	E0	Plans	SPECIFICATION CLARIFICATIONS (Cont'd)
27								X				Doc	PPGA package thermal specifications
28								X				Doc	PPGA package dimensions
29						X		X				Doc	Package thermal specifications
30	X	X	X	X	X	X	X	X	X	X	X	Doc	When L1 cache disabled, inquire cycles are blocked
31	X	X	X	X	X	X	X	X	X	X	X	Doc	Serializing operation required when one CPU modifies another CPU's code.
32	X	X	X	X	X	X	X	X	X	X	X	Doc	Cache test registers are modified during FLUSH#
33	X	X	X	X	X	X	X	X	X	X	X	Doc	For correct translations, the TLB should be flushed after the PSE bit in CR4 is set
34	X	X	X	X		X		X			X	Doc	PCHK# V_{OL} level for DP designs
35	X	X	X	X		X		X			X	Doc	When APIC enabled, its 4K block should not be used in regular memory
36	X	X	X	X	X	X	X	X	X	X	X	Doc	Extra code break can occur on I/O or HLT instruction if SMI coincides
37	X	X	X	X	X	X	X	X	X	X	X	Doc	4-MByte page extensions
38	X	X	X	X	X	X	X	X	X	X	X	Doc	External snoop difference between UP and DP when L1 cache disabled
39	X	X	X	X	X	X	X	X	X	X	X	Doc	Recognizing INTR after its INTA cycle
40	X	X	X	X	X	X	X	X	X	X	X	Doc	Multiple interrupt delaying instructions may not delay more than one instruction

NO.	B1	B3	B5	C2	mA1	cB1	mcB 1	cC 0	mA 4	mcC 0	E0	Plans	DOCUMENTATION CHANGES
1	X	X	X	X	X	X	X	X	X	X	X	Doc	Package dimensions table, Volume 1, page 25-5
2						X	X	X		X		Doc	Boundary scan chain bit sequence, Volume 1, page 27-2
3	X	X	X	X	X	X	X	X	X	X	X	Doc	BF1 pin location, Volume 1, page 17-3 and page 17-5
4	X	X	X	X	X	X	X	X	X	X	X	Doc	JMP cannot do a nested task switch, Volume 3, page 13-12
5					X		X		X	X		Doc	Incorrect TCP pinout drawing
6	X	X	X	X	X	X	X	X	X	X	X	Doc	Interrupt sampling window, Volume 3, page 23-39
7	X	X	X	X	X	X	X	X	X	X	X	Doc	Errors in the detailed descriptions of FSUB, FSUBR, FDIV, FDIVR and related instructions
8	X	X	X	X	X	X	X	X	X	X	X	Doc	Figures of TR1 and TR12, Volume 1, page 33-13 and page 33-16.
9	X	X	X	X		X		X			X	Doc	APIC AC specification t_{63}, Volume 1, page 23-12, 23-17 and 23-22
10	X	X	X	X	X	X	X	X	X	X	X	Doc	PUSHA, PUSHF, POPA & POPF can cause alignment faults

NO.	B1	B3	B5	C2	mA1	cB1	mcB 1	cC 0	mA 4	mcC 0	E0	Plans	DOCUMENTATION CHANGES (Cont'd)
11	X	X	X	X	X	X	X	X	X	X	X	Doc	Corrections to BT, BTC, BTR and BTS instruction descriptions
12	X	X	X	X	X	X	X	X		X	X	Doc	Corrections for description of a stack overflow on interrupt to inner privilege case
13	X	X	X	X	X	X	X	X		X	X	Doc	FSETPM is like NOP, not like FNOP
14	X	X	X	X	X	X	X	X	X	X	X	Doc	Corrections in the pseudocode descriptions of the instructions CALL, IRET(D) & RET
15	X	X	X	X	X	X	X	X	X	X	X	Doc	Errors in 3 tables of special descriptor types
16	X	X	X	X	X	X	X	X	X	X	X	Doc	Execution Tracing – Fast Messages

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the *Pentium Processor Family Developer's Manual,* Volume 1 and Volume 3, (Order Numbers 241428 and 241430, respectively), or the *Pentium Processor at iCOMP Index 610\75 MHz, 735\90 MHz, 815\100 MHz, 1000\120 MHz, 1110\133 MHz* datasheet, (Order Number 241997). All Specification Changes will be incorporated into future versions of the appropriate document(s).

1. Mixing Steppings in Dual Processing Mode

Some OEMs may choose to ship their systems with one processor, and then perform a field upgrade and add a second processor dual processing system. In some cases, the two processors may not be of the exact same stepping. If there is a need to mix steppings of the Pentium processor in a dual processing system, the following guidelines must be met:

1. The processors must be set to run at the same frequencies, and the same bus/core fractions. For example: If the primary processor is running at 60 and 90 MHz, the dual processor must also run at 60 and 90 MHz.
2. The CPUTYP pin of the dual processor socket must be tied to V_{CC}.
3. Use the following table for restrictions, or workarounds required to mix the steppings. Each of the notes is the errata that may cause the system to fail. There may be other applicable errata, please see the errata descriptions for a full listing and the complete details of the workaround.

Mixing Stepping Matrix				
	B1 as Dual (CM Package)	**B3 as Dual (CM Package)**	**B5 as Dual**	**C2 as Dual**
B1 as Primary	5DP	5DP	5DP	5DP
B3 as Primary	5DP	5DP	5DP	5DP
B5 as Primary	5DP	5DP	1DP	1DP
C2 as Primary	5DP	5DP	1DP	No pipeline restrictions

NOTES:

5DP: Workaround requires pipelining disabled.

1DP: Workaround requires either pipelining disabled, or AHOLD pin held active one clock longer than BOFF# deassertion.

2. MD/VR/VRE Specifications

There are some changes to the standard V_{CC} and timing specifications to support the highest performance operation of the Pentium processor.

STD: The V_{CC} specification for the C2 and subsequent steppings of the Pentium processor is V_{CC} = 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 V_{CC} = 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 V_{CC} 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 specification change 24 in this document and 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

3. Lock Step APIC Operation Specification

This feature is implemented in the C-stepping of the processor.

To support Lock Step operation of two CPUs that implement APIC as the interrupt controller, this specification has been added to guarantee the recognition of the interrupt on a specific clock in both processors. This is related to the FRC operation of the CPU, but FRC on the APIC pins is not fully supported in this way. There is not a comparator on the APIC pins but mismatches on these pins will result in a mismatch on other pins of the CPU.

This is also used for those customers who want to build fault tolerant systems that implement multiple CPUs running identical code sequences and generating identical bus cycles on all clocks. The specification required setup and hold times on PICCLK in relation to the CLK line. There is also a requirement to sustain specific integer ratios in the frequency for these to signals. This ratio (CLK/PICCLK) cannot be less than 4:1, this should support both the maximum frequency of the device and the maximum frequency of the PICCLK. To enter this operation mode, hold the configuration pin BE4# HIGH during the falling edge of RESET. This must meet Setup and Hold times t_{43c}, and t_{43d}, these are the same as the APICEN pin. The BE4# pin has an internal pulldown, so if this pin is pulled down, left as no connect, or floated during this time the default will be normal device operation.

To enter Lock Step operation hold the BE4# (AL13) pin HIGH during the falling edge of RESET, in accordance with timings t_{43c}, and t_{43d}. If a pullup resistor is connected to this pin to cause the part to operate in a dual processing mode, the value of this resistor must be 100 Ohms or less to override the internal pulldown on this pin.

General Conditions: 3.135 < V_{CC} < 3.465V, T_{CASE} = 0 to 70°C, CL = 0 pF

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{43c}	APICEN, BE4# Setup Time	2		CLKs	8	To RESET Falling edge
t_{43d}	APICEN, BE4# Hold Time	2		CLKs	8	To RESET Falling edge
t_{61}	PICCLK Setup Time	5.0		nS	7	To CLK, (31)
t_{62}	PICCLK Hold Time	2.0		nS	7	To CLK, (31)
t_{63}	PICCLK Ratio (CLK/PICCLK)	4				(32)

NOTES:

31 This is for the Lock Step operation of the component only. This guarantees that APIC interrupts will be recognized on specific clocks to support 2 processors running in a Lock Step fashion including FRC mode. FRC on the APIC pins is not supported but mismatches on these pins will result in a mismatch on other pins of the CPU.

32 The CLK to PICCLK ratio has to be an integer and the ratio (CLK/PICCLK) cannot be smaller than 4.

4. 120-MHz and 133-MHz Parts (Q0707, Q0708, Q0711, Q0732, Q0733, Q0751, Q0775, SK086, SX994, SU033, SK098) Do Not Support Dual Processing

The 120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0732, Q0733, Q0751, Q0775, SK086, SX994, SU033, SK098) do not support dual processing as defined in the *Pentium Processor Family Developer's Manual.* Dual processing support will be added in a future stepping.

5. 120-MHz and 133-MHz Parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) Do Not Support FRC

The 120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) do not support FRC as defined in *the Pentium Processor Family Developer's Manual.*

6. 120-MHz and 133-MHz Parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) V_{CC} to CLK Startup Specification

The specification for the maximum time from V_{CC} reaching nominal value to the time the CLK must toggle is 30 ms for 120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098). If this specification is not met it may impact the long term reliability of the component.

V_{CC} to CLK Startup Time: 30 ms maximum.

7. 120-MHz and 133-MHz Parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) Current Leakage on PICD1 Pin

The leakage current specification as described in the *Pentium Processor Family Developer's Manual* is 200µA. The leakage specification for 120-MHz and 133-MHz parts (Q0707, Q0708, Q0711, Q0733, Q0751, Q0775, SK086, SK098) differs as follows:

Symbol	Parameter	Min	Max	Unit	Notes
I_{IH}	Input Leakage Current		250	µA	Vin = 0.4V (2)

Notes: (2) This parameter is for input with pull up resistor.

8. Pentium Processors with Voltage Reduction Technology do not Support the APIC, Dual Processing, Upgrade, or FRC

The following features have been eliminated from the Pentium Processor with Voltage Reduction Technology TCP and SPGA: the upgrade feature, the dual processing (DP) feature, the master/checker functional redundancy feature, and the APIC feature. The following tables list the corresponding signals and pins which exist on the Pentium processor, but have been removed on the Pentium Processor with Voltage Reduction Technology TCP and SPGA. This information supersedes previously published information and is updated in the *Pentium Processor at iCOMP Index 735/90, 610/75 MHz with Voltage Reduction Technology* datasheet (Order Number 242557).

Signals Removed from Pentium® Processor with Voltage Reduction Technology

Signal	Function
ADSC#	Additional Address Status. This signal is mainly used for large or standalone L2 cache memory subsystem support required for high performance desktop or server models.
BRDYC#	Additional Burst Ready. This signal is mainly used for large or standalone L2 cache memory subsystem support required for high performance desktop or server models.
CPUTYP	CPU Type. This signal is used for dual processing systems.
D/P#	Dual/Primary processor identification. This signal is only used for an Upgrade processor.
FRCMC#	Functional Redundancy Checking. This signal is only used for error detection via processor redundancy, and requires two Pentium® processors (master/checker).
PBGNT#	Private Bus Grant. This signal is only used for dual processing systems.
PBREQ#	Private Bus Request. This signal is used only for dual processing systems.
PHIT#	Private Hit. This signal is only used for dual processing systems.
PHITM#	Private Modified Hit. This signal is only used for dual processing systems.
PICCLK	APIC Clock. This signal is the APIC interrupt controller serial data bus clock.
PICD0 [DPEN#]	APIC's Programmable Interrupt Controller Data line 0. PICD0 shares a pin with DPEN#.
PICD1 [APICEN]	APIC's Programmable Interrupt Controller Data line 1. PICD1 shares a pin with APICEN.

TCP Pin Changes

Pin Number	Pentium® Processor with Voltage Reduction Technology TCP	Pentium Processor 75-MHz TCP
155	NC	PICCLK
156	NC	PICD0
158	NC	PICD1[APICEN]

SPGA Pin Changes

Pin Number	Pentium® Processor with Voltage Reduction Technology SPGA	Pentium® Processor
AM02	NC	ADSC#
Y03	NC	BRDYC#
Q35	NC	CPUTYP
AE35	NC	D/P#
AD04	NC	PBGNT#
AE03	NC	PBREQ#
AA03	NC	PHIT#
AC03	NC	PHITM#
H34	NC	PICCLK
J33	NC	PICD0[DPEN#]
L35	NC	PICD1[APICEN]
Y35	NC	FRCMC#

9. SPGA T_{CASE} Specification is Improved to 85 C

The maximum operating temperature, T_{CASE}, of the Pentium processor with Voltage Reduction Technology SPGA is improved from 70°C to 85°C.

10. Maximum Power Specification for mA1 and mA4 Parts

Incorporated into the latest revision of the *Pentium Processor at iCOMP Index 815/100 MHz, Pentium Processor at iCOMP Index 735/90 MHz, Pentium Processor at iCOMP Index 610/75 MHz with Voltage Reduction Technology* datasheet (Order Number 242973).

11. V_{CC} Measurement Specification

The values of V_{CC} should be measured at the bottom side of the CPU pins using an oscilloscope with a 3 dB bandwidth of at least 20-MHz (100 MS/s digital sampling rate). There should be a short isolation ground lead attached to a CPU pin on the bottom side of the board. The measurement should be taken at the following V_{CC} /V_{SS} pairs: AN13/AM10, AN21/AM18, AN29/AM26, AC37/Z36, U37/R36, L37/H36, A25/B28, A17/B20, A7/B10, G1/K2, S1/V2, AC1/Z2. The display should show continuous sampling of the V_{CC} line, at 20 mV/div, and 500

nS/div with the trigger point set to the center point of the range, then slowly move the trigger to the high and low ends of the specification. There are no allowances for crossing the high and low limits of the voltage specification. For more information on measurement techniques, see application note *Implementation Guidelines for 3.3V* Pentium® *Processors with VR/VRE Specifications.*

12. STI Followed by FP instruction Doesn't Delay Interrupt Window

In the *Pentium Processor Family Developer's Manual*, Volume 3, page 25-301, the paragraph under "Description" should read as follows:

The STI instruction sets the IF. **STI enables interrupts after the next instruction executed following STI unless the next instruction clears the IF**. **Assuming interrupts are not already enabled, STI guarantees that interrupts will not be enabled until after the next instruction unless that next instruction is an FP instruction.** If external interrupts are disabled and the STI instruction is followed by the RET instruction (such as at the end of a subroutine), the RET instruction is allowed to execute before external interrupts are recognized. Also, if external interrupts are disabled and the STI instruction is followed by a CLI instruction which clears the IF flag, then external interrupts are not recognized because the CLI instruction clears the IF flag during its execution.

13. 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.

14. DP Not Supported in FRC Systems

Dual processing in a FRC system is not supported. This should have been stated explicitly in the *Pentium Processor Family Developer's Manual*, Volume 1, beginning of Section 18, page 18-2.

15. Decreased Maximum Power Specification for mcB1 Parts

Incorporated into the latest revision of the *Pentium Processor at iCOMP Index 1110\133, 1000\120, 815\100 MHz with Voltage Reduction Technology* datasheet (Order Number 242557).

16. PICD0-1 Valid Delay (Resistor Pull-up & V_{OL})

The specified range of values for the external pullup resistor to V_{CC} has changed from 150 Ohms – 1K Ohms to 300 Ohms – 1K Ohms with V_{OL} = 0.55V. Note 28 on page 23-24 of the *Pentium Processor Family Developer's Manual*, Volume 1 has been modified as follows:

28. This assumes an external pullup resistor to V_{CC} and a lumped capacitive load. The pullup resistor must be between 300 Ohms and 1K Ohms, the capacitance must be between 20 pF and 240 pF, and the RC product must be between 3 ns and 36 ns. V_{OL} for PICD0-1 is 0.55V.

Typical V_{OL} for a 150 Ohm pullup is 0.5V. Intel knows of no system that has exhibited a problem with a 150 Ohm pullup.

The Valid Delay Timing specifications for PICD0-1 remain unchanged.

17. PHITM# Timing Change

The flight time of the PHITM# signal has been reduced from 2.0 nS to 1.8 nS for the cC0 and subsequent steppings.

Symbol	Signal	Previous Flight Time (50/60/66 MHz)	Current Flight Time (50/60/66 MHz)
t_{80}	PHITM#	2.0 nS	1.8 nS

18. Min Delay Timings

Effective with the cC0 stepping, standard (STD) timings for all speeds of SPGA-packaged parts will be replaced by the following. These correspond to the existing Min Valid Delay (MD) timings, with the pins ADS#, ADSC# and SCYC also now having reduced minimum valid delays.

Symbol	Signal	Min Valid Old STD (50/60/66 MHz)	Min Valid New STD (50/60/66 MHz)
t_{6a}	W/R#	1.0 nS	0.8 nS
t_{6a},t_{6f}	M/IO#	1.0 nS	0.8 nS
t_{6a}	D/C#	1.0 nS	0.8 nS
t_{6a}	BE0-7#	1.0 nS	0.9 nS
t_{6e}	A3-A16	1.1 nS	0.5 nS
t_{6e}	A17-A31	1.1 nS	0.6 nS
t_{6c}	LOCK#	1.1 nS	0.9 nS
t_{10b}	HITM#	1.1 nS	0.7 nS
t_{6a},t_{6d}	ADS#	1.0 nS	0.8 nS
t_{6a}	ADSC#	1.0 nS	0.8 nS
t_{6a}	SCYC	1.0 nS	0.8 nS

19. Requirements for STPCLK# Re-assertion Recognition

In the *Pentium Processor Family Developer's Manual*, Volume 1, Section 30.5.2.2 the following sentences explain the requirements for recognition by the processor of STPCLK# assertion, de-assertion and subsequent re-assertion: "If STPCLK# does not remain active until the Stop Grant bus cycle has been issued, it may not be recognized. In the event that the first assertion of STPCLK# is recognized, a subsequent re-assertion of STPCLK# before the completion of the first Stop Grant bus cycle may not generate a second Stop Grant bus cycle." These requirements are insufficient to guarantee recognition of each STPCLK# assertion under all circumstances, and will be replaced with the following:

To guarantee that every STPCLK# assertion, and subsequent de-assertion and re-assertion, is recognized and thus will get a Stop Grant bus cycle response (which will also ensure that each de-assertion of STPCLK# allows execution of at least one instruction), the system must meet the following requirements:

1. Hold STPCLK# active at least until the processor's Stop Grant cycle response has been completed by the system's BRDY# response.
2. STPCLK# must not be re-asserted until 5 clocks after the **last** of the following events:
   a. The processor's Stop Grant cycle has been completed by the system's BRDY# response.
   b. HITM# is de-asserted. (This applies only if HITM# was asserted while waiting for one of the other two events listed here, **or** within 2 bus clocks of their completion.)
   c. EWBE# becomes active after it was sampled inactive at the last relevant BRDY#. A relevant BRDY# is one which ends **either** a stop-grant cycle **or** an external snoop writeback caused by HITM# being asserted as in case b) above.

Events b) and c) can in principle alternate indefinitely, continuing to delay STPCLK# de-assertion recognition, if the system design allows that to happen.

Note that if a system is not relying on either a Stop Grant bus cycle response for every STPCLK# assertion, or for each de-assertion of STPCLK# to allow execution of at least one instruction, these detailed requirements can be ignored.

20. Power Specifications for 150, 166 and 200 MHz Parts

The power specifications for the Pentium® Processor cC0 Step Parts at 150, 166 and 200 MHz are:

Part	Bus Freq.	Core Freq.	Power Max.[1] (W)	Power Typical[2] (W)	I_{CC} Max.[3] (mA)	Stop Grant AutoHalt Max	StopClk (W)
cC0	66	200	15.5	6.5	4600	2.5	<0.1
cC0	66	166	14.5	5.4	4250	2.1	<0.1
cC0	60	150	11.6	4.9	3850	1.9	<0.1

Notes:

1. Maximum active power dissipation is measured using the worst case instruction mix, with VRE V_{CC} = 3.5 V for 166 and 200 MHz parts, and STD V_{CC} = 3.3 V for 150 MHz parts.
2. Typical power is estimated for typical systems running typical applications, with VRE V_{CC} = 3.5 V for 166 and 200 MHz parts, and STD V_{CC} = 3.3 V for 150 MHz parts.
3. Maximum I_{CC} is measured under worst case conditions: V_{CC} = 3.6V and T_{CASE} = 70°C

This information will be added to the *Pentium Processor Family Developer's Manual*, Volume 1, in tables 23-2 and 23-5.

21. Boundary Scan Min Valid-Delay Timings (t_{53} & t_{55})

Boundary scan output min delays (specifications t_{53} and t_{55}) have been reduced from 3.0 nS to 2.5 nS for cC0 processors and future steppings.

22. Leakage Current I_{LI}/I_{LO} Applies Only in the Valid Logic States

The leakage currents I_{LI} and I_{LO} only apply in the valid logic states, i.e. $V_{IN}<V_{IL}$ and $V_{IN}>V_{IH}$, instead of $0<V_{IN}<V_{IH}$, which is the current specification. The *Pentium Processor Family Developer's Manual*, Volume 1, Tables 7-2 and 23-4 will be updated accordingly.

23. 150 MHz Active Power Dissipation (typical) Change

These new specifications for Pentium Processors @150 MHz active power dissipation (typical) apply ONLY to the mcC0 stepping (mobile specific) TCP parts. The previous, preliminary range giving typical power (3.8 to 5.0 W) for VccCore of 2.9V has been reduced to the range (3.4 to 4.5 W). Parts with a VccCore of 3.1V are specified to have a typical power range (3.8 to 5.0 W).

Parameter	Previous Value (Watts)	Current Value (Watts)
150 MHz Active Power	N/A	3.8 - 5.0 W @ 3.1V
(typical) Dissipation	3.8 - 5.0 W @2.9V	3.4 - 4.5 W @ 2.9V

S-SPECS

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

Symbol	Parameter	Datasheet Max (nS)	S-Spec Max (nS)	Notes
t_{6a}	ADS#, BE0-7#	7.0	7.25	50 MHz, 60 MHz bus
t_{6c}	A31-A3, LOCK#	7.0	7.25	50 MHz, 60 MHz bus

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

In order to ensure proper operation a minimum of 120mV separation must be maintained between V_{CC3} (min) and V_{CC2} (max) at all times while the processor is powered up. In order to provide flexibility in the power supply design, the voltage tolerance of +/- 165mV will be supported on both supplies, however, the voltage difference between the two supplies must remain greater than 120mV. Therefore, the capacitive decoupling scheme that handles current transients must be chosen to support this requirement. It is recommended that the OEM provide sufficient decoupling capacitance for the desired voltage tolerance distribution between V_{CC3} and V_{CC2}, and perform actual system measurements to validate the design.

For further information, contact your local Intel Sales office.

V_{CC3} (Min)	V_{CC2} (Max)	V_{CC3} (Min) - V_{CC2} (Max) Standard	V_{CC3} (Min) - V_{CC2} (Max) S-Spec
3.3 - 0.165V	2.9 + 0.165V	70mV	120mV*

* Test condition is at $V_{CC}2$ = 2.9V + 0.165V = 3.065V: $V_{CC}3$ = 3.3V - 0.115V = 3.185V

3. V_{IH} For TRST#

Symbol	Pin	Standard Min	S-Spec Min	Unit
V_{IH}	TRST#	2.0	2.2	Volts

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

Symbol	Pin	Standard Max	S-spec Max	Unit
V_{IL}	BF, BF1	0.8	0.6	Volts

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 V_{CC2} Supply Voltage Change

The core supply voltage (V_{CC2}) required is changed from 2.9V to 3.1V. This applies to SPGA 100/120/133 MHz units only. I/O voltage supply (V_{CC3}) 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

The TCP and SPGA Pentium Processor with Voltage Reduction Technology AC specifications for 60- and 66-MHz bus operation have been published in the *Pentium® Processor with Voltage Reduction Technology* datasheets (Order Numbers 242973 and 242557). The mA1 and mA4 steppings differ from the published specifications as noted below.

SPGA AC Specifications for 60-MHz Bus

Symbol	Parameter	S-Specs 60MHz Bus		Mobile Standard 60MHz Bus	
		Min	Max	Min	Max
t_{6e}	A3-A31 Valid Delay	1.1	7.7	1.1	6.3
t_{12}	D0–D63, DP0–7 Write Data Valid Delay	1.3	7.8	1.3	7.5
t_{35}	D0–D63, DP0–7 Read Data Hold Time	2.0		1.5	

TCP AC Specifications for 60-MHz Bus

Symbol	Parameter	S-Specs 60MHz Bus		Mobile Standard 60MHz Bus	
		Min	Max	Min	Max
t_{6e}	A3-A31 Valid Delay	1.1	7.0	1.1	6.3
t_{35}	D0–D63, DP0–7 Read Data Hold Time	2.0		1.5	

SPGA AC Specifications for 66-MHz Bus

Symbol	Parameter	S-Specs 66MHz Bus		Mobile Standard 66MHz Bus	
		Min	Max	Min	Max
t_{6a}	BE0-7# Valid Delay	1.0	7.25	1.0	7.0
t_{6c}	LOCK# Valid Delay	1.1	7.25	1.1	7.0
t_{6d}	ADS# Valid Delay	1.0	7.0	1.0	6.0
t_{6e}	A3-A31 Valid Delay	1.1	7.5	1.1	6.3
t_{6f}	M/IO# Valid Delay	1.0	7.0	1.0	5.9
t_{9c}	HLDA Valid Delay	1.0	7.2	1.0	6.8
t_{10a}	HIT# Valid Delay	1.0	8.0	1.0	6.8
t_{12}	D0–D63, DP0–7 Write Data Valid Delay	1.3	7.8	1.3	7.5
t_{16b}	EADS# Setup Time	5.5		5.0	
t_{24b}	PEN# Setup Time	5.0		4.8	
t_{34}	D0–D63, DP0–7 Read Data Setup Time	3.0		2.8	
t_{35}	D0–D63, DP0–7 Read Data Hold Time	2.0		1.5	

TCP AC Specifications for 66-MHz Bus

Symbol	Parameter	S-Specs 66MHz Bus		Mobile Standard 66MHz Bus	
		Min	Max	Min	Max
t_{6c}	LOCK# Valid Delay	1.1	7.25	1.1	7.0
t_{6d}	ADS# Valid Delay	1.0	7.0	1.0	6.0
t_{6e}	A3-A31 Valid Delay	1.1	7.0	1.1	6.3
t_{6f}	M/IO# Valid Delay	1.0	6.8	1.0	5.9
t_{10a}	HIT# Valid Delay	1.0	8.0	1.0	6.8
t_{16b}	EADS# Setup Time	5.5		5.0	
t_{24b}	PEN# Setup Time	5.0		4.8	
t_{34}	D0–D63, DP0–7 Read Data Setup Time	3.0		2.8	
t_{35}	D0–D63, DP0–7 Read Data Hold Time	2.0		1.5	

8. Reduced V_{IL} For TCK

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

ERRATA

1. Branch Trace Message During Lock Cycles

PROBLEM: During instruction execution tracing only two Branch Trace messages can be buffered. If there is a possibility of a third message being delivered from the instruction being executed, the machine will stall to avoid overwriting either of the messages that are buffered. If this instruction is a "locked read-modify-write" operation, the processor will hang up due to internal service contention for the bus controller logic.

IMPLICATION: This problem only effects operation of the component while performing instruction tracing on the CPU. It has not been seen using Intel development tools.

WORKAROUND: None identified at this time.

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

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

PROBLEM: If the following conditions are met, the processor may shut off the interrupt window for one instruction following STI:

1. The address of the instruction preceding the STI instruction is a hit in the BTB.
2. The target address of the BTB hit does not correspond to the instruction following the STI instruction.

This will prevent breakpoints, single-step or other external interrupts from being recognized during this time.

IMPLICATION: The processor may not recognize NMI, SMI# INIT, FLUSH#, BUSCHK#, R/S#, code/data breakpoint and single-step for one instruction after executing STI. This is not a problem unless breakpoints or single-stepping is used and then the only effect is that the breakpoint would be missed.

WORKAROUND: Do not set a breakpoint on the next sequential instruction after STI, or disable branch prediction to prevent this problem.

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

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

PROBLEM: If an SMI# interrupt is generated while a data breakpoint exception is pending or during single-stepping, an I/O restart attempt will not be successful.

IMPLICATION: If this problem occurs, it will not be possible to restart the I/O instruction.

WORKAROUND: Do not allow single-stepping or data breakpoint exceptions when attempting to restart an I/O instruction.

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

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

PROBLEM: An NMI# or INIT may be falsely accepted while in an SMM handler if *all* of the following conditions are met:

1. I/O restart is enabled with the ITR bit in TR12.
2. An SS or DBP exception is pending at the time that the SMI# is recognized.
3. NMI# or INIT is asserted before another fault occurs or before an INTR occurs (and the IF flag is set).

IMPLICATION: If the above conditions are met, the processor may falsely go into an interrupt service routine or begin the INIT sequence.

WORKAROUND: Do not enable I/O trap restart while single-stepping.

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

5. SMI# and FLUSH# During Shutdown

PROBLEM: If the processor transitions from the shutdown state to System Management mode via an SMI# interrupt, and FLUSH# is asserted after the SMI# interrupt is recognized, the processor returns to the shutdown state rather than to the SMM handler.

IMPLICATION: After the FLUSH# is asserted, the processor erroneously returns to the shutdown state when it should return to the SMM handler.

WORKAROUND: Do not assert SMI# while the processor is in the shutdown state.

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

6. No Shutdown After IERR#

PROBLEM: If an internal parity error is reported to the IERR# pin and a mispredicted branch or a trap with higher priority than shutdown occurs, then the processor may not shutdown.

IMPLICATION: During the reporting of an internal parity error, the IERR# pin may go active without a processor shutdown.

WORKAROUND: When the IERR# pin is asserted, force the system to shutdown.

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

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

PROBLEM: If I/O restart is enabled during single-stepping or while a breakpoint is pending, and a FLUSH# is asserted, the wrong value will be stored in DR6.

IMPLICATION: The debug status register (DR6) may contain false information.

WORKAROUND: Do not assert FLUSH# when I/O restart is enabled with single-stepping or a breakpoint is pending.

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

8. Processor Core May Not Serialize on Bus Idle

PROBLEM: Under rare circumstances, a BOFF# asserted to the processor, while it is internally waiting (looping) for the completion of all pending bus cycles, may cause the processor to proceed with the event before all pending bus activity is completed. However, bus cycle ordering will not change.

The following is a list of events identified that may possibly effect system operation:

- SMI# pending
- OUT instructions
- Serializing instructions
- Stop Grant special cycle
- AutoHALT special cycle
- INIT asserted while there is a memory mapped APIC register access

SMI# pending	If BOFF# is used to back off a bus cycle while an SMI# is pending, the processor may assert SMIACT# before re-starting the aborted bus cycles.
Serializing instruction	If BOFF# is used to back off a bus cycle due to a serializing instruction, the processor may start executing the next instruction before restarting or completing the previous bus cycle. The processor, however, will not reorder any bus cycles for the new instruction in front of bus cycles for the previous instruction.
Invalidation during cache line fill	If BOFF# is used to back off a cache line fill and BOFF# occurs after the data has been returned to the processor but before the end of the line fill, an invalidation request during this time may result in the cache invalidation to occur before the line fill has completed, This may cause the cache line to remain in a valid state after the invalidation has completed. Note that if the invalidation request comes in via WBINVD or FLUSH#, the line fill would have to be backed off at least twice (or once for INVD) in order for the cache line to remain in a valid state after the invalidation has completed.
OUT instruction	If BOFF# is used to back off a bus cycle due to an OUT instruction, the processor may start executing the next instruction before the bus cycle due to OUT has completed. (NOTE: the OUT instruction is similar to the serializing instructions except that it does not stop the prefetch of the subsequent instruction.) The processor, however, will not reorder any bus cycles for the new instruction in front of the OUT bus cycle.
Stop Grant special cycle	If BOFF# is used to back off a Stop Grant special cycle, the processor may hang.
AutoHALT special cycle	If BOFF# is used to back off a AutoHALT special cycle, the processor may hang.
INIT asserted while there is a memory mapped APIC register access	If BOFF# is used to back off a memory mapped APIC register access while an INIT is pending, the processor may hang. The access could be either a read or a write, and is an access to the local APIC register space.

IMPLICATION: This problem has only been observed in internal test vehicles. The six events have different possible implications.

SMI# pending	The processor may enter SMM before restarting the aborted bus cycle. The SMIACT# assertion may cause the restarted bus cycle to run to SMRAM space.

Serializing Instruction	Since the cycles are not reordered, a system should not encounter any problems unless it depends on the serializing instruction causing an external event prior to execution of the next instruction.
Invalidation during cache line fill	In a rare instance, a cache line may remain in the valid (E or S) state after the cache invalidation has completed.
OUT instruction	Since the cycles are not reordered, a system should not encounter any problems unless it depends on the OUT instruction causing an external event prior to execution of the next instruction. For example, an OUT instruction may be used to assert the A20M# signal prior to the next instruction. In this case, observed code has followed the OUT with an I/O read (IN) to ensure the signal is properly asserted. A second case, could be using an OUT instruction to configure/initialize and interrupt controller and follow it with STI to enable interrupts. Once again no failure would be observed. The controller would respond with the spurious interrupt vector.
Stop Grant special cycle	If BOFF# is used to back off a Stop Grant special cycle, the processor may hang. Stop Grant and Stop Clock states for low power consumption cannot be used.
AutoHALT special cycle	If BOFF# is used to back off a AutoHALT special cycle, the processor may hang. This means that the lower power AutoHALT state is not usable. This does not affect the normal HALT state, entered with the HLT instruction though.
INIT asserted while there is a memory mapped APIC register access	If BOFF# is used to back off a memory mapped APIC register access (read or write) while an INIT is pending, the processor may hang. The INIT would only be used during a switch from Protected to Real mode, which is normally associated with a system reboot. In the processor lockup occurs a reboot may have to be performed via system powerdown.

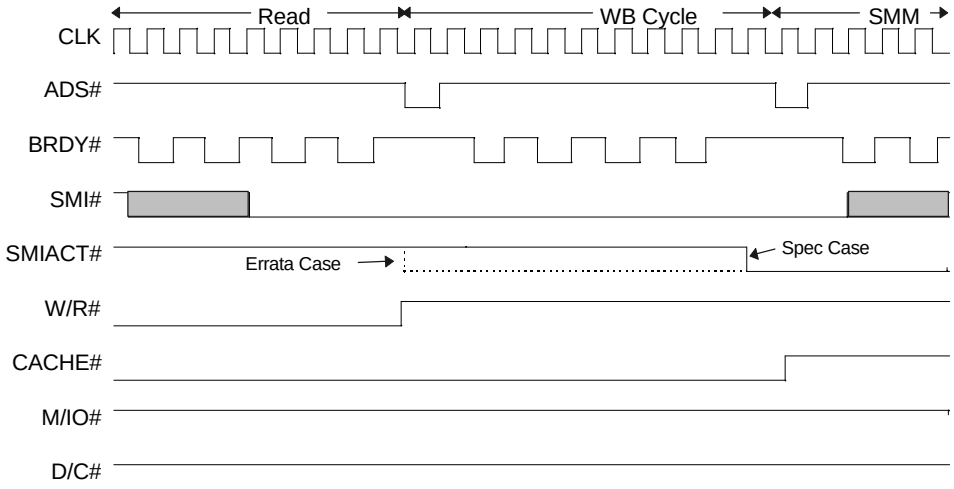
WORKAROUND: Restrict the use of BOFF# for the described events. In addition, the SMI# pending event can be eliminated by locating SMRAM so that it does not shadow standard memory and does not require SMIACT# for memory decode. The OUT or Serializing instruction events are also eliminated if the next instruction does not depend on the result of the event before executing the instruction. The Stop Grant special cycle event is also eliminated by not asserting STPCLK#. The AutoHALT special cycle event is also eliminated by disabling AUTOHALT (set TR12.AHD bit to '1').

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

9. SMIACT# Premature Assertion During Replacement Writeback Cycle

PROBLEM: If a data read cycle triggers a replacement writeback cycle and the SMI# signal is asserted before the last BRDY# of the read cycle, the processor may assert the SMIACT# signal prematurely. It occurs on all steppings when the processor is in 2/3 bus mode. **If the processor is in 1/2 bus mode it will not exhibit this erratum for C2 and subsequent steppings.**

Before the processor asserts SMIACT# in response to an SMI# request, it should complete all pending write cycles (including emptying the write buffers). However, if the appropriate conditions occur, the SMIACT# signal may get asserted during the replacement writeback cycle, at anytime during the writeback cycle. See below diagram.



IMPLICATION: If the SMIACT# signal is used by the system logic to decode SMRAM (e.g., SMRAM is located in an area that is overlaid on top of normal memory space, e.g. system, video etc.), then the replacement writeback cycle with SMIACT# asserted could occur to SMM space. Systems that locate SMRAM in its own distinct memory space (non-overlaid) should not experience data corruption once the SMRAM has been initialized and relocated.

Some board designs and/or chip sets may contain logic which locks when an SMIACT# is asserted during the writeback cycle. This is logic dependent and not all systems will fail when this condition occurs, although data corruption could still result.

WORKAROUND: Use one of following:

Non Overlaid SMRAM

In systems that relocate the SMBASE so that it does not overlay normal memory space.

1. Do not use SMIACT# as a decode signal once SMRAM has been relocated. When entering SMM for the first time to relocate the SMBASE to the non-overlaid region the system must be in one of three modes: the processor L1 cache must be in WT mode, the default SMBASE location (30000H) should be marked non-cacheable, or the L1 cache should be turned off. Once the SMBASE has been relocated to the non-overlaid region and SMIACT# is no longer used to decode SMRAM, the processor's L1 cache may be used in WB mode or the memory area at 30000H may be configured as a cacheable region.

Overlaid SMRAM

In systems that locate SMRAM over normal cacheable or non-cacheable memory space there are several software and hardware workarounds.

1. Operate the processor with the L1 cache in WT mode only. This will eliminate all processor WB cycles and thus prevent the error condition from occurring. The performance impact to the system for this could be between 5-15% depending on L2 cache size, speed, and operating mode (WB/WT).

2. Assert FLUSH# one clock before SMI# or assert FLUSH# and SMI# simultaneously and synchronously with the processor clock. The processor always assigns FLUSH# a higher priority than SMI# and thus it will delay the assertion of SMIACT#. This is already a necessary requirement to maintain cache coherency when SMRAM overlays a cacheable normal memory area. Depending on the chip set, external hardware may be required to synchronize SMI# and FLUSH#. The performance impact of this solution depends heavily on frequency of SMIs. No performance impact should be visible if SMM is accessed infrequently, such as during periods of bus inactivity. Frequent SMM access (i.e. 18/sec) will result in less than 1% processor performance decrease.

Note that other hardware workarounds may be possible. Any other hardware workaround needs to take into account the possible case when BOFF# or AHOLD is asserted thereby restarting the writeback cycle after SMIACT# may have been asserted.

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

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

This erratum has been replaced by Specification Change 19.

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

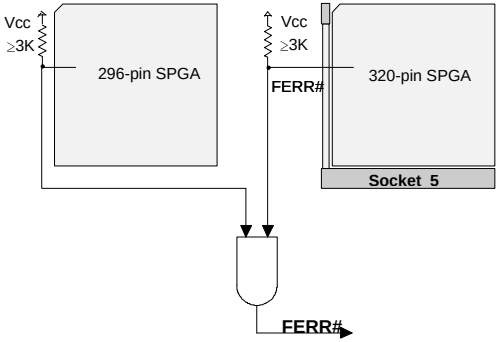
PROBLEM: When the future Pentium OverDrive® processor is plugged into Socket 5 of a two socket 75-, 90-, or 100-MHz Pentium processor system, the OEM processor shuts down following RESET to allow the OverDrive processor to drive the bus. In this case, the FERR# output of the 75-, 90-, and 100-MHz Pentium processor continues to be driven HIGH (inactive).

IMPLICATION: If the system uses the FERR# output of the OEM processor, and has the signal connected together between the two sockets (296-pin SPGA OEM socket and 320-pin Socket 5), contention on this signal is certain since the future Pentium OverDrive processor, when placed into Socket 5, will also drive this output. This signal contention can cause component and even board reliability issues.

WORKAROUND: There are two possible workarounds for this erratum.

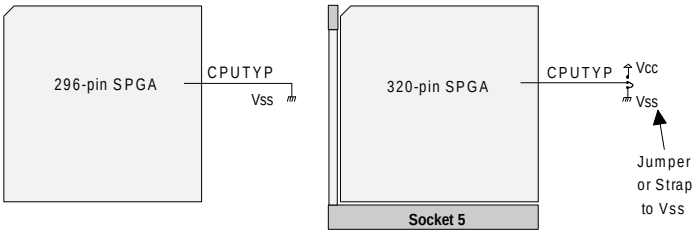
1. **For upgradability in two socket systems:** External logic may be used to connect the FERR# outputs from the two sockets (296-pin SPGA and 320-pin Socket 5) in a way that will resolve the signal contention.

   An external AND gate may be used to combine the outputs of the FERR# signals from the two sockets. A pullup resistor (≥3KΩ) is required on the FERR# output from both of the sockets in order to allow proper operation of both the dual processor and the future Pentium OverDrive processor. See the following figure:



2. **Upgradability by modifying a two socket system to be a single socket system:** This workaround involves modifying a design that includes two socket sites (296-pin SPGA and 320-pin Socket 5) such that it effectively becomes a single socket design.

   A dual processing two socket system must have CPUTYP tied to V_{SS} on the 296-pin SPGA OEM socket, and tied to V_{CC} on Socket 5 (320-pin SPGA). This workaround includes inserting a jumper on the Socket 5 CPUTYP signal (or strapping Socket 5 CPUTYP directly to V_{SS}) to make this the primary processor site. The system would then effectively become a single-socket design. NOTE: If a jumper is used, it must be set by the OEM prior to system sale (not by the end-user at the time of the future Pentium OverDrive processor purchase and installation). This jumper would set the CPUTYP signal on Socket 5 to V_{SS}. If the same board design is used for dual processing, this jumper (or strapping option) may be set to V_{CC} for those systems, as shown in the following figure:

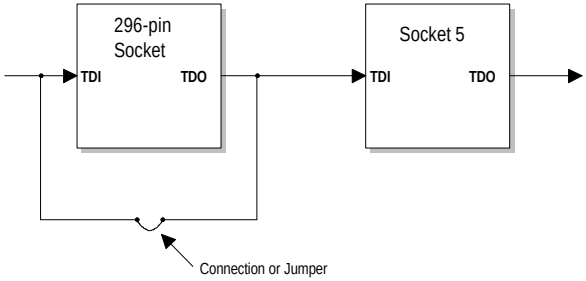


By setting CPUTYP to V_{SS}, and inserting the 75-, 90-, and 100-MHz Pentium processor into Socket 5 prior to system sale, the design can be treated as if it is a single socket system. When upgrading with the future Pentium OverDrive processor, the 75-, 90-, and 100-MHz Pentium processor is removed from Socket 5 and replaced by the OverDrive processor upgrade.

IMPLICATIONS OF WORKAROUND #2:

Other implications of this workaround include Boundary Scan, and any other signals not connected together between Socket 5 and the 296-pin SPGA socket site.

If Socket 5 follows the 296-pin socket in the Boundary Scan chain, the TDI input and TDO output of the 296-pin socket site must be connected together by the OEM prior to system sale in order to skip this socket site and complete the path to socket, as shown in the following figure. This connection is necessary only if Boundary Scan will be used by end-users after system sale.



All of the signals which are not connected together in a dual socket system must be handled by both socket sites if the feature is desired. These signals are APCHK#, BP[3:0], IERR#, PM[1:0], PRDY, and R/S#.

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

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

PROBLEM: If the code cache is snooped while the processor is in the Stop Grant state or the AutoHALT Powerdown state, and there is a hit to a line in the code cache with the INV pin asserted, the line may not be invalidated. In normal operation, a hit to a line in the code cache results in that line being invalidated.

IMPLICATION: This problem will cause the snooped line to remain valid in the cache. This may cause the processor to execute an invalid instruction that erroneously remained valid in the code cache. NOTE: HIT# is properly asserted. This may occur in DMA transfers, or transfers to hard disks. It was found on a disk drive that used a BIOS that used HALTs extensively in the boot sequence and performed data transfers after the CPU entered the AutoHALT state. Since this occurs in both the AutoHALT state and the Stop Grant states of the SL Enhanced power management features, both of these features should not be used unless one of the listed workarounds is implemented. Not using the power management features could impact the compliance of an Energy Star Compliant System.

WORKAROUND: Use one of the following:

1. Disable the AutoHALT Powerdown feature by setting the TR12.AHD bit (bit 6) to '1' and do not assert STPCLK#.
2. Flush the caches before or upon entry into the AutoHALT or Stop Grant states. The flush will be serviced immediately if in the AutoHALT state, while the flush will be serviced after the Stop Grant state is exited.
3. Wake up the processor via an NMI or an R/S# prior to snooping the code cache three clocks before the EADS# of the snoop. If the system generates either NMI or R/S# pins based on AHOLD the processor will come out of the out of the low power state to service the Snoop. (This workaround assumes that the minimum 2 clock space between AHOLD and EADS# is not being used.)
4. Use the HIT# indication from the processor to flush the cache if the processor is in the AutoHALT Powerdown or Stop Grant state. The 75-, 90-, and 100-MHz Pentium processors asserts the HIT# pin when a snoop has caused a hit in the code cache. With this Workaround, it is necessary for the system to externally track the status of the cache line in the processor. (i.e., if the processor is in AutoHALT Powerdown or Stop Grant state and the INV pin was active during the snoop, then if the HIT# is returned active, assert FLUSH#.)

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

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

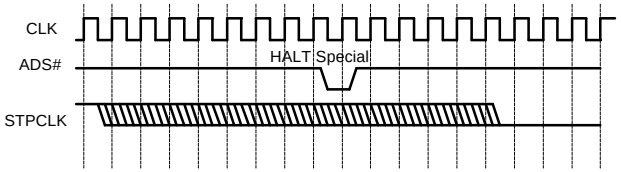
PROBLEM: If STPCLK# is asserted during the execution of the HALT instruction the processor will enter the Stop Grant mode without driving a Stop Grant bus cycle on to the bus. There is a 14 clock window around the ADS# for the HALT special cycle that this erratum occurs. (see figure) This erratum happens even when the AutoHALT power down feature is disabled. There are 3 scenarios for the symptom of this problem, depending on the way the system gets out of HALT.

1. When using INTR, the Interrupt Acknowledge Cycle appears on the bus, but then no other cycles. (The device has entered the Stop Grant state without issuing a Stop Grant special cycle.)
2. When using NMI or INIT, no bus cycles appear on the bus. (The device has hung up, the core has started a bus cycle but the clocks to the core have been stopped)
3. When using SMI#, the SMIACT# is driven asserted, and the SMM state dump completes, and no other cycles appear on the bus.

In addition when there is an event that brings the CPU out of HALT temporarily like FLUSH# or R/S#, if STPCLK# is asserted as the processor re-enters the HALT state, the erratum will occur.

At this time the processor has entered the Stop Grant mode but the part should have generated a Stop Grant special cycle prior to entry. If STPCLK# is deasserted for at least 1 clock, prior to the interrupt assertion the processor will resume correct operation.

The following figure depicts the minimum window around the HALT special cycle ADS#:



IMPLICATION: If the processor enters the Stop Grant state without issuing a Stop Grant special cycle, the state tracking machines of a chip set will be corrupted. A chip set will typically wait for the Stop Grant cycle before deasserting the STPCLK# pin. This will cause a system to hang.

WORKAROUND: Use one of the following:

- Do not use STPCLK# and disable the STPCLK# feature.
- If there is a way to deassert STPCLK# based on a timer tick, or the decode of the HALT special cycle prior to the interrupt then the system will not hang.
- If STPCLK# is being generated via software control such as an I/O instruction, then correct STPCLK# assertion can be guaranteed by using a code loop or string of no-ops that are equal to the latency of the STPCLK# assertion. As long as this code does not contain the HALT instruction, there is no possibility of this erratum occurring.

For example:

```
      MOV   CX, STPCLK_Delay  ;set the delay to the falling edge of STPCLK#.
      OUT   Stop_Clock_port, AX     ;trigger STPCLK#
L1:   NOP                           ;The loop delay should be at least equal to
                                    ;the hardware delay in asserting the STPCLK#
                                    ;signal.
      LOOP  L1
```

```
NOP                                ;Ten NOP instructions must follow the
   .                               ;assertion of STPCLK#.
   .
   .
NOP
```

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

14. NMI or INIT During HALT Within an 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 or INIT while in SMM, except after an IRET instruction.

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 the use 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 at the beginning of this section.

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 the specification clarification near the end of this document.

IMPLICATION: The Runbist operation is started without implementing one of the listed workarounds. 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 Specifications Clarification 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.

or

2. If RESET is held HIGH during the execution of the RUNBIST Boundary Scan instruction and the subsequent 2^{19} clocks.

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

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 at the beginning of this section.

17. STPCLK# Restrictions During EWBE#

This erratum has been replaced by Specification Change 19.

18. Multiple Allocations Into Branch Target Buffer

PROBLEM: A specific sequence of code may cause the Pentium processor to erroneously allocate the same branch into multiple ways of the Branch Target Buffer. These multiple entries may contain conflicting branch predictions. If this occurs and the branch address is accessed for a branch prediction, an incorrect entry may be written into the instruction cache, resulting in the possible execution of invalid or erroneous opcodes and probable activation of the IERR# signal. The incorrect write is dependent on process and circuit sensitivities which vary from one unit to the next.

The occurrence is extremely rare and is software dependent. A specific sequence of code is required to create the condition. In addition, Branch Target Buffer taken/not taken predictions associated with this code must proceed in a specific pattern.

Sensitive code can be summarized as follows: Any conditional jump (possibly paired with a previous instruction), sequentially followed by a move to a segment register, with any jump or instruction pair containing any jump at the target address (LBL_A below) of the first jump. An example follows:

```
                JCXZ LBL_A
                MOV ES, CX
                     .
                     .
                     .
LBL_A:          CALL LBL_B
```

For this erratum to occur, there must also be a specific pattern of taken/not-taken in the conditional jump (JCXZ), as well as a specific pattern of hit/miss in the segment descriptor cache for the segment register load.

IMPLICATION: When this erratum occurs, the processor may execute invalid or erroneous instructions and may assert IERR#. Depending on software and system configuration, the user may see an application error message or a system reset.

WORKAROUND: Several workarounds are available:

1. Disable the Branch Target Buffer by setting the NBP bit (0) to '1' in TR12. This results in approximately 7 percent degradation in performance on the BAPCo benchmark suite, a measure of typical system performance.
2. Use a software patch to avoid the sensitive sequence of instructions. The specific code sequence has only been observed in Windows* 3.10 and 3.11.
3. Ensure that the descriptor tables reside in non-cacheable areas of memory.

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

19. 100-MHz REP MOVS Speed Path

PROBLEM: A speed path exists in the Pentium processor that may cause failures at the rated operating frequency of the part. Under certain rare and specific conditions, the speed path can cause the REP MOVS instruction to be misprocessed.

For this speed path to be exercised, the following conditions must be met:

1. The processor must be executing a REP MOVS instruction.
2. The source and destination operands must reside within the same cache line.
3. There must be a snoop coincident with the REP MOVS.

Because this is a speed path, its occurrence is dependent on temperature, voltage, and process variation (differs from one unit to the next). Failures have only been observed when operating near the upper limit of the temperature range and near the lower limit of the voltage range, and, then, only in a fraction of parts.

When this erratum occurs, the result is that an extra data item is copied during the REP MOVS. For example, in following code sequence, the Dword in memory location 108h may be erroneously copied into memory location 118h. When the error occurs, ESI will contain the value 10Ch rather than 108h and EDI will contain 11Ch rather than 118h.

```
MOV ECX, 2
MOV ESI, 100h
MOV EDI, 110h
REP MOVSD
```

The problem has been only observed in 100-MHz multi- or dual-processor machines with multi-threaded software; there have been no observed failures in uniprocessor systems. Multi- and dual-processing environments have higher processor utilization and more intense snoop activity than uniprocessor systems.

IMPLICATION: When this erratum occurs, the software may malfunction. This erratum has only been observed when running several instances of the WinBEZ* application on Windows NT* 3.1.

The failure may manifest itself in one of four ways:

1. A process window is dropped.
2. The screen locks with a red, vertical stripe.
3. The system hangs completely.
4. An application error message occurs.

WORKAROUND: Intel has implemented a tighter test screen to preclude future instances of this speed path. Operating the L1 cache in writethrough mode reduces the frequency of occurrence and provides additional margin in the system design. For multiprocessor systems with dedicated caches, Intel's TPC benchmark testing indicates that operating the L1 cache in writethrough mode results in less than a 5 percent performance impact.

For shared-cache dual-processor systems, the performance impact is significantly higher, and L1 cache writethrough operation is not recommended.

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

20. Overflow Undetected on Some Numbers on FIST

PROBLEM: On certain large positive input floating-point numbers, and only in two processor rounding modes, the instructions FIST[P] m16int and FIST[P] m32int fail to process integer overflow. As a consequence, the expected exception response for this situation is not correctly provided. Instead, a zero is returned to memory as the result.

The problem occurs only when all of the following conditions are met:

1. The FIST[P] instruction is either a 16- or 32-bit operation; 64-bit operations are unaffected.
2. Either the 'nearest' or 'up' rounding modes are being used. Both 'chop' and 'down' rounding modes are unaffected by this erratum.
3. The sign bit of the floating-point operand is positive.
4. The operand is one of a very limited number of operands. The table below lists the operands that are affected. For an operand to be affected, it must have an exponent equal to the value listed and a significand with the most significant bits equal to '1'. Additionally in the 'up' rounding mode, at least one of the remaining lower bits of the significand must be '1'. The Exponent and the two Significand columns describe the affected operands exactly, while the values in the column titled 'Approximate Affected Range' are only for clarity.

FIST[P] Operation	Rounding Mode	Unbiased Exponent	Upper Bits of Significand	Lower Bits of Significand	Approximate Affected Range
16 bit	Up	15	16 MSB's = '1'	At least one '1'	65,535.50 ± 0.50
	Nearest	15	17 MSB's = '1'	don't care	65,535.75 ± 0.25
32 bit	Up	31	32 MSB's = '1'	At least one '1'	4,294,967,295.50 ± 0.50
	Nearest	31	33 MSB's = '1'	don't care	4,294,967,295.75 ± 0.25
64 bit	Problem does not occur				

ACTUAL vs. EXPECTED RESPONSE

Actual Response

When the flaw is encountered, the processor provides the following response:

- A zero is returned as a result. This result gets stored to memory.
- The IE (Invalid Operation) bit in the FPU status word is not set to flag the overflow.
- The C1 (roundup) and PE bits in the FPU status word may be incorrectly updated.
- No event handler is ever invoked.

Expected Response

The expected processor response when the invalid operation exception is masked is:

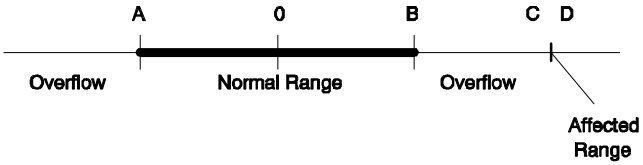
- Return a value 10000....0000 to memory.

- Set the IE bit in the FPU status word.

The expected processor response when the invalid operation exception is unmasked is:

- Do not return a result to memory. Keep the original operand intact on the stack.
- Set the IE bit in the FPU status word.
- Appropriately vector to the user numeric exception handler.

IMPLICATION: An unexpected result is returned when an overflow condition occurs without being processed or flagged. Integer overflow occurs rarely in applications. If overflow does occur, the likelihood of the operand being in the range of affected numbers is even more rare. The range of numbers affected by this erratum is outside that which can be converted to an integer value. The figure below and corresponding table detail the normal range of numbers (between A and B) and the range affected by this erratum (between C and D):



16-bit Operation	A	B	C	D
Round Nearest	[-32,768.5]	(+32,767.5)	[+65,535.5]	(+65,536.0)
Round Up	(-32,769.0)	[+32,767.0]	(+65,535.0)	(+65,536.0)
32-bit Operation	**A**	**B**	**C**	**D**
Round Nearest	[-2,147,483,648.5]	(+2,147,483,647.5)	[+4,294,967,295.5]	(+4,294,967,296.0)
Round Up	(-2,147,483,649.0)	[+2,147,483,647.0]	(+4,294,967,295.0)	(+4,294,967,296.0)

NOTE:
[xxx.x] indicates the endpoint is included in the interval; (xxx.x) indicates the endpoint is *not* included in the interval.

Furthermore, given that the problem cannot occur in the 'chop' rounding mode, and given that the 'chop' rounding mode is the standard rounding mode in ANSI-C and ANSI-FORTRAN 77, it is unlikely that this condition will occur in most applications.

This erratum is not believed to affect application programs in general. Applications will need to handle exceptional behavior and take the appropriate actions to recover from exceptions. In order to do this applications will need to do either range checking prior to conversion or implement explicit exception handling. If an application relies on explicit exception handling the possibility of an error exists. It is, however, believed that applications written in languages that support exception handling will most likely do range checking, thereby allowing the application to be compiled with a no check option for performance reasons when the application has been debugged.

WORKAROUND: Any of three software workarounds will completely avoid occurrence of this erratum:

1. Range checking performed prior to execution of the FIST[P] instruction will avoid the overflow condition from occurring, and is likely to already be implemented.
2. Use the 'chop' or 'down' rounding modes. This erratum will never occur in these modes. By default, both ANSI-C and FORTRAN will use the 'chop' mode.
3. Use the FRNDINT (Round floating-point value to integer) instruction immediately preceding FIST[P]. FRNDINT will always round the value correctly.

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

21. Six Operands Result in Unexpected FIST Operation

PROBLEM: For six possible operands and only in two processor rounding modes (up, down), the FIST or FISTP (floating-point to integer conversion and store) instructions return unexpected results to memory. Additionally, incorrect status information is returned under certain conditions in all 4 rounding modes.

The flaw occurs only on certain operands on the instructions FIST[P] m16int, FIST[P] m32int, and FIST[P] m64int. These operands are ±0.0625, ±0.125, and ±0.1875.

The following table details the conditions for the flaw and the results returned. For use of any of the six above-listed operands, refer to the left-hand side of the table in the column for a given combination of sign and rounding mode. The corresponding right-hand side of the table shows the results which will occur for the given conditions. These results include the C1 (Condition Code 1) and PE (Precision Exception) bits and, in two instances, storing of unexpected results.

Operand (any one of)	Rounding Mode	Result Expected / Actual	Status Bits	
			PE Expected / Actual	C1 Expected / Actual
+0.0625	nearest	SAME	1 / Unchanged	SAME
+0.1250	chop	SAME	1 / Unchanged	SAME
+0.1875	down	SAME	1 / Unchanged	SAME
	up	0001 / 0000	1 / Unchanged	1 / 0
–0.0625	nearest	SAME	1 / Unchanged	SAME
–0.1250	chop	SAME	1 / Unchanged	SAME
–0.1875	down	- 0001 / 0000	1 / Unchanged	1 / 0
	up	SAME	1 / Unchanged	SAME

IMPLICATION: An incorrect result (0 instead of +1 or -1) is returned to memory for the six previously listed operands. Incorrect results are returned to memory only when in the ‘up’ rounding mode with a positive sign or in the ‘down’ rounding mode with a negative sign. The majority of applications will be unaffected by this erratum since the standard rounding mode in ANSI-C and ANSI-FORTRAN 77 is ‘chop’, while BASIC and the ISO PASCAL intrinsic ROUND function use ‘nearest’ rounding mode. In addition, ‘nearest’ is also the default rounding mode in the processor.

Incorrect status information is also insignificant to most applications. Given that the PE bit in the numeric status register is ‘sticky’, and that most floating-point instructions will set this bit, PE will most likely have already been set to ‘1’ before execution of the FIST[P] instruction and therefore the PE bit will not be seen to be incorrect. In addition, the unexpected values for the C1 bit are unlikely to be significant for most applications since the only usage of this information is in exception handling.

WORKAROUND: Either of two software workarounds will completely avoid this erratum.

1. Use the FRNDINT (Round floating-point value to integer) instruction immediately preceding FIST[P]. FRNDINT will always round the value correctly.
2. Use of ‘nearest’ or ‘chop’ modes will always avoid any incorrect result being stored (although the PE bit may have incorrect values).

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

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

PROBLEM: If an internal snoop (as a result of ADS#) or external snoop (EADS#) with invalidation (INV) occurs coincident with a page table walk violation, the snoop may fail to invalidate the entry in the instruction cache. A page table walk violation occurs when the processor speculatively prefetches across a page boundary and this page is not accessible or not present. This violation results in a page fault if this code is executed. A page fault does not occur if the code is not executed.

For this erratum to occur, all the following conditions must be met:

1. A snoop with invalidation is run in the code cache. The snoop may be internal or external.
2. The Pentium processor is performing a page table walk to service an instruction TLB miss.
3. The page table walk results in a violation (this may or may not lead to a page or other fault due to a speculative fetch).
4. The EADS# of the external snoop or ADS# of the table update occur within the window of failure.

   The window is defined by:

   a. 1-3 clocks after BRDY# is returned for the page directory or table read.

   b. 2-n clocks after BRDY# is returned for the page directory or table read if the set address of a buffered write matches that of the instruction cache lookup. “n” is determined by the time to complete two new data write bus cycles from the data cache.

IMPLICATION: This erratum has not been observed on any system. It was found only through investigation of component schematics, and Intel has only duplicated it on a proprietary test system by forcing failure conditions using the internal test registers. Its low frequency of occurrence is due to the way most systems operate; DMA devices snoop code 4 bytes at a time so that each line will get snooped and invalidated multiple times.

If this erratum occurs and a line is not invalidated in the instruction cache, then the instruction cache may have a coherency problem. As a result the processor may execute incorrect instructions leading to a GPF or an application error. This erratum affects only self Modifying code and bus masters/DMA devices. Due to necessary conditions, this erratum is expected to have an extremely low frequency of occurrence.

WORKAROUND: There are two workarounds. Because of the rarity of occurrence of this erratum, many OEMs may choose not to implement either workaround.

1. Rewrite the device driver for the DMA devices such that after DMA is complete, the instruction cache is invalidated using the TR5.cntl=11 (flush) and CD=0 (code cache) bits.
2. Disable the L1 cache.

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

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

PROBLEM: For certain input datum the divide, remaindering, tangent and arctangent floating-point instructions produce results with reduced precision.

The odds of encountering the erratum are highly dependent upon the input data. Statistical characterization yields a probability that about one in nine billion randomly fed operand pairs on the divide instruction will produce results with reduced precision. The statistical fraction of the total input number space prone to failure is $1.14 \times 10^{-}$

[10]. The degree of inaccuracy of the result delivered depends upon the input data and upon the instruction involved. On the divide, tangent, and arctangent instructions, the worst-case inaccuracy occurs in the 13th significant binary digit (4th decimal digit). On the remainder instruction, the entire result could be imprecise.

This flaw can occur in all three precision modes (single, double, extended), and is independent of rounding mode. This flaw requires the binary divisor to have any of the following bit patterns (1.0001, 1.0100, 1.0111, 1.1010 or 1.1101) as the most significant bits, followed by at least six binary ones. This condition is necessary but not sufficient for the flaw to occur. The instructions that are affected by this erratum are: FDIV, FDIVP, FDIVR, FDIVRP, FIDIV, FIDIVR, FPREM, FPREM1, FPTAN, FPATAN.

During execution, these instructions use a hardware divider that employs the SRT (Sweeney-Robertson-Tocher) algorithm which relies upon a quotient prediction PLA. Five PLA entries were omitted. As a result of the omission, a divisor/remainder pair that hits one of these missing entries during the lookup phase of the SRT division algorithm will incorrectly predict a intermediate quotient digit value. Subsequently, the iterative algorithm will return a quotient result with reduced precision.

The flaw will **not** occur when a floating-point divide is used to calculate the reciprocal of the input operand in single precision mode, nor will it occur on integer operand pairs that have a value of less than 100,000.

IMPLICATION: For certain input datum, there will be a loss in precision of the result that is produced. The loss in precision can occur between the 13th and 64th significant binary digit in extended precision. On the remainder instruction the entire result could be imprecise.

The occurrence of the anomaly depends upon all of the following:

1. The rate of use of the specific FP instructions in the Pentium processor.
2. The data fed to them.
3. The way in which the results are propagated for further computation by the application.
4. The way in which the final result of the application is interpreted.

Because of the low probability of the occurrence with respect to the input data (one in nine billion random operand pairs), this anomaly is of low significance to users unless they exercise the CPU with a very large number of divides and/or remainder instructions per day, or unless the data fed to the divisor is abnormally high in sensitive bit patterns.

WORKAROUND: Due to the extreme rarity of this flaw, a workaround is not necessary for almost all users. However, Intel is replacing components for end users and OEM's upon request. In addition, a software patch is available for compiler developers. If you are a compiler developer, contact your local Intel representative to obtain this, or download from the World Wide Web Intel support server (www.intel.com).

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

A white paper, *Statistical Analysis of Floating-point Flaw in the Pentium Processor,* (Order Number 242481), is available that includes a complete analysis performed by Intel.

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

PROBLEM: HARDWARE FLUSH and INIT requests and Machine Check exceptions may be dropped if they occur coincidentally with a floating-point exception.

The following conditions are necessary to create this erratum:

1. Two floating-point instructions are paired and immediately followed by an integer instruction.
2. The floating-point instruction in the u-pipe causes a floating-point exception.

3. The FLUSH, INIT, or Machine Check occurs internally on the same clock as the integer instruction.

IMPLICATION: The processor caches will not be flushed, or the INIT request may be dropped.

WORKAROUND: None identified at this time.

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

25. Floating-point Operations May Clear Alignment Check Bit

PROBLEM: The Alignment Check bit (bit 18 in the EFLAGS register) may be inadvertently cleared.

This erratum occurs if a fault occurs during execution of the FNSAVE instruction. After servicing the fault and resuming and completing the FNSAVE, the AC bit will be '0'. Expected operation is that the contents of AC are unchanged.

IMPLICATION: The only known use being made of the AC bit, at this time, is to aid code developers in aligning data structures for performance optimizations. As a result, there are no hardware or system application implications known to Intel at this time. Operating systems and applications will not be affected by this erratum.

WORKAROUND: None identified at this time.

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

26. CMPXCHG8B Across Page Boundary May Cause Invalid Opcode Exception

PROBLEM: Use of the new Pentium processor specific CMPXCHG8B instruction may result in an invalid opcode exception.

If the CMPXCHG8B opcode crosses a page boundary such that the first two bytes are located in a page which is present in physical memory and the remaining bytes are located in a non-present page, unexpected program execution results. In this case, the processor generates an Invalid Opcode exception and passes control to exception handler number 6. Normal execution would be for a Page Fault to occur when the non-present page access is attempted, followed by reading in of the requested page from a storage device and completion of the CMPXCHG8B instruction.

IMPLICATION: This erratum only affects existing software which is Pentium processor aware and uses the CMPXCHG8B instruction. Any occurrence would generate an invalid opcode exception and pass control to exception handler 6.

WORKAROUND: Any software which uses Pentium processor specific instructions should ensure that the CMPXCHG8B opcode does not cross a page boundary.

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

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.

IMPLICATIONS: 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 at the beginning of this section.

29. FRC Lock Step Failure During APIC Write

PROBLEM: During FRC, the APIC write cycle is not driven on the external bus. When the internal APIC write data is compared with data on the external bus, the checker processor sees a comparison error and asserts IERR#.

IMPLICATIONS: This erratum only affects operation during FRC mode and will cause inadvertent IERR#'s to occur.

WORKAROUND: Ignore IERR# when doing APIC write cycles.

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

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

PROBLEM: Due to strong internal pull down resistors on BE4#-BE0#, these pins are pulled low upon RESET high. A minimum input high voltage may be read into these pins incorrectly.

IMPLICATION: An input high voltage could be sampled as a "0", for instance, during APIC identification or boundary scan. This violates IEEE spec 1149.1 which states that regardless of the state of RESET in the boundary scan mode, the value driven should be that in the boundary scan cell.

WORKAROUND: To ensure a logical "1" is read, drive a minimum Input High Voltage (Vih) of 3.0V @6mA into these pins.

Otherwise, if testing boundary scan in a system environment which does not meet the above criteria, these pins can be left untested by marking these pins as "INTERNAL" in the BSDL file.

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

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

PROBLEM: In a DP system, weak output drivers are enabled at PCHK# during the falling edge of RESET when APICEN and DPEN# are enabled on the primary processor, (and CPUTYP tied to Vss). For the dual processor the weak driver at PCHK# is enabled whenever APICEN is enabled, (and CPUTYP tied to V_{CC}). The weak driver at PCHK# will drive a "1" with a long rise time, exceeding the specification for PCHK# max valid delay.

IMPLICATION: A weak "1" driven by PCHK# could be sampled as "0" during boundary scan testing. The maximum frequency of TCK may have to be decreased. This violates IEEE spec 1149.1 which states that regardless of the state of APICEN, DPEN#, CPUTYP or RESET in the boundary scan mode, the value driven should be that in the boundary scan cell.

WORKAROUND: To ensure a "1" is driven during boundary scan test, from the falling edge of TCK wait at least 55 nS before sampling PCHK#. The board's capacitve load is estimated at 40pf. Please refer to *Pentium Processor Family Developer's Manual*, Volume 1, Section 21.1.28 which specifies Ro=360Ohms for the PCHK# weak driver.

Otherwise, if testing boundary scan in a system environment which does not meet the above criteria, these pins can be left untested by marking these pins as "INTERNAL" in the BSDL file.

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

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

PROBLEM: A specific sequence of code may cause corruption of the EIP. This specific code sequence must contain **all** of the following characteristics:

1. Three consecutive FP instructions
2. The third FP instruction must be immediately followed by a MOV Sreg, Reg instruction. (Sreg means one of the 6 segment registers. Note that **all other** ways of changing an Sreg, such as "POP Sreg", "MOV Sreg, Mem", LES, LDS, LSS etc., and far JMPs, CALLs & RETs etc., will **not** cause this problem.)
3. The descriptor selected by the MOV Sreg, Reg must not be available in the on-chip cache of the most recently used descriptors.
4. In addition, there are specific restrictions on each of the 3 consecutive FP instructions:
5. The first instruction in the sequence must be an FP instruction which can be paired with FXCH (FADD, FSUB, FMUL, FDIV, FLD, FCOM, FUCOM, FCHS, FTST & FABS.)
6. The second FP instruction must be FXCH, since it's the only FP instruction that can go down the v-pipe (a pair of FP instructions must run together in u and v to allow this problem to occur).
7. The last (third) FP instruction must cause the CPU to test for an unsafe condition. Instructions that compare 2 numbers, and adjust FP flags as the result (FCOM(P), FICOM(P), FUCOM(P) & FTEST) are the usual sources of this test, but only with certain arguments (e.g. NANs and infinities). (In rare situations, FDIV (with a denominator of zero) and FSQRT (negative argument) will cause an unsafe condition test.)

```
Example:    FADD            ; is able to go into the u pipe & pair with FXCH in the v pipe
            FXCH            ; must be in the v pipe for this problem to occur
            FCOMP           ; can cause test for an unsafe condition
            MOV Sreg, Reg   ; selected descriptor must be a "miss' in the segment descriptor cache
```

IMPLICATION: If this erratum occurs, an erroneous value is written into the EIP (instruction pointer), causing unpredictable results. This will most frequently result in an invalid opcode exception.

Intel knows of no existing application or OS code which involves this sequence. First, FXCH is rarely used just before a compare type instruction. Usually a calculation would be done just before compare which would set up one of the numbers to be compared at the FP stack top. Further, to make use of the compare, the FP flags must first be stored in a location where they can be tested, or loaded into EFLAGS for testing, so the next instruction is typically FSTSW, not an Sreg load. Also, Sreg loads are typically not done by application code in a 32-bit environment.

WORKAROUND: Move any instruction between the compare and the Sreg load, such as FSTSW. Even NOP will work.

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

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 at the beginning of this section.

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

PROBLEM: This is a generalization of the Dual Processing specific erratum 5DP. Although this erratum has not been verified in a real Pentium processor-based system without using the DP mode, detailed analysis with the simulation model indicates that it is possible for the erratum to occur as described here. The possible occurrence requires the following conditions:

1. The processor begins a WRITE cycle to a writeback (WB) line in its L1 cache that is in the (S) (shared) state.
2. A non-invalidating snoop using AHOLD/EADS# is generated by another bus master by reading the *same* cache line *before* the completion of the WRITE.
3. There is at least one more cache in the system that holds a copy of the cache line.

When the snoop occurs during this window, the snoop is mishandled and the cache line will transition to the exclusive (E) state instead of the shared (S) state. An additional write to the same cache line by the processor will cause a cache state transition from (E) to modified (M) and will not generate a bus cycle. Since a bus cycle is not generated, other caching agents in the system that hold the cache line in the shared (S) state will not be updated and will contain stale data.

This erratum occurs because the sequence of cycles completed inside the processor is different from the sequence of cycles started on the bus, which is a memory write by the processor followed by a snoop on the same address. Inside the processor, the snoop occurs, and then the memory write completes. This can only happen if the snoop occurs in a window between the ADS# assertion by the processor for its WRITE cycle, and up to 2 CPU clocks after the system assertion of BRDY# at the end of the WRITE cycle.

This erratum can only occur if AHOLD/EADS# is used for snoops; if HOLD or BOFF# are used to force a snoop before the processor WRITE is completed, it will be restarted after the snoop and handled correctly. In addition, this erratum cannot occur on memory updates where the LOCK# signal is asserted.

IMPLICATION: The processor's L1 cache may become incoherent with an external cache. A memory read cycle by an external bus master could read stale data.

WORKAROUND: Designs which do not snoop under AHOLD are not affected. Uniprocessor systems using a lookaside L2 cache, such as those built with either the 82430NX or 82430FX PCIsets, are not affected because a read by an external bus master will always snoop the L1 as well as L2. Intel knows of no uniprocessor implementation which is subject to this erratum. While this erratum is more likely to occur in a multi-processing environment, Intel is not aware of any designs which have demonstrated a susceptibility to this issue.

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

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

PROBLEM: During boundary scan, BE3#-BE0# are always tristated when RESET is high regardless of the value in the control cell. If RESET is low the proper value will be driven.

IMPLICATION: When RESET is high stong pull downs are enabled, and the value of "0" will always appear at the BE3#-BE0# pins. This violates IEEE spec 1149.1 which states that regardless of RESET in the boundary scan mode, the value driven should be that in the boundary scan cell.

WORKAROUND: If testing boundary scan in a system environment, these pins can be left untested by marking these pins as "INTERNAL" in the BSDL file.

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

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 *Pentium Processor Family Developer's Manual*, Volume 3, Section 16.2. This will set the FIP to 00000000h. Otherwise, no workaround is required.

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

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 at the beginning of this section.

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

PROBLEM: On the Intel486, 80386 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 itself (with FLDENV, FRSTOR) or its image in memory (with FSTENV, FSAVE) partly updated, thus corrupted, and the instruction generally is non-restartable. In Section 23.3.3.9, Volume 1, the *Pentium Processor Family Developer's Manual* states 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. But 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 in Section 23.3.6.2, Volume 1, the *Pentium Processor Family Developer's Manual* 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.

WORKAROUNDS:

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 at the beginning of this section.

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 at the beginning of this section.

40. Current in Stop Clock State Exceeds Specification

PROBLEM: During the stop clock mode, a portion of the internal processor circuitry may remain active. This problematic circuitry is part of the boundary scan logic, and is not properly reset unless boundary scan is run. The current drawn will result in a violation of the stop clock power dissipation specification.

IMPLICATION: The Pentium processor will draw more current than specified for the stop clock state. The specification for Stop Clock Power dissipation is 50mW. Under the condition of the erratum, the power consumption can reach 60-80mW, depending on the type of processor and core voltage.

WORKAROUND: Use one of following:

1. Ground TCK or TRST#. Note that this will NOT allow boundary scan to be run.
2. Ground TCK or TRST# through a 1K Ohm resistor. This will allow a normal boundary scan to be performed if desired.

For power sensitive designs, please observe that as both TCK and TRST# have an internal 30K Ohm pull-up resistor, the workarounds will not result in any significant additional current draw.

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

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 at the beginning of this section.

42. Incorrect Decode of Certain 0F Instructions

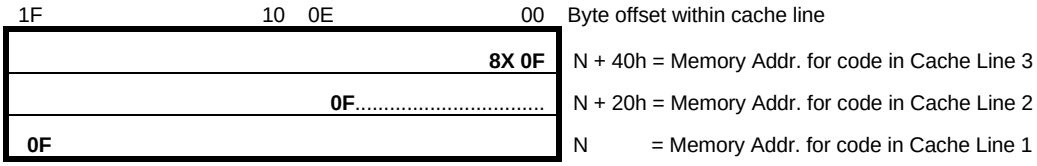
PROBLEM: With a specific arrangement of instructions in memory and certain asynchronous events, the processor may incorrectly decode certain 0F prefixed instructions.

In order for this erratum to occur there must be a very specific arrangement of instructions in memory. In conjunction, these instructions must be resident in the cache and an asynchronous cache line replacement must occur during execution of these instructions. The conditions for this erratum are as follows:

1. There **is a 0F prefix instruction (other than 0F80-0F8F) which can begin in the range of the most significant byte of the first cache line (1F) through the 0E byte of the second cache line**, followed by the rest of the 0F prefixed instruction. (See cache lines 1 & 2 in Fig. 1; the 0F byte is at offset 1F in line 1.)
2. The processor must **execute a branch to the second half of the first cache line shown below (10 - 1F**).
3. There is a **replacement or invalidation of cache line 2** shown below. This replacement or invalidation must complete **in a narrow window** (3 or less CPU clocks) between the decode of the 0F byte from the instruction queue, and the decode of the rest of the instruction.
4. The **third consecutive cache line must contain the bit pattern 0F80 - 0F8F offset by 33 bytes from the 0F byte of the first instruction.** There must be a spacing of exactly 33 bytes between the first and subsequent 0F bytes.
5. All 3 lines must already be resident in the instruction cache and must be from sequential linear memory addresses.

This erratum can occur only if **all** of the above conditions are met. After the first byte (0F) of the opcode is decoded, but before the rest of the instruction can be decoded, the second cache line gets replaced or invalidated. While the processor is waiting for the line containing the second byte of the 0F opcode to be read in again from memory, the 2 bytes offset by 33 bytes from the 0F byte of the stalled instruction are temporarily presented to the instruction decoder. Normally this data would be completely ignored, however if the bit pattern is in the range of 0F80 - 0F8F, then the decode of the 0F byte of the stalled instruction is discarded. (Note that this happens **only** with the 0F prefixed instructions.) When the cache line fill has returned the missing line, the second byte of the stalled instruction is incorrectly interpreted as the start of the instruction.

3 Consecutive L1 Cache Lines, Holding Consecutive Code



IMPLICATION: When this erratum occurs, the processor will execute invalid or erroneous instructions. Depending on software and system configuration, the user will typically see an application error message or system reset.

WORKAROUND: There are currently no workaround identified for existing code.

This erratum may be eliminated in code that is being created or recompiled as follows: The compiler must check for the occurrence of bytes 0F, not 8X, 31 other bytes, 0F and 8X. When such an occurrence is found, a NOP inserted or any other change in spacing will prevent the alignment required for this erratum to occur.

A loader based workaround can also be implemented as follows: At load time, scan the executable for the existence of a 0F instruction (other than 0F8X), check the cache alignment of the 0F instruction and check for the existence of a 0F8X bit pattern 33 bytes beyond the 0F byte of the first instruction. If these conditions are found, the page containing this code sequence can be marked as non-cacheable.

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

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 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 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 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 at the beginning of this section.

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 at the beginning of this section.

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 at the beginning of this section.

46. BUSCHK# Interrupt Has Wrong Priority

PROBLEM: Section 3.4 of the Pentium Processor Data Book 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. Section 30.5.2 further specifies 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 at the beginning of this section.

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 at the beginning of this section.

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

PROBLEM: Assertion of STPCLK# can interfere with a feature described in the Pentium Processor Family Developer's Manual Vol. 3 on p. 17-5: "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 at the beginning of this section.

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 at the beginning of this section.

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 inititiate 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 external pullup.

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

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 Section 14.9.8 in the *Pentium Processor Family Developer’s Manual*, Volume 3 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 at the beginning of this section.

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. (E.g. 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 at the beginning of this section.

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 incidentally 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 Section 23.3.6.2 of Volume 3 of the *Pentium Processor Family Developer's Manual*) 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:

1. Do not use 16 bit addressing to cause wraps at 64K inside a USE32 segment.
2. 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 at the beginning of this section.

54. V86 Interrupt Routine at Illegal Privelege 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 on p. 25-176 in the *Pentium Processor Family Developer's Manual*, Volume 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 at the beginning of this section.

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 at the beginning of this section.

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 14-3 of the *Pentium Processor Family 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.

Note that Case 2 does not apply to OverDrive processors.

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

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 at the beginning of this section.

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

PROBLEM: Unused input signals are typically tied off (either high or low). 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. >2KOhms), and CPU I/O power supply (V_{CC3}) ramps before CPU core power supply (V_{CC2}), 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 at the beginning of this section.

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 on page F-15 in the *Pentium Processor Family Developer's Manual,* Volume 3. 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 at the beginning of this section.

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 *Pentium Processor Family Developer's Manual,* Volume 3, Section 16.6.2, lists a sequence of 9 steps for switching from protected mode to real-address mode. Listed here is step 1.

1. If paging is enabled, perform the following sequence:
   a. - 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.
   b. - Clear the PG bit in the CR0 register.
   c. - 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 at the beginning of this section.

61. STPCLK# or FLUSH# 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# or FLUSH# is asserted and serviced before the instruction boundary of this next instruction.

IMPLICATION: External interrupts assumed blocked until after the instruction following STI may be recognized before this instruction executes. No operating system is known by Intel to be affected.

WORKAROUND: None identified at this time.

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

62. REP String Instruction Not Interruptable by STPCLK#

PROBLEM: The *Pentium Processor Family Developer's Manual,* Volume 3, Chapter 24 under the REP string instruction (page 25-269), 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 at the beginning of this section.

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 at the beginning of this section.

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 at the beginning of this section.

65. TRST# Not Asynchronous

PROBLEM: TRST# is not an asynchronous input as specified in section 5.1.56 of the *Pentium Processor Family Developer’s Manual*, Volume 1.

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 at the beginning of this section.

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 at the beginning of this section.

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,* Volume 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, using the methods described in the *Pentium Processor Family Developer's Manual,* Volume 1 may not terminate the Hi-Z state.

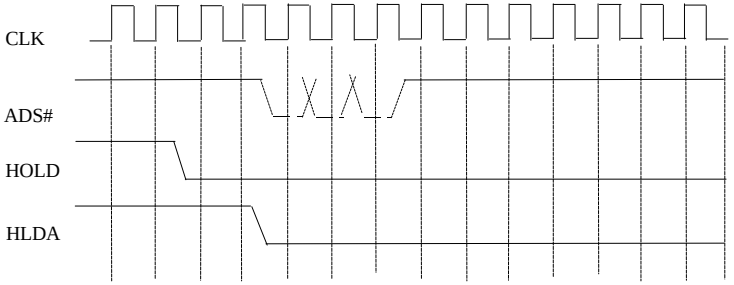
WORKAROUND: To exit TAP Hi-Z state, in addition to the methods described above, the processor needs to be RESET as well.

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

69. ADS# May be Delayed After HLDA Deassertion

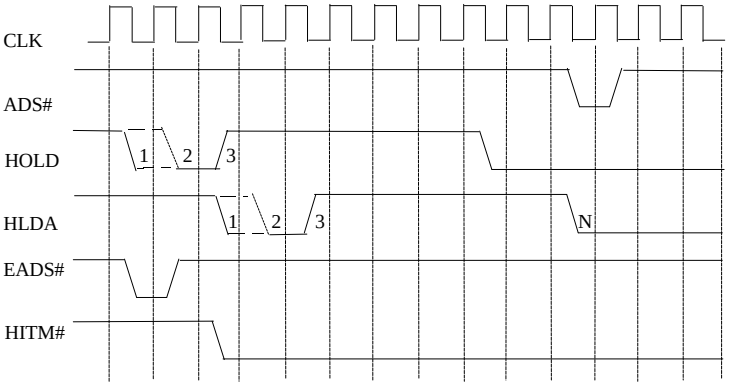
PROBLEM: The *Pentium Processor Family Developer's Manual*, Volume 1, Section 5.1.26 states, "If the Pentium processor has a bus cycle pending, it will be driven in the same clock that HLDA is deasserted." The Pentium processor typically starts the cycle on the same clock that HLDA is deasserted, however, it may be delayed by as many as two clocks. See diagram below

Pending Cycle May Be Delayed



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.

Snoop Writeback Cycle Delayed



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 at the beginning of this section.

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.

IMPLICATIONS: 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 at the beginning of this section.

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 at the beginning of this section.

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

PROBLEM: In a dual processor system, the following sequence of events can cause the processors to lock up:

1. There are two cycles pending on the bus, one from each processor. In this case the first cycle is from the C and the second from CM; therefore, C is the LRM and CM is the MRM.
2. AHOLD is asserted and then an external snoop occurs (EADS# is asserted) causing a hit to a modified line in the CM. Since the CM is the MRM, it does not give an indication to the C that it has been hit.
3. Now a BOFF# is asserted and backs off the pending cycles. Once BOFF# is released, the C becomes the MRM in order to maintain cycle order.
4. C now owns the bus but cannot run its cycle because AHOLD is still active. Since C is not aware that CM has been hit by an external snoop, C is not willing to give up the bus to the CM and thus prevents the CM from performing a writeback. Since the C will not give up the bus to the CM and cannot run its own cycle, the system hangs.

IMPLICATION: If each processor in a dual processor system has a cycle pending on the bus and an external snoop results in a hit to a modified line, the processors may lock up.

WORKAROUND:

1. Disable pipelining.
2. Deassert AHOLD no earlier than one clock after BOFF# has been deasserted. Note that if this workaround is used, the system will continue to run but a re-ordering of cycles will occur. The C will run its cycle first (rather than the writeback occurring first), and then grant the bus to the CM to complete its writeback cycle and then its outstanding cycle.

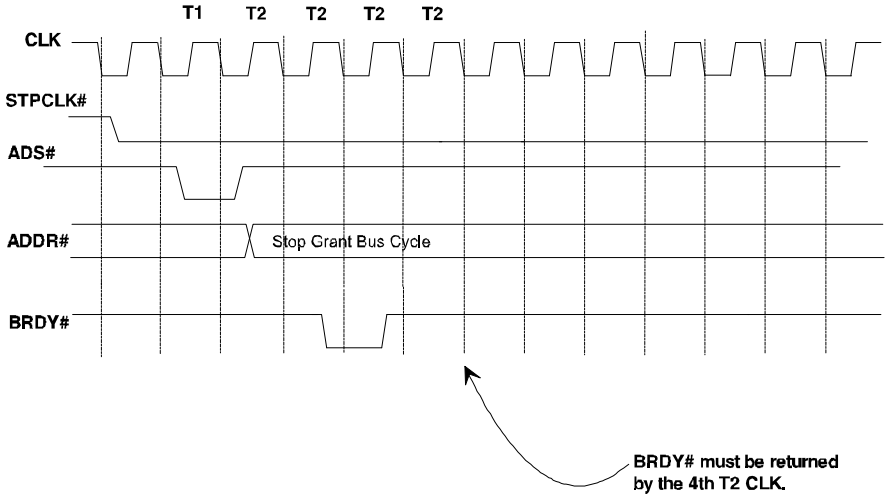
Designs based on the 82430NX PCIset and other chip sets which do not generate an external snoop when two cycles are pending on the bus are not affected.

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

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

PROBLEM: If a STPCLK# interrupt occurs and BRDY# is not asserted within 4 CLKs following the Stop Grant bus cycle, then the processor which ran the Stop Grant bus cycle may hang.

IMPLICATION: This problem occurs only in dual processor systems and will cause one of the processors to hang.



WORKAROUND:

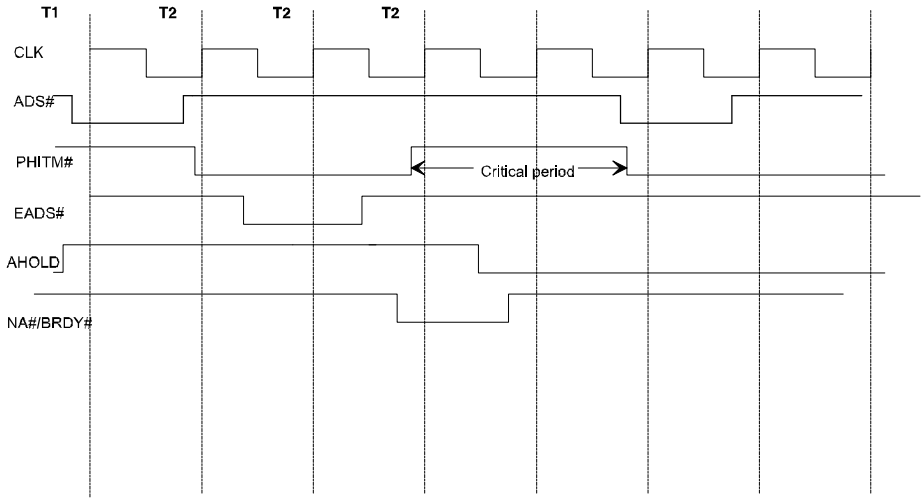
1. Do not assert STPCLK#.
2. Ensure that BRDY# is asserted within 4 CLKs after the Stop Grant bus cycle has begun.

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

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

PROBLEM: The following sequence of events may cause one of the processors in a dual processing system to hang:

1. The MRM (Most Recent Bus Master, this could be the C or CM processor) issues an ADS# of a memory-write (or memory read) cycle and an AHOLD assertion follows.
2. This ADS# causes an automatic snoop by the LRM (Least Recent Bus Master) and hits a modified line in the LRM causing PHITM#/PHIT# to be asserted.
3. After PHITM# is asserted, EADS# is asserted by the system and does **not** hit a modified line. This causes the LRM to deassert PHITM# for one or two clocks, and then assert PHITM# again.
4. One or two clocks after EADS# is deasserted, AHOLD is deasserted.
5. BRDY# or NA# is asserted within two clocks after EADS# is deasserted. With a BRDY# or NA# assertion, the MRM samples the PHITM# pin before driving the next cycle on the bus. If the MRM samples PHITM# high, during the "critical period" (shown in the figure below, the critical period is defined as the two clock periods after EADS# is sampled active.), the MRM will incorrectly issue the ADS# of its memory-write cycle again before surrendering the bus to the LRM to do its writeback.
6. After the memory-write cycle is complete, the LRM performs its writeback.
7. The MRM then re-issues the memory write-cycle again. This cycle, now being issued for the third time, causes an internal hangup in the MRM.



IMPLICATION: If a memory write or memory read cycle is pending on the bus and an external snoop occurs, one of the processors may hang.

WORKAROUND:

1. Ensure that NA#/BRDY# is asserted before, or after the "critical period". (the critical period is the two CLK period after EADS# is sampled active.)

   or

2. Ensure that AHOLD is held high for a minimum of three clocks after EADS# has been sampled active. See the following figure.

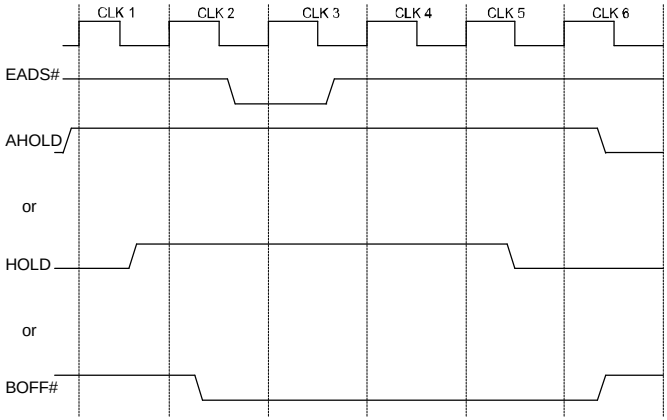
   or

3. Drive HOLD for two clocks after the EADS# was sampled active. Removal of HOLD can be done without regard to the HLDA signal. See the following figure.

   or

4. Drive BOFF# for 3 clocks after the EADS# was sampled active. See the following figure.

All of the listed workarounds prevent the MRM processor from starting a new cycle, thus preventing the third restart of the pending cycle.



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

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

PROBLEM: This is a dual processor erratum: There is a very short setup and hold time for the address bus lines in Dual Processing mode to support the single clock interprocessor snoop response required for cache coherency. In this case the 3ns setup specification is enough for the address to propagate to the cache but it does not allow enough time for the address to propagate to the parity calculation circuits. There is a chance that the APCHK# line will spuriously go active showing a parity error on the address bus.

IMPLICATION: Internal measurements of these address signals show that if the 3ns spec is met, the addresses will be latched correctly. Since the parity portion of the circuitry does not meet this timing there is no way to guarantee the APCHK# output is valid. Systems using the APCHK# pin will perform the APCHK# interrupt error routines.

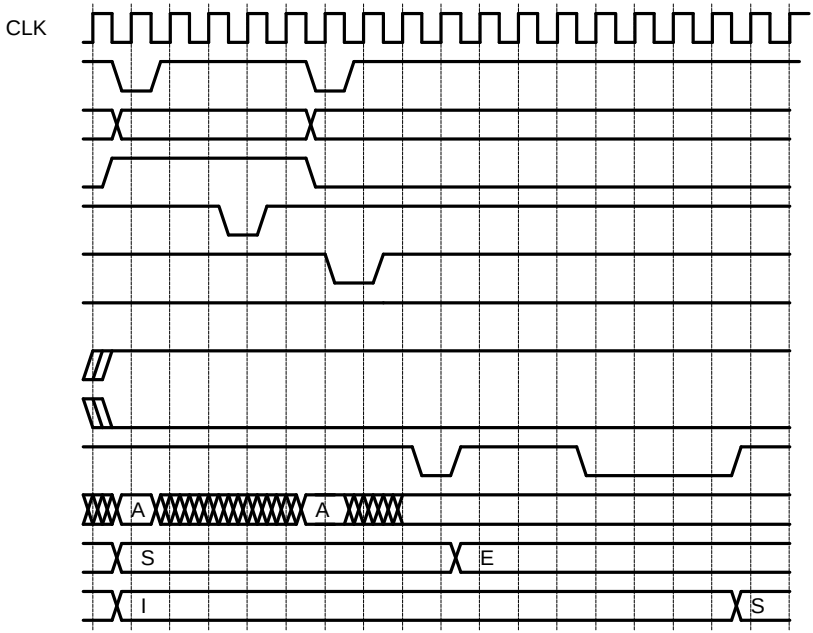
WORKAROUND: Designs based on the 82430NX PCIset and other chip sets which ignore APCHK# in a dual processor environment are therefore not affected. If the address setup (t_{83}) and Maximum Valid Delay (t_6) timings in the *Pentium Processor Family Developer's Manual*, Volume 1, Section 23.4.5.2 and 23.4.5.3 are followed, this erratum will not apply to C-step components and future steppings.

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

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

PROBLEM: This is a dual processor erratum: If there is a READ generated by processor 2 pipelined into a WRITE generated by processor 1, and both of these cycles are to the identical address, the cache states for these lines become inconsistent. In this case the WB/WT# pin is driven HIGH, and the KEN# pin is active. Processor 1 will have data that changes from the (S) shared state to the (E) exclusive state, and processor 2 will have data that changes from the (I) invalid state to the (S) shared state. This violates the cache coherency protocol, since any writes to the line that is in the (E) state in processor 1 will not be seen on the bus, resulting in the second processor operating with stale data. This is a symmetrical problem, such that the initiator of the WRITE cycle can either be the primary or dual processor in a two processor configuration. The reason this happens is that each ADS# causes a snoop in the LRM processor, but in this case at the time of the snoop the

line state is (S) which will generate the PHIT#, and no PHITM#, the transition to the (E) state has been posted but is not performed until the BRDY# of the WRITE cycle.



IMPLICATION: If the described conditions are met, then an additional write to the same address in processor 1 will cause a cache state transition from (E) to (M) and will not generate a bus cycle. This will mean that the data in processor 2 is stale but it could continue operating with the stale data. For example: When 2 processors are sharing a cached semaphore, and processor 1 is updating the semaphore just as processor 2 is reading the semaphore, then processor 2 would eventually end up with stale data in its cache. Due to the nature of the problem, this could cause a number of unknown system problems and may or may not cause a system to hang.

WORKAROUND: Disable pipelining while using dual processing, this is done by not asserting NA#.

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

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

PROBLEM: In a dual processing system when the following conditions are met: A non-bursted read or write cycle that hits a modified line in the data cache is pipelined into a data read cycle, and this dual processing system is running in zero Wait State mode to the L2 Cache, (2-1-1-1). If the ADS# for the pipelined cycle is in the same clock as the 3 or 4th BRDY# of the bursted data read then the processors will hang.

IMPLICATION: Possible system hang in dual processing operation. System restart/reboot would be required.

WORKAROUND: Use one of the following:

1. Run the dual processing configuration in a non-zero Wait State mode, a (3-1-1-1) burst operation has an anticipated impact of 2-4 percent performance decrease. This is how the 82430NX PCIset operates.

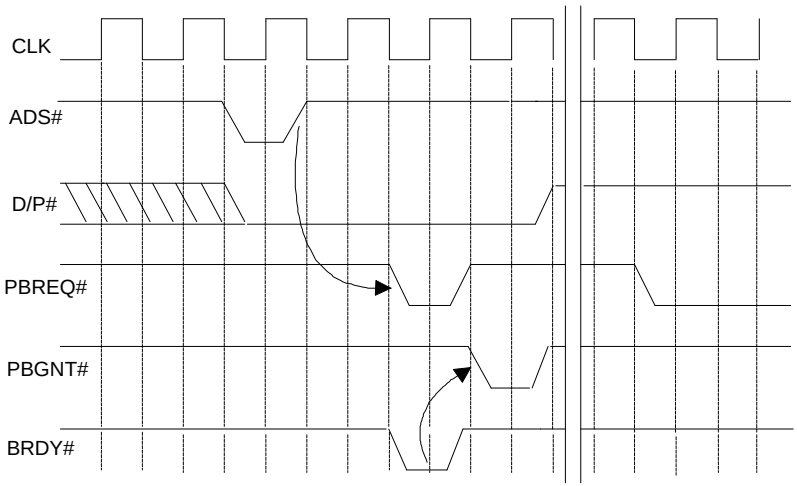
2. Disable pipelining by setting NA# pin high.

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

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

PROBLEM: In a dual processor system with the CPUs operating in 2/3 (bus/core) Bus Fraction mode, the following sequence of events may cause the system to lock up:

1. The LRM processor is 'spinning' on a semaphore location, with interrupts disabled, waiting for the other CPU to complete its task.
2. The MRM (CPU A) issues a bus cycle by asserting ADS#.
3. The LRM (CPU B) samples the ADS# from the MRM and initiates an internal snoop.
4. Due to an internal circuit problem, the snoop may cause the LRM to assert PBREQ# for one clock erroneously even though it does not have a bus cycle to run. The LRM deasserts PBREQ# in the next clock.
5. The MRM (CPU A) on seeing PBREQ# asserted, grants the bus to the LRM (CPU B) by asserting PBGNT#.
6. Since CPU B actually does not need the bus, it does not run any bus cycle but it continues to own the bus.
7. CPU A asserts PBREQ# to CPU B in order to obtain bus ownership back to run its pending cycle.
8. CPU B, however does not grant the bus back to CPU A since it needs to run a bus cycle before relinquishing the bus ownership. Since interrupts are disabled and the processor is executing in a tight 'spin' loop, it does not have any bus cycles to run and does not relinquish the bus.



IMPLICATION: A system lockup can occur because one CPU requests the bus, while the other CPU does not relinquish bus control. Running Windows NT operating system, it has been observed that when the hang condition occurs, the code inside the Windows NT kernel is always inside a very tight code loop, with at least

one of the processors 'spinning' on a semaphore location, with interrupts disabled, waiting for the other CPU to complete its task.

WORKAROUND: Asserting STPCLK# to the processor that owns the bus will cause the system to come out of the lock up condition. Using a timer, when the PBREQ# signal is seen asserted for a few thousand clocks without the PBGNT# signal asserted, STPCLK# can then be asserted to the processor that owns the bus in order to get it out of the hang condition and resume normal operation. The D/P# pin can be used to tell which processor owns the bus. Alternatively, asserting STPCLK# to both processors will also work.

Although this problem is extremely rare, the failure rate is higher:

1. At lower temperatures, closer to approximately 30° C.
2. With pipelining enabled. Pipelining is disabled by setting NA# pin high.
3. Operating in zero or one Wait State mode.

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

8DP. Incorrect Assertion of PHITM# without PHIT#

PROBLEM: In a dual processing environment, assertion of BOFF# in a pipelined bus situation may cause the same cycle to be completed twice on the bus.

This problem occurs in the following case:

1. The Primary processor issues cycle A, either a memory read or write.
2. While cycle A is in progress, the dual processor obtains the bus and issues pipelined memory cycle B, creating an internal self-snoop in the dual processor. Cycle B is of the type that creates a snoop into the data cache (TLB snoop or a code read), and this snoop hits a modified line. As a result, the dual processor has a pending internal writeback cycle.
3. BOFF# is asserted to both processors, backing off cycles A and B.
4. After deassertion of BOFF#, the Primary processor restarts cycle A.
5. The dual processor erroneously asserts PHITM# (without PHIT#) due to the pending internal writeback cycle. This causes the Primary processor to internally back off cycle A, even though there is no hit to a modified line in the dual processor and cycle A completes on the bus externally.
6. The dual processor receives the bus and issues the pending internal writeback.
7. The dual processor restarts cycle B.
8. The Primary processor receives the bus and erroneously re-issues cycle A, completing it a second time.

IMPLICATION: Cycle A is completed twice on the system bus. In most cases the extra cycle will not create any system problems. If the cycle is to a memory-mapped I/O device, mis-operation could occur.

WORKAROUND: Either of two workarounds will avoid this erratum.

1. Disable pipelining.
2. Do not assert BOFF# during a pipelined cycle condition. Designs based on the 82430NX PCIset and other chip sets which do not assert BOFF# during a pipelined condition are therefore not affected

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

9DP. Double Issuance of Read Cycles

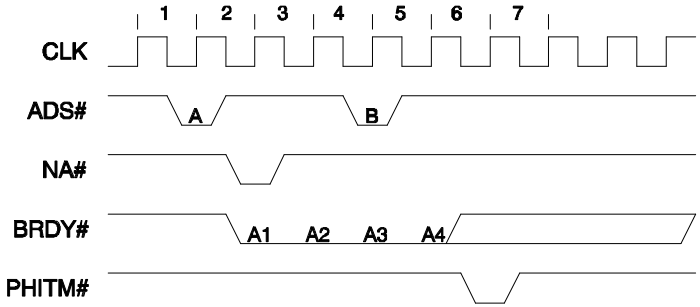
PROBLEM: In a dual processing environment, a memory read cycle (either data read or prefetch) may occur twice. The second read cycle may cause mis-operation of the CPU.

This problem occurs only in dual processing systems with a bus/core ratio of 2/3. For this to occur, pipelining must be enabled with some or all read cycles occurring in zero wait states.

For this erratum to occur, either of two specific sequence of conditions must occur on the bus as shown in the figures below:

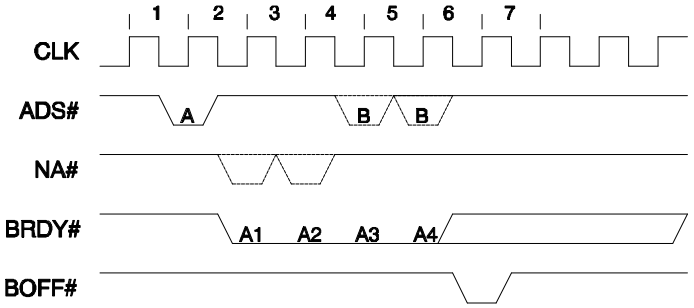
CASE 1:

- The processor issues read cycle A which may be pipelined into an earlier (not shown) cycle.
- Cycle A will be completed by the system as a 2-1-1-1 (zero wait state) cycle.
- One clock after issuance of the ADS#, NA# is asserted by the system.
- A pipelined cycle (B) is asserted by the MRM in clock 4.
- Cycle A completes on the bus in clock 5.
- One clock later, PHITM# is issued by the LRM.



CASE 2:

- The processor issues read cycle A which may be pipelined into an earlier (not shown) cycle.
- Cycle A will be completed by the system as a 2-1-1-1 (zero wait state) cycle.
- One or two clocks after issuance of the ADS#, NA# is asserted by the system.
- A pipelined cycle (B) is asserted by the MRM in either clock 4 or clock 5.
- Cycle A completes on the bus in clock 5.
- One clock later, BOFF# is issued by the system.



Specifically given the above conditions, the error condition occurs internally in the CPU due to the assertion of PHITM# or BOFF# one clock after the final BRDY# of cycle A. If this occurs, the CPU will respond to the PHITM# or BOFF# by subsequently re-issuing both cycles A and B, where expected operation would be that only cycle B is re-issued.

IMPLICATION: The code fetch or data read cycle (A) will be re-issued even though it is already completed. The second occurrence of the cycle, even if harmless from a system point of view, will confuse the internal state of the CPU and may cause subsequent CPU mis-operation.

WORKAROUND: Any of three workarounds will avoid this erratum:

1. Avoid assertion of BOFF# or PHITM# during the sensitive clock (clock 6). Avoiding assertion of PHITM# in clock 6 is guaranteed by asserting NA# no earlier than clock 3. Since ADS# occurs two or more clocks after NA# and since PHITM# occurs 2 clocks after ADS#, assertion of NA# in clock 3 or after will ensure that PHITM# is not driven active in clock 6.
2. Disable pipelining by not asserting NA#.
3. Do not perform zero wait-state read cycles.

Designs based on the 82430NX PCIset and other chip sets which do not perform zero wait-state read cycles are therefore not affected.

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

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

PROBLEM: When operating in dual processing mode, a read or prefetch cycle from either CPU may invalidate a cache line in the other.

In a dual processor environment, the LRM performs an internal snoop for each memory cycle the MRM drives onto the bus. If this snoop results in a hit in the either the code or data cache and the INV (Invalidate) signal is asserted by the system, the LRM will assert either the PHIT# or PHITM# signal and invalidate the snooped line. Expected operation is that the INV signal is not meaningful and that the LRM should respond only by asserting the PHIT# or PHITM# signal.

IMPLICATION: Unnecessary and unexpected invalidations in the LRM's caches will result. If this occurs, the only impact is to system performance; no functional problems occur with this erratum. Unnecessary invalidations in the LRM L1 caches will possibly decrease subsequent hit rate. The amount of performance degradation is a function of how many lines are shared between the two processors.

WORKAROUND: After external snoops have completed, the system should deassert the INV signal so that no invalidations are performed on subsequent private snoop operations. Designs based on the 82430NX PCIset and other chip sets which only assert INV during external snoops are therefore not affected.

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

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.

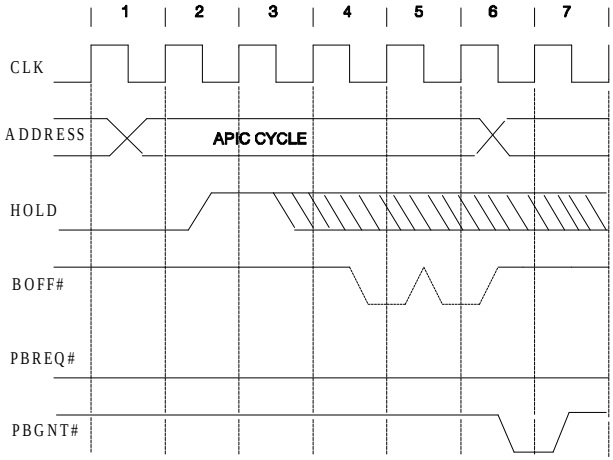
WORKAROUND: The system should not assert EADS# while the CPU owns the bus. Designs based on the 82430NX PCIset and other chip sets which do not assert EADS# while the CPU owns the bus are therefore not affected.

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

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

PROBLEM: In a dual processor system, the following sequence of events may cause the dual processing arbitration machines of both processors to lose synchronization and cause the system to hang:

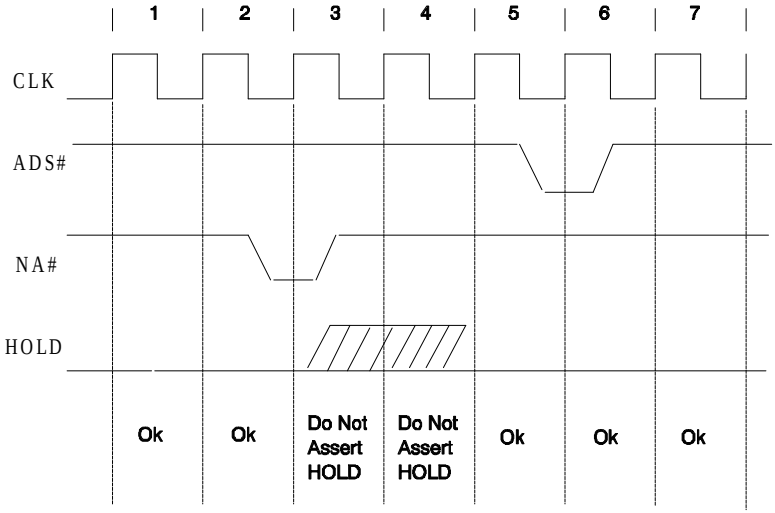
1. One of the CPUs initiates an internal APIC cycle (read or write). The LRM CPU requests ownership of the bus by asserting PBREQ#
2. HOLD is asserted during the APIC cycle
3. BOFF# is asserted before the APIC cycle gets completed, two or three clocks after HOLD is asserted (as shown in figure below)
4. Once BOFF# is deasserted, both processors may assume ownership of the bus at the same time resulting in possible contention of the CPU pins.



IMPLICATION: This problem affects dual processing (with CPU local APIC enabled) that assert HOLD and BOFF#. When the problem occurs, the dual processing arbitration machines of both processors get out of synchronization causing both processors to park on the bus. This will result in a system hang.

WORKAROUND: Use one of the following workarounds:

1. Do not use HOLD/HLDA protocol together with BOFF#. Use one or the other. Designs based on the 82430NX PCIset and other chip sets which do not use the HOLD/HLDA protocol together with BOFF# are therefore not affected.
2. In a non-pipelined system, avoid assertion of HOLD when the bus is idle. Asserting HOLD while CPU is running a bus cycle (between ADS# and last BRDY#) will ensure that HOLD does not hit an APIC cycle. Alternatively, if HOLD is asserted when the bus is idle, avoid asserting BOFF# two or three clocks after HOLD is asserted (as shown in figure above.)
3. In a pipelined system, use the same workaround as described in #2 with an additional requirement. Since an APIC cycle can be pipelined into another bus cycle, avoid assertion of HOLD in the clocks between NA# and the next ADS# (as shown in figure below.)

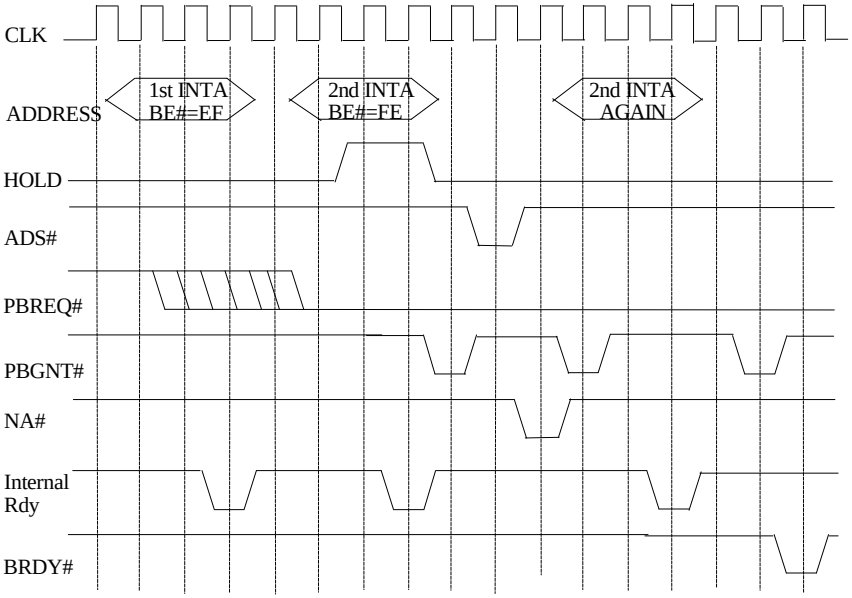


STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

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

PROBLEM: In a dual processor system, the following sequence of events may cause the system to lock up:

1. One of the CPU's (i.e. CPU A) initiates the first and second internal APIC INTA cycles. The INTA cycles are not driven on the external bus. CPU B requests ownership of the bus by asserting PBREQ#.
2. HOLD is asserted during the 2nd INTA cycle and PBGNT# is asserted.
3. CPU B issues a read or write cycle and receives NA# before completion of the cycle. The bus is granted back to CPU A.
4. CPU A reissues the second INTA cycle and grants the bus to CPU B.
5. CPU A hangs because the internal Rdy# and BRDY# were recognized out of order.



IMPLICATION: Asserting HOLD during the second INTA cycle causes the processors to lose synchronization. The external cycle completes on the bus after the INTA cycle completes which hangs the system.

WORKAROUND: Use one of the following workarounds:

1. Do not use HOLD/HLDA protocol. Designs based on the 82430NX PCIset and other chip sets which do not use the HOLD/HLDA protocol are therefore not affected.
2. Disable pipelining.

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

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 incorrectly invalidated, which in turn will cause unnecessary write back cycles, resulting in sub-optimal performance if the system uses non invalidating external snoops frequently. The degree of sub-optimal performance 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 at the beginning of this section.

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

PROBLEM: The *Pentium Processor Family Developer's Manual*, Volume 1, Section 30.5.2.2, as modified by specification change # 19, 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, and specification clarification # 19 says 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.

IMPLICATIONS: 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 at the beginning of this section.

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

PROBLEM: The *Pentium Processor Family Developer's Manual,* Volume 1, Section 20.2.1.2, 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 at the beginning of this section.

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

PROBLEM: If an APIC Remote Read (RR) transmission suffers checksum error, the RR bits of the register are mistakenly set to valid when they should show an invalid message state.

IMPLICATION: The implication is that the data portion (cycles 21-36) of the remote read message could be corrupted, but the RR status bits (bits [17:16] of the ICR0 register) would show a valid status of '10', when they should show an invalid status of '00'.

WORKAROUND: There is no workaround for this erratum, but checksum errors on the APIC bus imply that there are more serious noise issues inherent to the system that need to be addressed. In any event the RR messages should not be used if there are noise issues on the bus.

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

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

PROBLEM: A normal read of the APIC Error register clears the register. The clearing process waits 3 clocks to complete due to the possibility of being backed off. In the mean time if another error is written during this 3 clock delay, this new error overwrites the originally read error, and then is cleared at the end of the original 3 clock period.

IMPLICATION: An error could be posted in the APIC Error register but cleared prior to being read.

WORKAROUND: None identified at this time.

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

3AP. Writes to Error Register Clears Register

PROBLEM: The APIC Error register is intended to only be read. If there is a write to this register the data in the APIC Error register will be cleared and lost.

IMPLICATION: There is a possibility of clearing the Error register status since the write to the register is not specifically blocked.

WORKAROUND: Writes should not occur to the Pentium processor APIC Error register.

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

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

PROBLEM: If three interrupts of the same priority level (priority is defined in the 4MSB of the interrupt vector), arrive in the following circumstance:

1. A interrupt is being serviced by the CPU, and the proper bit is set in the ISR register.
2. A second interrupt is received from the serial bus.
3. At the same time a third interrupt is received from a local interrupt source, which could include local pins (LVT), an APIC timer (Timer), self-interrupt, or an APIC error interrupt.

If the first two conditions are met the third interrupt will be lost, and not serviced.

IMPLICATION: The third interrupt will be ignored and not serviced if the specific scenario happens as listed above.

WORKAROUND: The problem can be avoided if different priority levels are assigned to serial interrupts, than to local interrupts.

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

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

PROBLEM: This error only occurs when a Remote Read message request is processed, and the returned data has a non-zero value in the bits [30:29], and this returned data suffers a checksum (CS) error in the transmission. When the device that generated the Remote Read responds with the end of interrupt message (EOI) or the ICR message the APIC bus will lose synchronization.

IMPLICATION: If this rare condition occurs the APIC bus will become unusable, and will impact system operation. The system will hang because there will be no service on interrupts. Since RR messages are primarily used in system debug procedures, there is no impact foreseen on normal APIC or system operation.

WORKAROUND: There is no known workaround for this erratum; Remote Read messages should not be used. This error is mainly caused by checksum errors on the APIC bus which means that there are more serious noise issues inherent to the system that need to be fixed.

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

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

PROBLEM: If the processor is reading one of its local APIC registers when the HOLD pin is asserted, PCHK# may be asserted for one clock even though there is no data parity error. PCHK# will be asserted if the values of the data bits [D31:0] and the parity bits [DP3:DP0] do not match during the HOLD/HLDA transaction.

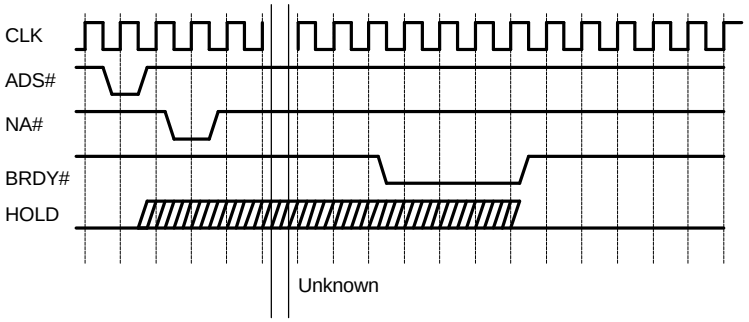
IMPLICATION: This will impact a system that implements or responds to parity checking by the CPU, the response will be specific to the parity error recovery routines implemented in the system. If parity is not implemented in a system, there will be NO adverse effect from this erratum since there is no real parity problem.

WORKAROUND: If data parity checking and the local APIC are both enabled, deassert PEN# (parity enable) during the time that HOLD is active. This signals to the processor that parity is not being driven from the system and PCHK# will never be driven in response to this data transfer.

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

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

PROBLEM: This is an APIC related dual processor erratum: When an APIC read cycle is interprocessor pipelined into any other allowable cycle, and HOLD is prior to the last BRDY# of the outstanding cycle, the MRM processor will hang.



IMPLICATION: A system that uses dual processor with pipelining enabled, is subject to periodic lockups if HOLD/HLDA# protocol is used.

WORKAROUND: Use one of the following:

1. Disable pipelining in dual processor operation.
2. Do not use the HOLD/HLDA# protocol, use BOFF# instead.

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

8AP. PICCLK Reflection May Cause an APIC Checksum Error

PROBLEM: This is an APIC-related erratum: Even though the PICCLK signal is a slower frequency clock, it has been found to be extremely sensitive to the signal transition speeds and slopes. If the PICCLK specification for rise time is met but there is a “knee” or a reflection during a high or low going transition between 0.8V to 2.0V then the CPU may not correctly receive the APIC message, and will generate a checksum error and in addition it will try to resend the APIC message. This knee could be as small as 100-200ps, and it may still cause a problem. If the reflection occurs regularly, the resend tries on the bus could saturate the bus bandwidth and the system could lose interrupts or hang up.

IMPLICATION: Checksums happen occasionally, but if there is a knee in the PICCLK transition range then there is a much higher likelihood of the occurrence. A healthy system will only see one or two per day of operation, if this problem shows up then there is a chance that the resends of the checksum errors will saturate the APIC bus and hang the system.

WORKAROUND: Use the Intel Diagnostic tool under Microsoft Windows NT 3.1 to count the number of checksum errors that occur. This tool is available to OEMs through Intel, and only works with the latest release of the Windows NT 1.1 HAL. If you are an OEM, contact your Intel representative to get a copy.

Verify that the PICCLK signal meets the new .15ns (min), 2.5ns (max), specification for a rise from 0.8V to 2.0V or a fall between 2.0V and 0.8V. Also verify that this signal is “clean” , and there are no chances or evidence of reflection during this time. The reflection would show up as ledges in the transition of the signal in the 0.8V to 2.0V transition range. If there is any evidence of a reflection and the system shows errors on the diagnostic tool, then the PICCLK line must be reworked to clean this up. The rework could include a different clock driver, and/or rerouting the clock lines on the board. See the Specification Clarification that is part of this document for guidance on PICCLK routing. See also the Specification Change section of this document for more details on the PICCLK specification.

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

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.

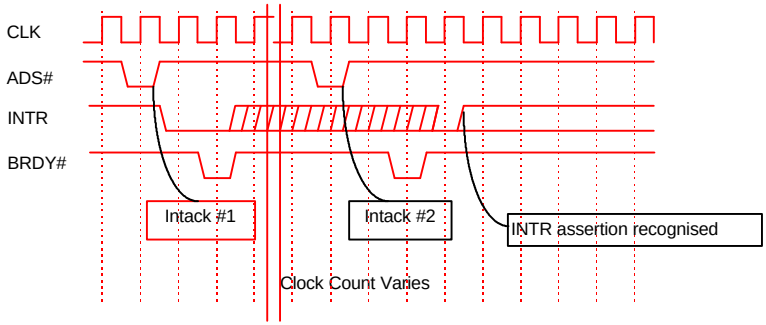
   By Hardware: By deasserting the APICEN pin prior to the falling edge of reset.

   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 at the beginning of this section.

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

PROBLEM: This erratum affects APIC in through Local (virtual wire) mode. If a uniprocessing or dual processing system is using a Pentium processor with the APIC in through Local (virtual wire) mode, and the chip set is able to re-assert the INTR line prior to completion of the second Interrupt acknowledge cycle (from a prior assertion of the INTR line), then the processor will neither recognize nor service the second interrupt. The assertion edge of INTR has to occur after the completion of the second IntAck BRDY#, if there is a transition and this transition is held high during the restricted time period, this INTR will not be recognized. Software disabling of the APIC by clearing bit 8 of the SVR (spurious vector interrupt register) will not prevent this from occurring.



IMPLICATION: If the conditions listed above are met the system may hang up since there would be an interrupt that would not get serviced.

WORKAROUND: Use one of the following:

1. Verify/Modify chip sets such that they cannot assert a second INTR for processing prior to the completion of both Interrupt acknowledge cycles for the first INTR.
2. Disable APIC instead of running in through Local mode.

   By Hardware: This can be done through hardware by deasserting the APICEN pin prior to the falling edge of reset.

   By Software: This can be done on the B-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 the B-steppings (B1, B3, or B5) of the 75-, 90-, and 100-MHz Pentium processors and the function of this bit may change in future steppings. When implementing this workaround ensure the BIOS does a CPUID check looking for a specific B-stepping of the device. CPUIDs for the following components are B1 = 0521H, B3= 0522H and B5= 0524H. If the TR12 register is used 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.

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

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

PROBLEM: If the processor is writing to one of its local APIC registers when the HOLD or BOFF# pin is asserted, then the next assertion of HOLD or BOFF# can potentially cause a subsequent APIC write cycle to be lost. This may occur as a result of the following sequence of events:

1. The CPU issues a write cycle to one of its local APIC registers (this cycles runs internally to the CPU but the corresponding APIC address is observed on the address bus while the cycle is executing)
2. HOLD or BOFF# is asserted while this APIC cycle is executing. (The write cycle to the local APIC register does complete but internal logic remembers to restart this cycle. For HOLD case, the cycle restarts upon deassertion of HOLD)
3. The processor asserts HLDA (if HOLD is asserted)
4. HOLD or BOFF# is deasserted
5. The processor deasserts HLDA (if HOLD is asserted)
6. The first APIC write cycle appears to restart

7. Another HOLD or BOFF# is asserted while this restarted APIC write cycle is executing internally
8. The processor asserts HLDA (if HOLD is asserted)
9. HOLD or BOFF# is deasserted
10. The processor deasserts HLDA (if HOLD is asserted)
11. The CPU issues the next APIC write cycle to one of its local APIC registers and there is no bus activity prior to this cycle and the previous restarted APIC write cycle.
12. This subsequent APIC write cycle is observed to start (the correct address is observed on the bus), however it fails to complete internally. In other words, from a software perspective this APIC write instruction is lost.

IMPLICATION: This problem affects systems that use HOLD/HLDA or BOFF# and enable the local APIC of the CPU. If the second APIC write cycle is an EOI (End of Interrupt) cycle, the CPU will stop servicing subsequent interrupts of equal or less priority. This may cause the system to hang. If the second APIC write cycle is not an EOI, the failure mode would depend on the particular APIC register that is not updated correctly.

WORKAROUND: Using one of the following workarounds will avoid this erratum:

1. This problem will not occur if an instruction in between the two APIC write commands in the code results in a bus cycle. This may also be achieved by inserting an APIC read instruction (reading one of the local APIC registers) before every APIC write instruction. Other instructions such as I/O or locked instructions would also force bus activity prior to executing an APIC write and will avoid this erratum.
2. Disable the local APIC if running in Uniprocessor mode.
3. In Dual processor mode, delay the next assertion of HOLD or BOFF# to allow the restarted APIC write cycle to complete.

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

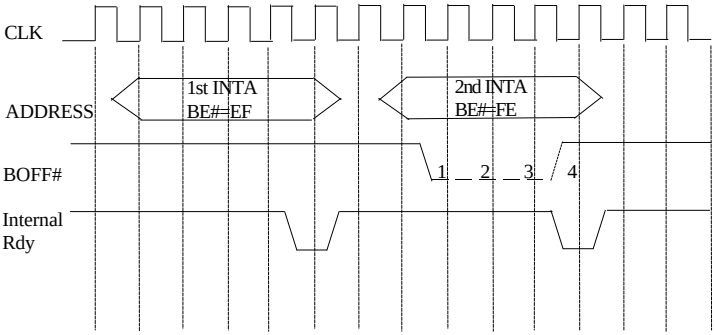
12AP. System May Hang When BOFF# Is Asserted During the Second Internal APIC INTA Cycle

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.

BOFF# During 2nd INTA (Non-remote Read)



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 at the beginning of this section.

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 at the beginning of this section.

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 at the beginning of this section.

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 at the beginning of this section.

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

PROBLEM: The *Pentium Processor Family Developer's Manual,* Volume 1 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 at the beginning of this section.

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 at the beginning of this section.

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 (1.2µS – 10µS 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 dual-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 at the beginning of this section.

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

PROBLEM: The functional redundancy master/checker (FRCMC#) input is sampled by the processor during reset (it is ignored after reset). If it is sampled active, then it tri-states the outputs. In the TCP package, this input is not bonded out, and is therefore floating internally. The possibility exists that the processor will sample this input low during reset and tri-state the outputs.

IMPLICATION: The system may fail to boot up.

WORKAROUND: If CPU fails to reset, reboot the system.

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

2TCP. BRDY# Does Not Have Buffer Selection Capability

PROBLEM: The capability of configuring selectable buffer sizes via the BRDY# and BUSCHK# pins is not available in the TCP package; only the Typical Stand Alone Component strength is available.

IMPLICATION: Only the Typical Stand Alone Component buffer size (the smallest, # EB2) is available in the TCP package. This erratum is expected to impact few if any notebook designs; the higher buffer strengths are helpful only for large designs, which are normally built into server and some desktop systems.

WORKAROUND: None identified at this time.

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

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:

1. After any assertion of RESET.

   or

2. 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. The algorithm will work with future processors too.

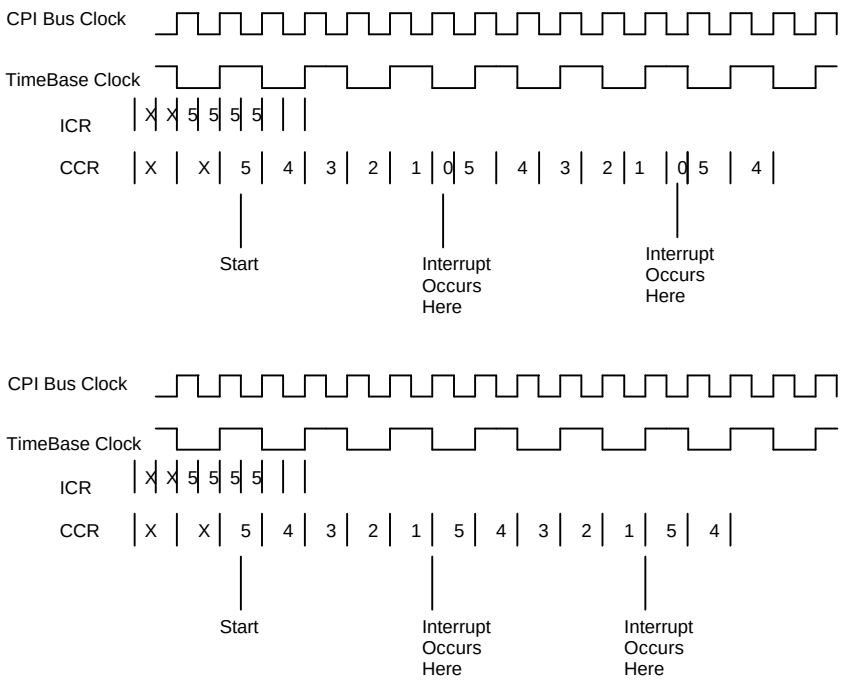
```
BSP sends AP an INIT IPI
BSP DELAYs (10mSec)
If (APIC VERSION is not an 82489DX)
{
    BSP sends AP a STARTUP IPI
    BSP DELAYs (200uSec)
    BSP sends AP a STARTUP IPI
    BSP DELAYs (200uSec)
}
BSP verifies synchronization with executing AP
```

For additional information please refer to the Intel Multiprocessor Specification Version 1.4 (Order Number 242016-004).

2. APIC Timer Use Clarification

The APIC Timer functions correctly, but there is one timer "tick" that is a different pulse width than the other timer "ticks". The countdown performs correctly regardless of the divisor that is programmed, each of these ticks is of the same pulse width, but the last tick is held for just one single timer clock regardless of the divisor programmed. This results in a slightly inaccurate timer. This last tick pulse width is shown in the top diagram. The lower diagram shows how the accuracy of the timer will be corrected in a future stepping. This phenomenon occurs when the timer is used as a single shot generator as well.

NOTE: The CPU Bus clock in which the CCR is loaded from the ICR occurs one clock after the ICR is loaded.



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

Many of the APIC errors that are listed in the erratum concern checksum errors on the APIC bus. This specification clarification is to address the elimination of checksum errors on the APIC bus. Doing so would reduce the error rate and would eliminate the possibility of dropping of Interprocessor Interrupts (IPI) due to multiple data errors on the same APIC message. Getting a single checksum error does not typically pose a problem, because the error will cause the APIC message will be resent, but if there is a high error rate then there is a possibility that the retries of the APIC messages may take up the bandwidth of the APIC bus some may be rescheduled or accidentally dropped. A system that performs in this manner has a fundamental design problem that needs to be corrected.

Most of the checksum errors observed are a result of the PICCLK not crossing the threshold cleanly, this can be tracked down to 2 possible issues: PICCLK marginally meeting rise/fall time specification, and reflections causing PICCLK to re-cross the threshold. Both of the problems are fairly easily solved, and a robust system design will typically show zero checksum errors in a 24 hour period during stress testing.

The current specification for the rise/fall time of the PICCLK signal is shown in the timing tables as t_{60e} and t_{60f} for all frequencies. This rise/fall time must be met to guarantee correct operation of the device. This rise/fall time should be verified at all receivers of the PICCLK signal. If a daisy chain type route is used with a large series termination resistor the rise/fall time at the devices near the driver end of the net are the most critical and should be checked. The high and low time specifications also need to be verified to meet the specification.

There is also a good chance that in a system using daisy chain route topologies that there will be reflections seen by receivers located close to the driver. It would be recommended to use a balanced star type route on clock signals like PICCLK to ensure there are no reflections that may re-cross any trip thresholds on the inputs to the CPU. The correct balanced route is based on both the length of the traces and the relative input capacitance loading presented by each device. It is also important to verify the selection of the correct series terminating resistors to dampen out any reflections on this line.

The values of the series resistances should be chosen using the following guidelines:

Single Receiver:

1. Driver and receiver on opposite ends of the trace.
2. R_driver + R_terminator = Zo.

Multiple Receivers:

1. Trace has 1 branch per receiver, branches are of equal equivalent capacitive loads.
2. Branch as close as possible to the Driver.
3. For single termination resistor (n branches):
   a. Place terminator as close to driver as possible.
   b. R_driver + R_terminator = Zo / n
4. For multiple termination resistors (n branches):
   a. Place terminators on each branch, as close to the branch point as possible.
   b. R_driver + R_terminator / n = Zo / n

Even though PICCLK is a lower frequency clock this clock is still critical and should be routed with care and reflections at each node should be eliminated. More detailed clock routing techniques are available in the *Pentium™ Processor Clock Design* application note (AP-479), (Order Number 241574).

4. Boundary Scan RUNBIST Register Requires Initialization Prior to Use

It has been found that the Reset cell of the Boundary Scan register is not correctly initialized prior to use. There is a failing result reported from running the RUNBIST Command through the Boundary Scan circuitry.

The IEEE 1149.1-1990 specification states, “Where a test data register (other than the Boundary Scan register) must be initialized prior to execution of the self-test this must occur at the start of the self-test without any requirement to shift data into the component.”

To execute the TAP RUNBIST instruction:

1. Select "Sample/Preload” TAP instruction (XXXXXXXXX0001) and load the RESET BSCAN cell (cell #52) with ‘0’.
2. Shift in the "Runbist" TAP instruction move to and wait in the "run-test-idle" state for 2^{19} clocks.
3. Examine the pass/fail status by advancing to the "shift-dr" state to read the runbist register.

5. Pullup/Pulldown Resistor Values for Some Configuration Pins

Several inputs are sampled during RESET to optionally invoke features of the processor. These pins include BE[0:3]#, BF[0:1], CPUTYP, BUSCHK#, BRDYC#, APICEN, DPEN#, and FRCMC#.

The following should be added to the *Pentium Processor Family Developer's Manual*, Volume 1, Section 21.1.6: BE[0:3]# pins establish the APIC ID for the processor and are input/output pins. These pins have strong internal pull down resistors and typically high external capacitive loading. A strong pullup on BE[0:3]# is needed to make sure that the pins reach the correct value. In addition, since these pins are also outputs, a large resistive load would degrade the signal output during normal operation. A 50 Ohm tristate driver is recommended to drive these pins during RESET only.

The *Pentium Processor Family Developer's Manual*, Volume 1, Section 21.1.7 and Section 21.1.11 states that the BF[0:1] and CPUTYP pins must be tied to either the power or ground plane depending on the processor features desired. Since some manufacturing test systems would show a short to power or ground on that net, it is common practice to put either a pullup or pulldown resistor on a net. Because these pins have internal pullup/pulldown resistors, they can be left floating when the default value is desired. In order to override the strong internal pulldown of CPUTYP, an external pullup of 100 ohms or less must be used. In the case of BF[0:1], external pulldowns of 500 ohms or less must be used between the pins and ground to effectively override the default. For maximum flexibility, external pullups of 2.2K Ohms or less should be used on BF[0:1] to override the internal default of future Pentium processors.

In addition to their normal functionality, BUSCHK# and BRDYC# are sampled at RESET to select the buffer strength for some pins. They have internal pullup resistors. To override the default settings for the buffer strengths, these pins should be driven and not permanently strapped to ground since this will interfere with the normal operation of these pins. The driver should have an internal resistance of 1K Ohms or less. This should be noted in the *Pentium Processor Family Developer's Manual*, Volume 1, Section 24.1.

The APICEN/PICD1 has an internal pulldown resistor and is sampled at the falling edge of RESET. When using an active circuit to override the internal pulldown resistor, the driver should have an internal effective pullup resistance of 1K Ohms or less. This should be noted in the *Pentium Processor Family Developer's Manual*, Volume 1, Sections 21.1.5 and 21.1.32.

Similarly, the DPEN#/PICD0 pin has an internal pullup resistor. When using an active circuit to override the internal pullup, the driver should have an internal effective pulldown resistance of 1K Ohms or less. This should be noted in the *Pentium Processor Family Developer's Manual*, Volume 1, Section 21.1.32.

Finally, FRCMC# is also sampled at RESET to select functional redundancy master/checker configurations. There is an internal pullup resistor on all cB1 step parts. The pin can be left floating, however to override the default setting a 100 Ohm pulldown should be used on cB1 step parts. On all other parts, however, there is no internal default. Therefore, the pin must be driven and the 100 Ohm restriction does not apply.

6. 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 2nd 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.

7. The Stop Clock State Is Not Supported in FRC Mode

In order to clearly document that the Stop Clock state can't be used in the FRC mode, the WARNING at the end of Section 30.5.4.6 in Volume 1 of the *Pentium Processor Family Developer's Manual* will be expanded to include FRC: "The Stop Clock state feature cannot be used in dual processing or functional redundancy checking modes because there is no way to re-synchronize the internal clocks of the two processors." The same sentence will be added as the next to the last paragraph in Section 12.2.5.

8. Nominal V_{CC} for V_{IH3}

In the *Pentium Processor Family Developer's Manual,* Volume 1, Chapter 23, Table 23-2 3.3V D.C Specifications note 5 should be added for V_{IH3} as follows.

Symbol	Parameter	Min	Max	Unit	Notes
V_{IH3}	Input High Voltage	2.0	V_{CC} + 0.3	V	TTL Level (3) (5)

5. This parameter is measured at nominal V_{CC}, which is 3.515 V.

9. Branch Trace Message Discrepancies

The *Pentium Processor Family Developer's Manual,* Volume 1, Section 29.1 states that when normal branch trace messages are enabled, two cycles are produced. The address field of the first cycle contains the branch target linear address and the address field of the second cycle contains the linear address of the instruction causing the branch. The following clarifications should be added:

The second message will always contain the linear address of the instruction executed in the u pipe even if the instruction that caused the branch was executed in the v pipe.

For serializing instructions and segment descriptor loads the address field of the first cycle will contain the address of the next sequential instruction after the instruction that caused the BTM.

When switching between the normal and fast formats by using the WRMSR instruction to change bit 8 of TR12, the WRMSR instruction causes a branch trace message when they are enabled. The format for this branch trace message will be the format that was programmed **before** the WRMSR instruction was executed.

Also note that the WRMSR instruction to enable branch trace messages will cause a BTM to be generated (WRMSR is a serializing instruction and serializing instructions cause BTMs). A WRMSR to disable BTMs will not generate a BTM.

Section 29.1 also states that the Pentium processor will treat segment descriptor loads as causing taken branches. This should state that **some** segment descriptor loads will cause taken branches.

It should also be noted in Section 29.1 that instructions which cause masked floating point exceptions may cause one or more branch trace special cycles. This is because execution of an instruction may be aborted and restarted several times due to the exception.

Chapter 13 states that conditions which cause the VERR, VERW, LAR and LSL instruction to clear the ZF bit in EFLAGS will also cause these instructions to be treated as taken branches. However, if these instructions fail the protection checks, no branch trace message will be generated.

10. 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.

11. SMIACT# Handling During Snoop Writeback

In Section 14.2.1 in the *Pentium Processor Family Developer's Manual*, Volume 1, the following text should be added as item 5 in the SRAM Interface Section.

Inquire cycles are permitted during SMM, but it is the responsibility of the system to ensure that any snoop writeback completes to the correct memory space, irrespective of the state of the SMIACT# pin. Specifically, if SMM is overlaid, and SMM space is non cacheable, then any snoop writeback cycle occurring during SMM must complete to system memory, even though SMIACT# will remain active.

If an inquire cycle occurs after assertion of SMI# to the processor, but before SMIACT# is returned, note that SMIACT# could be returned at any point during the snoop writeback cycle. Depending on the timing of SMI# and the inquire cycle, SMIACT# could change states during the writeback cycle. Again, it is the responsibility of the system, if it supports snooping during SMM, to ensure that the snoop writeback cycle completes to the correct memory space, irrespective of the state of the SMIACT# pin.

12. Package Thermal Specifications

The Pentium Processor is available in two types of SPGA packages, with and without the heatspreader. The package change is described in Specification Change 11. Both packages have the following thermal specifcations:

Max T_{CASE} = 70°C.

The packages also have the same parasitics, ESDI, and EMI characteristics.

However, thermal resistance of the packages are different. See the tables below.

Staggered Ceramic Pin Grid Array with CuW Heat Spreader (CPGA)
Θ_{JA} [°C/W] for different heat sink heights and air flow rates

	Θ_{JA} [°C/W] vs Airflow [LFM]					
	0	**100**	**200**	**400**	**600**	**800**
With 0.25" Heat Sink	9.5	8.4	7.0	4.8	4.0	3.4
With 0.35" Heat Sink	9.2	7.9	6.4	4.4	3.7	3.2
With 0.45" Heat Sink	8.8	7.4	5.7	4.0	3.3	2.9
With 0.55" Heat Sink	8.5	6.9	5.1	3.6	3.0	2.7
With 0.65" Heat Sink	8.1	6.4	4.7	3.4	2.8	2.5
With 0.80" Heat Sink	7.4	5.7	4.3	3.0	2.6	2.4
With 1.00" Heat Sink	6.7	5.0	4.0	3.0	2.5	2.2
With 1.20" Heat Sink	6.3	4.7	3.7	2.8	2.4	2.2
With 1.40" Heat Sink	5.8	4.3	3.4	2.6	2.3	2.1
With 1.50" Heat Sink	5.6	4.2	3.2	2.5	2.3	2.1
Without Heat Sink	12.7	11.8	10.0	7.0	5.8	5.1

Staggered Ceramic Pin Grid Array with CuW Heat Spreader (CPGA)
Θ_{CA} [°C/W] for different heat sink heights and air flow rates

	Θ_{CA} [°C/W] vs Airflow [LFM]					
	0	**100**	**200**	**400**	**600**	**800**
With 0.25" Heat Sink	8.7	7.6	6.2	4.0	3.2	2.6
With 0.35" Heat Sink	8.4	7.1	5.6	3.6	2.9	2.4
With 0.45" Heat Sink	8.0	6.6	4.9	3.2	2.5	2.1
With 0.55" Heat Sink	7.7	6.1	4.3	2.8	2.2	1.9
With 0.65" Heat Sink	7.3	5.6	3.9	2.6	2.0	1.7
With 0.80" Heat Sink	6.6	4.9	3.5	2.2	1.8	1.6
With 1.00" Heat Sink	5.9	4.2	3.2	2.2	1.7	1.4
With 1.20" Heat Sink	5.5	3.9	2.9	2.0	1.6	1.4
With 1.40" Heat Sink	5.0	3.5	2.6	1.8	1.5	1.3
With 1.50" Heat Sink	4.8	3.4	2.4	1.7	1.5	1.3
Without Heat Sink	11.4	10.5	8.7	5.7	4.5	3.8

Staggered Ceramic Pin Grid Array with CuW Heat Spreader (CPGA)
Θ_{JC} [°C/W] for a package with and without a heat sink

	No Heat Sink	With Heat Sink
Average Θ_{JC}	1.3	0.8

Staggered Ceramic Pin Grid Array without Spreader (CPGA)
Θ_{JA} [°C/W] for different heat sink heights and air flow rates

	Θ_{JA} [°C/W] vs Airflow [LFM]					
	0	100	200	400	600	800
With 0.25" Heat Sink	9.9	8.8	7.4	5.2	4.4	3.8
With 0.35" Heat Sink	9.6	8.3	6.8	4.8	4.1	3.6
With 0.45" Heat Sink	9.2	7.8	6.1	4.4	3.7	3.3
With 0.55" Heat Sink	8.9	7.3	5.5	4.0	3.4	3.1
With 0.65" Heat Sink	8.5	6.8	5.1	3.8	3.2	2.9
With 0.80" Heat Sink	7.8	6.1	4.7	3.4	3.0	2.8
With 1.00" Heat Sink	7.1	5.4	4.4	3.4	2.9	2.6
With 1.20" Heat Sink	6.7	5.1	4.1	3.2	2.8	2.6
With 1.40" Heat Sink	6.2	4.7	3.8	3.0	2.7	2.5
With 1.50" Heat Sink	6.0	4.6	3.6	2.9	2.7	2.5
Without Heat Sink	15.7	14.4	13.0	10.1	8.7	7.8

Staggered Ceramic Pin Grid Array without Spreader (CPGA)
Θ_{CA} [°C/W] for different heat sink heights and air flow rates

	Θ_{CA} [°C/W] vs Airflow [LFM]					
	0	100	200	400	600	800
With 0.25" Heat Sink	9.1	8.0	6.6	4.4	3.6	3.0
With 0.35" Heat Sink	8.8	7.5	6.0	4.0	3.3	2.8
With 0.45" Heat Sink	8.4	7.0	5.3	3.6	2.9	2.5
With 0.55" Heat Sink	8.1	6.5	4.7	3.2	2.6	2.3
With 0.65" Heat Sink	7.7	6.0	4.3	3.0	2.4	2.1
With 0.80" Heat Sink	7.0	5.3	3.9	2.6	2.2	2.0
With 1.00" Heat Sink	6.3	4.6	3.6	2.6	2.1	1.8
With 1.20" Heat Sink	5.9	4.3	3.3	2.4	2.0	1.8
With 1.40" Heat Sink	5.4	3.9	3.0	2.2	1.9	1.7
With 1.50" Heat Sink	5.2	3.8	2.8	2.1	1.9	1.7
Without Heat Sink	14.4	13.1	11.7	8.8	7.4	6.5

Staggered Ceramic Pin Grid Array without Spreader (CPGA)
Θ_{JC} [°C/W] for a package with and without a heat sink

	No Heat Sink	With Heat Sink
Average Θ_{JC}	1.3	0.8

13. 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.

14. EADS# Recognition

The following text replaces the "When Sampled" section of the *Pentium Processor Family Developer's Manual*, Volume 1, section 5.1.19.

When sampled

To guarantee recognition, EADS# should be asserted two clocks after an assertion of AHOLD or BOFF#, or one clock after an assertion of HLDA. In addition, the Pentium processor will ignore an assertion of EADS# if the processor is driving the address bus, or if HITM# is active, or in the clock after ADS# or EADS# is asserted.

15. Event Monitor Counters

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-25, the last sentence of the FLOPs event description should be changed as follows: The integer multiply instructions and other instructions which use the FP arithmetic circuitry will be counted.

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-25, it states that the performance monitor counters can be programmed to count the number of breakpoint matches on DR0, DR1, DR2 or DR3 registers and the count will be incremented in the event of a breakpoint match whether or not breakpoints are enabled. It should be noted that when breakpoints are not enabled, for code breakpoints the counter will only be incremented if a code breakpoint match was on an instruction that was executed in the u pipe. Code breakpoint matches are not checked for instructions executed in the v pipe when breakpoints are not enabled and thus will not cause the counter to increment.

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-24, the "Instructions Executed" event counter (event 010110) is used to count the total number of instructions executed. It should be noted that for any Repeat prefixed string instruction (ex., REP MOVS, REPNE SCAS) the "Instructions Executed" counter will only increment once despite the fact that the repeat loop executes the same instruction multiple times until the loop criteria is satisfied. This applies to all the Repeat string instruction prefixes (i.e., REP, REPE, REPZ, REPNE, and REPNZ). It should also be noted that the "Instructions Executed" counter will only increment once per each HALT instruction executed regardless of how many cycles the processor remains in the HALT state. In addition, it should be noted that there are additional events which can also trigger the "Instructions Executed" (event 010110) event counter. All hardware and software interrupts and exceptions will also cause the count to be incremented.

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-24, there are additional events which trigger the "Instructions Executed in the v-pipe" (event 010111) event counter which should be noted. All hardware and software interrupts and exceptions will also cause the count to be incremented.

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-24, it should be noted that the "Pipeline Flushes" (event 010101) event counter will not be incremented for serializing instructions (serializing instructions cause the prefetch queue to be flushed but will not trigger the Pipeline Flushed event counter). Software interrupts will also not cause this event counter to be incremented since they do not flush the pipeline.

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-22, the "Data Read Miss" (event 000011) event counter counts the number of memory data reads that miss the internal data cache whether or not the access is cacheable or non-cacheable. It should be noted that additional reads to the same cache line after the first BRDY# of the burst linefill is returned but before the final (fourth) BRDY# has been returned, **will not** cause this event counter to be incremented additional times.

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-23, the description for the "Code Read" (event 001100), "Code TLB Miss" (event 001101) and "Code Cache Miss" (event 001110) event counters should say that individual eight byte non-cacheable instruction reads **are** counted.

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-23, it should be noted that the VERR an VERW instructions are treated as branches and will cause the "Branches" (event 010010) event counter to be incremented.

In Volume 1 of the *Pentium Processor Family Developer's Manual*, Section 33.4.5, page 33-23, it should be noted that the "Taken branch or BTB hit" (event 010100) event counter also is incremented for jumps, calls, returns, software interrupts, interrupt returns, serializing instructions, some segment descriptor loads, hardware interrupts, programmatic exceptions that invoke a trap or fault handler and the VERR and VERW instructions.

16. KEN# Sets Cacheability For Restarted Cycles

The *Pentium Processor Family Developer's Manual*, Volume 1, Section 5.1.36, the fourth paragraph under "Signal Description" should read as follows:

Once KEN# is sampled active for a cycle, the cacheability cannot be changed. If a cycle is **restarted for any reason** after the cacheability of the cycle has been determined, the same cacheability attribute on KEN# must be returned to the processor when the cycle is redriven."

17. NMI Signal Description

The *Pentium Processor Family Developer's Manual*, Volume 1, Section 5.1.40, the second paragraph under "Signal Description" should read as follows:

If NMI is asserted during the execution of the NMI service routine it will remain pending and will be recognized after the IRET is executed by the NMI service routine. At most, one assertion of NMI will be held pending. **If NMI is reasserted prior to the NMI service routine entry it will be ignored.**

18. BTB Behavior When Entering SMM

Section 3-2, page 3-6 of Volume 1 of the *Pentium Processor Family Developer's Manual*, states that the Pentium processor speculatively runs code fetches corresponding to instructions executed sometime in the past. Such code fetch cycles are run based on past execution history, regardless of whether the instructions retrieved are relevant to the currently executing instruction sequence.

One effect of the branch prediction mechanism is that the Pentium processor may run code fetch bus cycles to retrieve instructions which are never executed. Although the opcodes retrieved are discarded, the system must

complete the code fetch bus cycle by returning BRDY#. It is particularly important that the system return BRDY# for all code fetch cycles regardless of the address.

The following text should be added to Section 3-2 of Volume 1 of the *Pentium Processor Family Developer's Manual*, and also to Section 14.2.1 of Volume 1 as item number 5 on page 14-5.

It should also be noted that upon entering SMM, the branch target buffer (BTB) is not flushed and thus is possible to get a speculative prefetch to an address outside of SMRAM address space due to branch predictions based on code executed prior to entering SMM. If this occurs, the system must still return BRDY# for each code fetch cycle.

19. 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 NMI's could be nested in the first NMI handler. The interrupt handler should take this into consideration.

20. Exit from Shutdown

A note in the *Pentium Processor Family Developer's Manual*, Volume 1, Section 6.3.8 and Volume 3, Section 14.9.8 should be added as follows:

Upon entering shutdown, the state of the CPU is unpredictable and may or may not be recoverable. RESET or INIT should be asserted to return the system to a known state. Although some system operations (i.e. FLUSH#, SMI# and R/S#) are generally recognized during shutdown, these operations may not complete successfully in some cases once shutdown is entered.

Furthermore, upon exit from shutdown with NMI (to the NMI handler), the SS, ESP and EIP of the task that was executing when shutdown occurred can no longer be relied upon to be valid. Therefore, using NMI to exit shutdown should be used only for debugging purposes and not to resume execution from where shutdown occurred.

If invoking NMI to exit shutdown, use a task gate rather than an interrupt or trap gate in slot 2 of the IDT. One of the conditions that may lead to shutdown is an attempt to use an invalid stack segment selector (SS). In this case, if the NMI successfully exits shutdown, it will immediately re-enter shutdown because it has no valid stack on which to push the return address. It is more robust to vector NMI through a task gate rather than an interrupt gate in the IDT, since the task descriptor allocates a new stack for the NMI handler context.

21. Power Dissipation (Max) Clarification

On Page 23-6 of *Pentium Processor Family Developer's Manual*, Volume 1, Table 23-5 shows the power dissipation requirements for thermal design. Note 6 should be changed as follows: "This is determined using a worst case instruction mix with V_{CC}=3.52V. The use of V_{CC} in this measurement takes into account the thermal time constant of the package".

22. 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.

23. 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.

24. 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".

25. Supported Bus Fractions and Frequency Ranges

The following text should be added to the *Pentium Processor Family Developer's Manual,* Volume 1 in Sections 19.1.1 (immediately preceding Table 19-1) and 21.1.7 (immediately preceding the table titled "Bus Frequency Selections").

Each Pentium Processor product must be externally configured with the BF1-0 pins to operate in the specified bus fraction mode and corresponding frequency ranges. Operation out of the specifications is not supported. For example, the 150 MHz Pentium processor does not support the 1/2 or 2/3 bus fraction mode; it supports the 2/5 mode only. The 150 MHz Pentium Processor also does not operate beyond the 60 MHz bus frequency. The table below clarifies and summarizes the specifications.

The following table replaces Table 19-1 and the table titled "Bus Frequency Selections" in Section 21.1.7 in the *Pentium Processor Family Developer's Manual,* Volume 1.

BF1	BF0	Max Core/Bus Frequency (MHz)	Min Core/Bus Frequency (MHz)	Bus/Core Ratio
0	1	200/66	100/33	1/3
0	0	166/66	83/33	2/5
0	0	150/60	75/30	2/5
1	0	133/66	66/33	1/2
1	0	120/60	60/30	1/2
1	0	100/50	50/25	1/2
1	1	100/66	50/33	2/3
1	1	90/60	45/30	2/3
1	1	75/50	37.5/25	2/3

NOTE: The 100 MHz (Max Core Frequency) Pentium Processors can be operated with both 1/2 and 2/3 Bus/Core Ratios. The 100 MHz Pentium processors with Voltage Reduction Technology (mobile Pentium processor steppings prefixed with an "m") can only be operated with a 2/3 Bus/Core ratio.

26. CPUID Feature Flags

The *Pentium Processor Family Developer's Manual*, Volume 3, page 25-74 defines the feature flags returned in EDX after a CPUID instruction is executed. Some additional bits are defined as follows:

EDX[0:0] FPU on chip
EDX[1:1] Virtual Mode Extension
EDX[2:2] Debugging Extension
EDX[3:3] Page Size Extension
EDX[4:4] Time Stamp Counter
EDX[5:5] Model Specific Registers
EDX[6:6] Physical Address Extension
EDX[7:7] Machine Check Exception
EDX[8:8] CMPXCHG8 Instruction Supported
EDX[9:9] On-Chip APIC Hardware Enabled
EDX[10:11] Reserved
EDX[12:12] Memory Type Range Register
EDX[13:13] Page Global Enable
EDX[14:14] Machine Check Architecture

EDX[15:15] CMOV Instruction Supported
EDX[16:22] Reserved
EDX[23:23] MMX™ Technology Supported
EDX[24:31] Reserved

Additional information on the function of these bits can be found in Application Note AP-485, Intel Processor Identification With the CPUID Instruction order number 241618.

27. PPGA Package Thermal Specifications

The following tables list thermal resistance values θ_{ja}, θ_{ca} and θ_{jc} for the plastic pin grid array (PPGA) package. This information will be added to the *Pentium Processor Family Developer's Manual*, Volume 1, chapter 26.

Table1: θ_{ja} [°C/W] for Different Heat Sink Heights and Air Flow Rates (PPGA)

	θ_{ja} [°C/W] vs Air Flow Rate [LFM]					
Heat Sink Height	0	100	200	400	600	800
0.25"	9.5	8.4	7.0	4.8	4.0	3.4
0.35"	9.2	7.9	6.4	4.4	3.7	3.2
0.45"	8.8	7.4	5.7	4.0	3.3	2.9
0.55"	8.5	6.9	5.1	3.6	3.0	2.7
0.65"	8.1	6.4	4.7	3.4	2.8	2.5
0.80"	7.4	5.7	4.3	3.0	2.6	2.4
1.00"	6.7	5.0	4.0	3.0	2.5	2.2
1.20"	6.3	4.7	3.7	2.8	2.4	2.2
1.40"	5.8	4.3	3.4	2.6	2.3	2.1
1.50"	5.6	4.2	3.2	2.5	2.3	2.1
None	14.3	13.6	12.7	9.3	7.9	7.0

Table2: θ_{ca} [°C/W] for Different Heat Sink Heights and Air Flow Rates (PPGA)

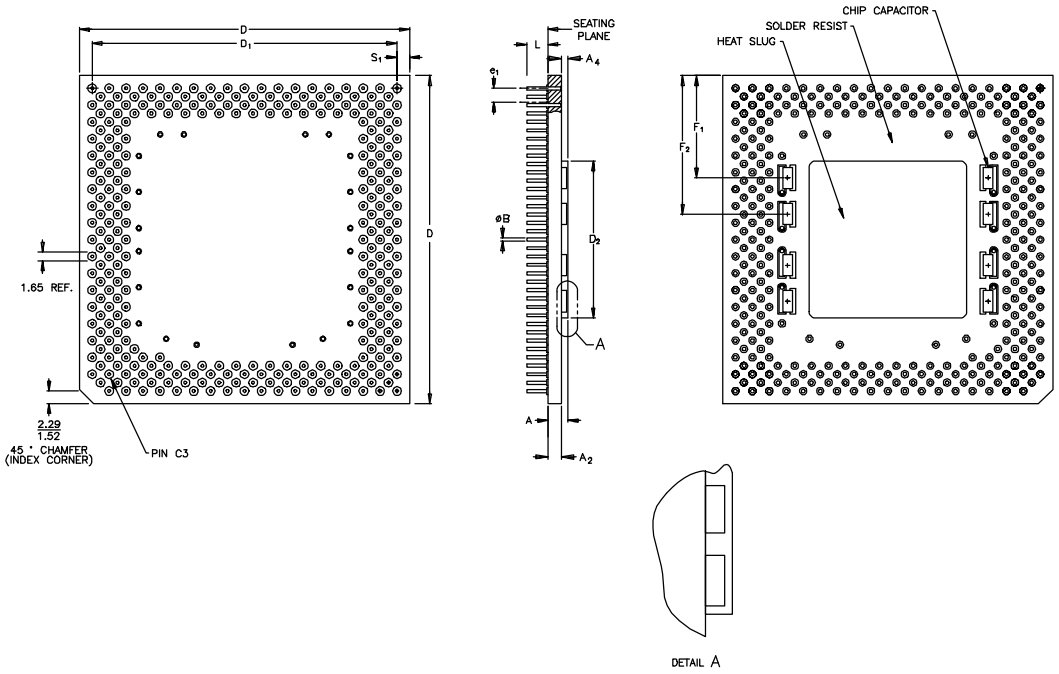
	θ_{ca} [°C/W] vs Air Flow Rate [LFM]					
Heat Sink Height	0	100	200	400	600	800
0.25"	9.0	7.9	6.5	4.3	3.5	2.9
0.35"	8.7	7.4	5.9	3.9	3.2	2.7
0.45"	8.3	6.9	5.2	3.5	2.8	2.4
0.55"	8.0	6.4	4.6	3.1	2.5	2.2
0.65"	7.6	5.9	4.2	2.9	2.3	2.0
0.80"	6.9	5.2	3.8	2.5	2.1	1.9
1.00"	6.2	4.5	3.5	2.5	2.0	1.7
1.20"	5.8	4.2	3.2	2.3	1.9	1.7
1.40"	5.3	3.8	2.9	2.1	1.8	1.6
1.50"	5.1	3.7	2.7	2.0	1.8	1.6
None	13.0	12.3	11.4	8.0	6.6	5.7

Table3: θ_{jc} [°C/W] for a PPGA Package with and without a Heat Sink

	No Heat Sink	**With Heat Sink**
Average θ_{jc}	1.3	0.5

28. PPGA Package Dimensions

Package dimensions for the Platic Pin Grid Array (PPGA) package are as follows. This information will be added to the *Pentium Processor Family Developer's Manual*, Volume 1, Chapter 25.



Family: Plastic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	2.72	3.33		0.107	0.131	
A_1	1.83	2.23		0.072	0.088	
A_2	1.00			0.039		
B	0.40	0.51		0.016	0.020	
D	49.43	49.63		1.946	1.954	
D_1	45.59	45.85		1.795	1.805	
D_2	23.44	23.95		0.923	0.943	
e_1	2.29	2.79		0.090	0.110	
F_1	17.56			0.692		
F_2	23.04			0.907		
L	3.05	3.30		0.120	0.130	
N	296			296		
S_1	1.52	2.54		0.060	0.100	
ISSUE						

NOTE:
Typical lead count: 296

29. Package Thermal Specifications

The thermal resistance of the cB1 and cC0 step parts are shown below. This information will be added to the *Pentium Processor Family Developer's Manual*, Volume 1, chapter 26.

Staggered Ceramic Pin Grid Array without Spreader (CPGA)
Θ_{JA} [°C/W] for different heat sink heights and air flow rates

	Θ_{JA} [°C/W] vs Airflow [LFM]					
	0	**100**	**200**	**400**	**600**	**800**
With 0.25" Heat Sink	10.6	9.5	8.1	5.9	5.1	4.5
With 0.35" Heat Sink	10.3	9.0	7.5	5.5	4.8	4.3
With 0.45" Heat Sink	9.9	8.5	6.8	5.1	4.4	4.0
With 0.55" Heat Sink	9.6	8.0	6.2	4.7	4.1	3.8
With 0.65" Heat Sink	9.2	7.5	5.8	4.5	3.9	3.6
With 0.80" Heat Sink	8.5	6.8	5.4	4.1	3.7	3.5
With 1.00" Heat Sink	7.8	6.1	5.1	4.1	3.6	3.3
With 1.20" Heat Sink	7.4	5.8	4.8	3.9	3.5	3.3
With 1.40" Heat Sink	6.9	5.4	4.5	3.7	3.4	3.2
With 1.50" Heat Sink	6.7	5.3	4.3	3.6	3.4	3.2
Without Heat Sink	16.2	15.5	14.3	12.2	10.3	9.2

Staggered Ceramic Pin Grid Array without Spreader (CPGA)
Θ_{CA} [°C/W] for different heat sink heights and air flow rates

	Θ_{CA} [°C/W] vs Airflow [LFM]					
	0	**100**	**200**	**400**	**600**	**800**
With 0.25" Heat Sink	9.4	8.3	6.9	4.7	3.9	3.3
With 0.35" Heat Sink	9.1	7.8	6.3	4.3	3.6	3.1
With 0.45" Heat Sink	8.7	7.3	5.6	3.9	3.2	2.8
With 0.55" Heat Sink	8.4	6.8	5.0	3.5	2.9	2.6
With 0.65" Heat Sink	8.0	6.3	4.6	3.3	2.7	2.4
With 0.80" Heat Sink	7.3	5.6	4.2	2.9	2.5	2.3
With 1.00" Heat Sink	6.6	4.9	3.9	2.9	2.4	2.1
With 1.20" Heat Sink	6.2	4.6	3.6	2.7	2.3	2.1
With 1.40" Heat Sink	5.7	4.2	3.3	2.5	2.2	2.0
With 1.50" Heat Sink	5.5	4.1	3.1	2.4	2.2	2.0
Without Heat Sink	14.5	13.8	12.6	10.5	8.6	7.5

Staggered Ceramic Pin Grid Array without Spreader (CPGA)
Θ_{JC} [°C/W] for a package with and without a heat sink

	No Heat Sink	**With Heat Sink**
Average Θ_{JC}	1.7	1.25

30. 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."

31. 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.

32. Cache Test Registers Are Modified During FLUSH#

The following Note will be added to the *Pentium Processor Family Developer's Manual,* Volume 1, Section 33.2.1.1.1 on **Direct Cache Access**:

Note: When the FLUSH# pin is asserted, it is treated as an interrupt, and when serviced at the next instruction boundary, it causes writeback of the data cache and then invalidation of the internal caches. The cache test registers TR2, TR3, TR4 and TR5 are used in this process, and thus their values after FLUSH# has been serviced are unpredictable. Therefore FLUSH# should not be asserted while code is being executed which uses these test registers.

33. 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 4K-byte page translation rather than a new 4M-byte 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.

34. PCHK# V_{OL} Level for DP Designs

The following note should be added to Table 23-2 (3.3V DC Specifications) in the *Pentium Processor Family Developer's Manual,* Volume 1.

6. In dual processing systems, up to a 10mA Ioad from the second CPU may be observed on the PCHK# signal. Based on silicon characterization data, V_{OL} of PCHK# will remain less than 400 mV even with a 10 mA Ioad. PCHK# V_{OL} will increase to approximately 500 mV with a 14mA Ioad (worst case for a DP system with a 4 mA system load).

35. 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."

36. 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.

37. 4-MByte Page Extensions

The pentium processor allows 4-MByte page extensions which are enabled by setting CR4.PSE bit (bit 4) to 1. For more information on 4-MByte pages, please refer to the Volume 3, Section 3.6 of the *Pentium Pro Family Developer’s Manual* (order number 242692).

38. External Snoop Difference Between UP and DP When L1 Cache Disabled

The following will be added to the *Pentium Processor Family Developer’s Manual,* Volume 1, Section 3.7.3 and Volume 3, Section 18.2.

When the L1 cache is disabled (CR0.NW and CR0.CD bits are both set to ‘1’) external snoops are accepted in a DP system and inhibited in a UP system. Note that when snoops are inhibited, address parity is not checked, and APCHK# will not be asserted for a corrupt address. However, when snoops are accepted, address parity is checked (and APCHK# will be asserted for corrupt addresses).

39. Recognizing INTR After Its INTA Cycle

The *Pentium Processor Family Developer’s Manual,* Volume 1, Section 5.1.32 and 21.1.21, states “To guarantee recognition if INTR is asserted asynchronously it must have been deasserted for a minimum of 2 clocks before being returned active to the Pentium processor”. However, to avoid an inadvertent second recognition of INTR, a clarification is being added to specify when INTR should be deasserted.

If INTR remains asserted beyond completion of its second INTA cycle and the IF flag is set, INTR can be recognized again. In real mode for example, if the handler enables interrupts prior to INTR being deasserted, a second INTR will be recognized. Interrupts may be re-enabled quickly if the STI instruction is executed early in the handler or the handler is very short, such that IF is set soon after the completion of the second INTA cycle. If trap gates are used in protected mode, INTR may be recognized even earlier because the IF flag remains set (and does not get cleared).

To clarify this, the *Pentium Processor Family Developer’s Manual,* will add a NOTE, “To ensure INTR is not recognized inadvertently a second time, deassert INTR no later than the BRDY# of the second INTA cycle and no earlier than the BRDY# of the first INTA cycle.

40. Multiple Interrupt Delaying Instructions May Not Delay More Than One Instruction

In the next publication of the *Intel Architecture Developer's Manual,* Volume 2: The Instruction Set, the following footnote will be added to the descriptions of the STI and the MOV SS,.. instructions as below. This footnote will be referenced from the part of the description of the instruction where the feature that delays interrupts for one more instruction is explained.

Footnote:
In a sequence of instructions which individually delay interrupts past the following instruction, only the first in the sequence is guaranteed to have this effect. Thus in the sequence STI/ MOV SS/ MOV ESP/ interrupts may be recognized before MOV ESP executes.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the *Pentium Processor Family Developer's Manual*, Volumes 1 and 3. All Documentation Changes will be incorporated into a future version of the appropriate Pentium processor documentation.

1. Package Dimensions, Volume 1, page 25-5

Page 25-5 containing table 25-3 is missing from Volume 1 of the *Pentium Processor Family Developer's Manual.* The correct table is shown below:

Table 25-3. Package (Without The Heat Spreader)

Family: 296 Pin Ceramic Pin Grid Array Package						
	Millimeters			Inches		
Symbol	Min	Max	Notes	Min	Max	Notes
A	3.27	3.83	Ceramic Lid	0.129	0.151	Ceramic Lid
A1	0.66	0.86	Ceramic Lid	0.026	0.034	Ceramic Lid
A2	2.62	2.97		0.103	0.117	
B	0.43	0.51		0.017	0.020	
D	49.28	49.78		1.940	1.960	
D1	45.59	45.85		1.795	1.805	
D3	24.00	24.25	Includes Fillet	0.945	0.955	Includes Fillet
e1	2.29	2.79		0.090	0.110	
F		0.127	Flatness of the top of the package, measured diagonally		0.005	Flatness of the top of the package, measured diagonally
L	3.05	3.30		0.120	1.130	
N	296		Total Pins	296		Total Pins
S1	1.52	2.54		0.060	0.100	

2. Boundary Scan Chain Bit Sequence, Volume 1, Page 27-2

Section 27.2 describes the bit order of the Pentium processor boundary scan register (left to right, top to bottom). On page 27-2, after bit FRCMC#, the sequence continues as "Reserved, Reserved, Reserved, Reserved, BF, STPCLK#, Reserved, Reserved, Reserved, Reserved, Reserved, Reserved".

The correct bit sequence after bit FRCMC# should be "Reserved, Reserved, BF0, BF1, Reserved, STPCLK#, Reserved, Reserved, Reserved, Reserved, Reserved, Reserved, CPUTYP".

3. BF1 Pin Location, Volume 1, Page 17-3 and Page 17-5

In Figure 17-2 Pentium processor Pinout Pin Side View, the BF1 pin is incorrectly located located at position W33. The correct location for the BF1 pin is position X34.

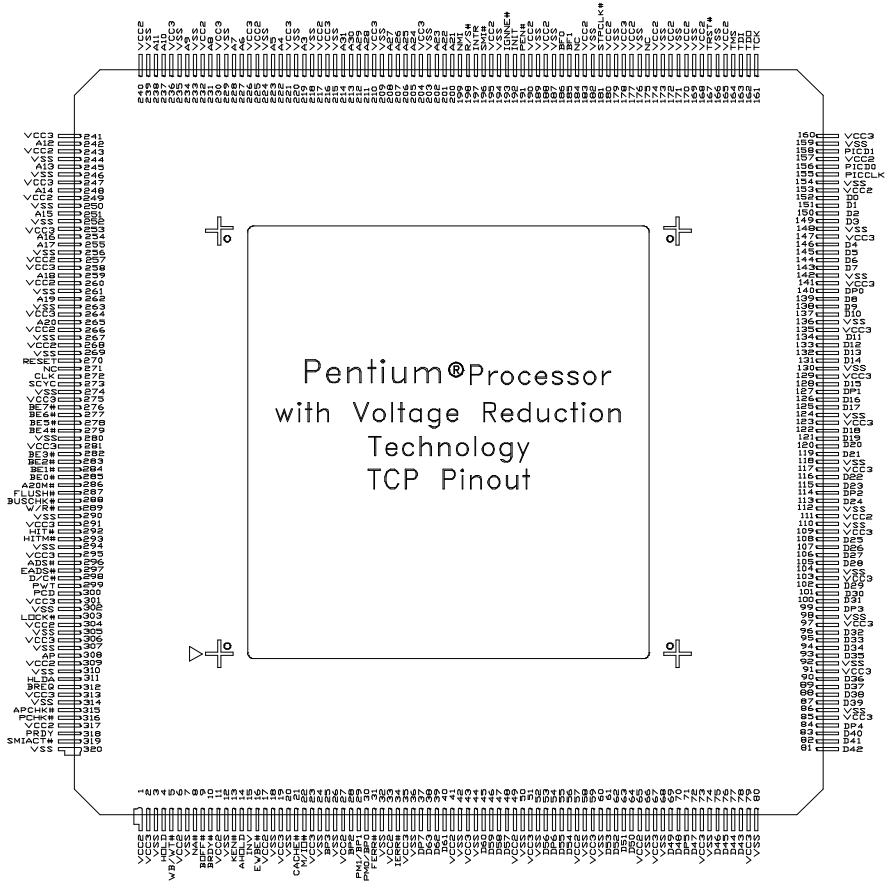
Table 17-1 Pin Cross Reference by Pin Name incorrectly lists BF1 at position Y34. The BF1 pin should be at position X34.

4. 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.

5. Incorrect TCP Pinout Drawing

The TCP pinout drawing on page 8 of the Pentium Processor with Voltage Reduction Technology Data Sheet was inadvertently rotated. The correct orientation is shown below.



6. 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.

7. Errors in the Detailed Descriptions of FSUB, FSUBR, FDIV, FDIVR and Related Instructions

The FSUB/FSUBP/FISUB instructions (page 25-145 in the 1995 *Pentium Processor Family Developer's Manual* , Volume 3) and the FSUBR/FSUBRP/FISUBR instructions (page 25-147 in Vol. 3 of the Manual) each have three erroneous entries in their tables of eight specific forms (the fourth, fifth and sixth forms are wrong in both cases). Also there are errors under the **Operation** and **Description** headings. Further, in the individual descriptions of the specific forms, three different styles are used. In the next revision, all descriptions will be changed to match the style used here for the corrections.

FSUB/FSUBP/FISUB	**Instruction**	**Description**
	FSUB ST(i), ST	ST(i) <-- (ST subtracted from ST(i)).
	FSUBP ST(i), ST	ST(i) <-- (ST subtracted from ST(i)); POP ST.
	FSUBP	ST(1) <-- (ST subtracted from ST(1)); POP ST.

Operation DEST <-- DEST - Other Operand <<2nd line is O.K.>>

Description The subtraction instructions subtract the other operand from the destination (which is the leftmost operand in the forms with two explicit operands; otherwise it is always the stack top, ST) and return the difference to the destination.

FSUBR/FSUBRP/FISUBR	**Instruction**	**Description**
	FSUBR ST(i), ST	ST(i) <-- (ST(i) subtracted from ST).
	FSUBRP ST(i), ST	ST(i) <-- (ST(i) subtracted from ST); POP ST.
	FSUBRP	ST(1) <-- (ST(1) subtracted from ST); POP ST.

Operation DEST <-- Other Operand - DEST <<2nd line is O.K.>>

Description The reverse subtraction instructions subtract the destination (which is the leftmost operand in the forms with two explicit operands; otherwise it is always the stack top, ST) from the other operand and return the difference to the destination.

The FDIV/FDIVP/FIDIV instructions (page 25-98 in the 1995 *Pentium Processor Family Developer's Manual*, Volume 3) and the FDIVR/FDIVRP/FIDIVR instructions (page 25-100 in Vol. 3 of the Manual) have errors under the **Operation** and **Description** headings. Also, in the table of eight specific forms for FDIVR/FDIVRP/FIDIVR the fourth, fifth and sixth forms have a meaningless (but probably not misleading) word "Divide" in front of their otherwise equation style descriptions; it should be ignored. Further, among the total of 16 individual descriptions of the specific forms, three different styles are used. In the next revision, all descriptions will be changed to match the style given here: *DEST* <-- (*OP1* divided by *OP2*)

FDIV/FDIVP/FIDIV

Operation <<The abbreviation for Source, which should be SRC, is misspelled twice as SCR>>

Description The division instructions divide the destination (which is the leftmost operand in the forms with two explicit operands; otherwise it is always the stack top, ST) by the other operand and return the quotient to the destination.

FDIVR/FDIVRP/FIDIVR

Operation <<O.K. as is>>

Description The reverse division instructions divide the other operand by the destination (which is the leftmost operand in the forms with two explicit operands; otherwise it is always the stack top, ST) and return the quotient to the destination.

8. Figures of TR1 and TR12, Volume 1, Page 33-13 and Page 33-16

In the *Pentium Processor Family Developer's Manual*, Volume 1, Section 33.2.1.4 Figure 33-4 should be replaced by Figure 33-5 in Section 33.3, and Figure 33-5 in Section 33.3 should be replaced by Figure 33-4 in Section 33.2.1.4.

9. APIC AC Specification t_{63}, Page 23-12, 23-17 and 23-22

In the *Pentium Processor Family Developer's Manual*, Volume 1, Table 23-6, 23-8 and 23-10 list t_{63} as:

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{63}	PICCLK Ratio (CLK/PICCLK)	4.0		nS		(32)

It should be changed to,

t_{63}	PICCLK Ratio (CLK/PICCLK)	4				(32)

10. PUSHA, PUSHF, POPA & POPF Can Cause Alignment Faults

The instructions **POPA/POPAD** (*Pentium Processor Family Developer's Manual*, Volume 3, page 25-250), **POPF/POPFD** (page 25-252), **PUSHA/PUSHAD** (page 25-256) and **PUSHF/PUSHFD** (page 25-258) can cause the alignment check fault (#AC; interrupt 17) when it is enabled, just as PUSH and POP can, but that specification has been omitted from the previous instructions' descriptions. For each of these four instruction descriptions, the following should be added:

To the sections called **Protected Mode Exceptions,** add at the end: #AC for unaligned memory reference if the current privilege level is 3 (the word operand form is unaligned unless the address is divisible by 2; the dword operand form (used by the mnemonic ending in D) is unaligned unless the address is divisible by 4).

To the sections called **Virtual 8086 Mode Exceptions,** add at the end: #AC for unaligned memory reference if the current privilege level is 3 (alignment is the same as for protected mode).

11. Corrections to BT, BTC, BTR and BTS Instruction Descriptions

The specific descriptions of the BTU, BTU, BTU and BCE instructions, on pages 25-42, 25-44, 25-46 and 25-48 of the *Pentium Processor Family Developer's Manual*, Volume 3, are incomplete. The **Flags Affected** section for all of these bit test instruction descriptions should include the statement that the flags OF, SF, ZF, AF and PF are all left undefined after execution of one of these instructions. These flag effects are already correctly specified on page B1 of the *Pentium Processor Family Developer's Manual*, Volume 3.

Also, on p. 25-44 for the BTC instruction, the current sentence in the Flags Affected section should be changed from "The CF flag contains the complement of the selected bit" to "The CF flag contains the value of the selected bit", meaning the value before the BTC execution.

12. Corrections for Description of a Stack Overflow On Interrupt to Inner Privilege Case

In the *Pentium Processor Family Developer's Manual,* Volume 3 starting on p. 25-173, the Pentium processor operation is described in pseudo-code for the INT/INTO instructions, and also all the external interrupts and internal exceptions which go through the interrupt vector/descriptor table. On p. 25-175 an IF/FI clause inside the "INT-TO-INNER-PRIV PROC" reads as follows:

IF 32-bit gate
THEN new stack must have room for 20 bytes else #SS(0)
ELSE new stack must have room for 10 bytes else #SS(0)
FI

The error code of zero given for stack overflows is incorrect. The correct error code is the SS selector, so the two entries "#SS(0)" will be changed to "#SS(SS selector)". A new clause will also be added immediately after the corrected clause above describing the checking done for the additional stack space required if the interrupt has an error code. The corrected text is as follows:

IF 32-bit gate
THEN new stack must have room for 20 bytes else #SS(SS selector)
ELSE new stack must have room for 10 bytes else #SS(SS selector)
IF interrupt was caused by exception with error code
THEN Stack limit must allow push of 4 more bytes if 32-bit gate, or 2 more bytes if 16-bit gate
ELSE #SS(SS selector);
FI;

These clauses apply in protected mode when an interrupt tries to transfer control through a trap or interrupt gate in the Interrupt Descriptor Table to an inner privilege level service routine. If the 20 or 24 bytes (or 10 or 12 bytes in 16-bit mode) that must be stored on the new stack would cause it to overflow, the protection system triggers #SS (interrupt 12) instead, which allows the O.S. to allocate more stack space.

13. 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 sec. 23.3.6. "FNOP" will be changed to "NOP" in the FSETPM description.

14. Corrections in the Pseudocode Descriptions of the Instructions CALL, IRET(D) & RET

In the *Pentium Processor Family Developer's Manual*, Volume 3 the pseudocode descriptions of the detailed operation of the instructions CALL, IRET(D) and RET have the following errors:

For the CALL instruction, on p. 25-52, in both the section labeled "CONFORMING-CODE-SEGMENT" and the section labeled "NONCONFORMING-CODE-SEGMENT", just after the line "Stack must be big enough for return address ELSE #SS(0)" the following line should be added: "**Push return address onto stack**".

Also on p. 25-52, in the section labeled "CALL-GATE", just after the line "DPL of selected descriptor must be ≤ CPL ELSE #GP(code segment selector)" the following line should be added: "**Segment must be present ELSE #NP(code segment selector)**".

For the IRET(D) instruction, on p. 25-187, in the section labeled “RETURN-OUTER-LEVEL”, the line “SS must be present, else #NP(SS Selector)” should be corrected to read “**Stack segment must be present, ELSE #SS(SS Selector)”.** (A not present stack segment always causes #SS, while other not present segments cause #NP. The capitalization of ELSE and spelling out of SS are just minor style improvements.)

Also on p. 25-187, 5 lines below the correction just given, the line “Load EFLAGS with values at (eSP+8);” should be followed by the line “**Increment eSP by 12**”. Similarly, 3 lines below that line, the line “Load EFLAGS with values at (eSP+4);” should be followed by the line “**Increment eSP by 6**”. These steps are needed, even though eSP is overwritten in the *next* step, because the new SS:eSP are read from the stack using the current SS:eSP values.

For the RET instruction, on p. 25-273, in the fifteenth line from the top of the page, “Segment must be present, else #NP(selector)” should be corrected to read “**Stack segment must be present, ELSE #SS(SS Selector)”.**

15. Errors in 3 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).

16. Execution Tracing – Fast Messages

Paragraph three of section 29-1 of the *Pentium Processor Family Developer’s Manual,* Volume 1 incorrectly implies the address supplied by the processor in fast branch trace messaging mode. Fast messages only produce the linear address of the instruction causing the taken branch as documented in the “Fast Cycle” section on the following page.

The third paragraph in section 29-1 should be replaced as follows:

“Branch trace messages may be enabled by setting the Execution Tracing bit, TR, of TR12 (bit 1) to a 1. Once enabled, there are two forms of branch trace messages: normal and fast. Normal messages produce two cycles, one for the branch target linear address and one for the linear address of the instruction causing the taken branch. Fast messages only produce the second of these two cycles. Fast execution tracing is enabled by setting bit 8 of TR12 to 1.”

Part III:

Specification Update for Pentium® OverDrive® Processors

GENERAL INFORMATION

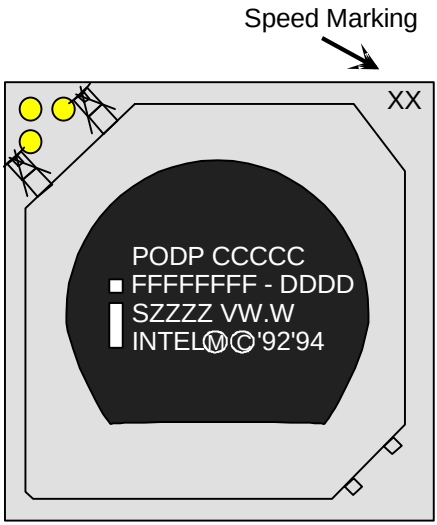
This section covers the various Pentium® OverDrive® processors.

The 63- and 83-MHz Pentium OverDrive processors are Pentium processor technology upgrades for Intel486™ processor-based systems. For more information on the 63- and 83-MHz Pentium OverDrive processors, please refer to the *Intel Pentium OverDrive Processor* datasheet, Order Number 290544.

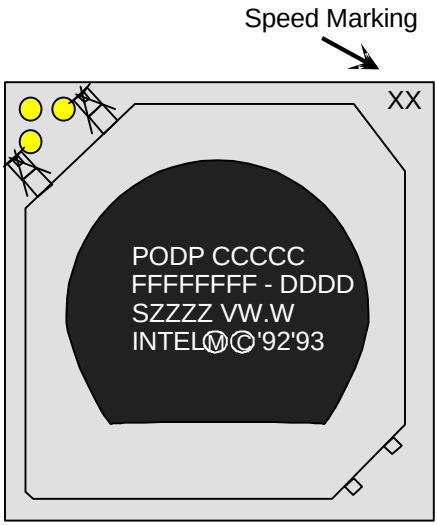
The 120/133-MHz Pentium OverDrive processors are Pentium processor technology upgrades for 60/66-MHz Pentium processor-based systems. The 125, 150, and 166-MHz Pentium OverDrive processors are Pentium processor technology upgrades for 75, 90, and 100-MHz Pentium processor-based systems. For more information on the Pentium OverDrive processors for Pentium processor based systems, please refer to the *Pentium OverDrive Processors for Pentium Processor-Based Systems* datasheet, (Order Number 290579).

Top Markings

The Pentium OverDrive processor can be identified by the number on the base of the heat sink, under the integrated fan. To remove the fan, squeeze the retaining clips in the upper left corner of the chip and lift up. The figure below shows the laser mark found on the heatsink base of the processor. A marking is also used to indicate the speed of the part and is located in the upper right of the package.



63- and 83-MHz Pentium® OverDrive® processors for Intel486™ processor-based systems



120/133-, 125-, 150- and 166-MHz Pentium® OverDrive® processors

NOTES:

- XX or XXX = Core Speed (MHz).
- CCCCC = Product Code
- FFFFFFFF = FPO # (Test Lot Traceability #).
- DDDD = Serialization Code
- SZZZZ = Spec number
- VW.W = Version number
- The FPO - Serial Number is unique for every Pentium® OverDrive® processor.
- The Version Number is used to easily identify major processor steppings: it applies to OverDrive processors only

Basic Pentium® OverDrive® Processor Identification Information

CPUID					Manuf. Stepping[1]	Speed (MHz) Core / Bus	S-Spec	Version	Notes
Product Code	Type	Family	Model	Stepping					
PODP5V63	1	5	3	1	B1	63/25	SZ953	1.0	
PODP5V63	1	5	3	1	B2	63/25	SZ990	1.1	
PODP5V83	1	5	3	2	C0	83/33	SU014	2.1	
PODP5V133	0	5	1	A	tA0	120/60,133/66	SU082	1.0	2
PODP3V125	0	5	2	C	aC0	125/50	SU081	1.0	3
PODP3V150	0	5	2	C	aC0	150/60	SU083	1.0	3
PODP3V166	0	5	2	C	aC0	166/66	SU084	1.0	3

NOTES:

- 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.
- 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.

1. Manufacturing steppings prefixed with a lower-case “t” refer to the 120/133-MHz Pentium OverDrive processors for 60/66 MHz Pentium processor-based systems. Steppings prefixed with a lower-case “a” refer to the 125/150/166-MHz Pentium OverDrive processors for 75/90/100 MHz Pentium processor-based systems. Steppings without a prefix refer to the 63/83-MHz Pentium OverDrive processors for Intel 486 processor-based systems.
2. The V_{CC} Operating Voltage for these parts is 4.75 - 5.40V.
3. The V_{CC} Operating Voltage for these parts is 3.135 - 3.6V

Summary Table of Changes

The following table indicates the Specification Changes, Errata, Specification Clarifications, or Documentation Changes which apply to the listed Pentium OverDrive processor steppings. Intel intends to correct 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. Any items that are shaded are new for this revision of the document. 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.
PP##:	Please refer to Part II: Specification Update for 75-, 90-, 100-, 120-, 133-, 150-, 166- and 200-MHz Pentium® Processors.
Shaded:	This item is either new or modified from the previous version of the document.

NO.	B1	B2	C0	tA0	aC0	Refer to:	Plans	SPECIFICATION CHANGES
1	X	X	X				Doc	Clock low time (t_3) change
2		X	X				Doc	Removal of V_{CC} pin A4
3		X	X				Doc	Minimum operating temperature
4	X	X	X			PP12	Doc	STI followed by FP instruction doesn't delay interrupt window
5	X	X	X			PP13	Doc	IDT limit violation causes GP fault, not interrupt 8
6	X	X	X				Doc	Maximum I_{CC} values
7	X	X	X		X	PP19	Doc	Requirements for STPCLK# re-assertion recognition
8				X	X	PP22	Doc	Leakage current I_{LI}/I_{LO} applies only in the valid logic states
NO.	**B1**	**B2**	**C0**	**tA0**	**aC0**	**Refer to:**	**Plans**	**ERRATA**
1	X	X	X			PP1	NoFix	Branch trace messages during lock cycles
2	X	X	X			PP2	NoFix	Breakpoint or single-step may be missed for one instruction following STI
3	X	X	X			PP3	NoFix	I/O restart does not function during single-stepping or data breakpoint exceptions

NO.	B1	B2	C0	tA0	aC0	Refer to:	Plans	ERRATA (Cont'd.)
4	X	X	X			PP4	NoFix	NMI or INIT in SMM with I/O restart during single-stepping
5	X	X	X			PP5	NoFix	SMI# and FLUSH# during shutdown
6	X	X					Fixed	BIST disabled
7								Maximum I_{CC} usage (Refer to Spec Change 6)
8	X	X	X			PP6	NoFix	No shutdown after internal error
9	X	X	X			PP7	NoFix	FLUSH# with a breakpoint pending causes false DR6 values
10	X	X	X		X	PP10	NoFix	STPCLK# deassertion not recognized for 5 CLKs after BRDY# returned
11	X	X	X	X	X	PP14	NoFix	NMI or INIT during HALT within an SMM may cause large amount of bus activity
12	X	X	X		X	PP17	NoFix	STPCLK# restrictions during EWBE#
13	X	X					Fixed	CLK required for UP# to be driven
14	X	X				PP18	Fixed	Multiple allocations into branch target buffer
15	X	X	X			PP24	NoFix	FLUSH#, INIT or Machine Check dropped due to floating-point exception
16	X	X	X			PP25	NoFix	Floating-point operations may clear Alignment Check bit
17	X	X	X			PP26	NoFix	CMPXCHG8B across page boundary may cause invalid opcode exception
18	X	X	X			PP27	NoFix	Single-step debug exception breaks out of HALT
19	X	X	X			PP28	NoFix	Branch trace message corruption
20	X	X	X			PP32	NoFix	EIP altered after specific FP operations followed by MOV Sreg. Reg
21	X	X	X	X	X	PP33	NoFix	WRMSR into illegal MSR does not generate GP fault
22	X	X	X			PP34	NoFix	Inconsistent data cache state from concurrent snoop and memory write
23	X	X	X	X	X	PP36	NoFix	Incorrect FIP after RESET
24	X						Fixed	Errors in least significant bit of an FMUL instruction result with specific operands
25	X	X	X	X	X	PP37	NoFix	Second assertion of FLUSH# not ignored
26	X	X	X			PP9	NoFix	SMIACT# premature assertion during replacement writeback cycle
27	X	X	X	X	X	PP38	NoFix	Segment limit violation by FPU operand may corrupt FPU state

NO.	B1	B2	C0	tA0	aC0	Refer to:	Plans	ERRATA (Cont'd.)
28	X	X	X	X	X	PP39	NoFix	FP exception inside SMM with pending NMI hangs system
29	X	X	X			PP40	NoFix	Current in Stop Clock state exceeds specification
30	X	X	X	X	X	PP43	NoFix	Data breakpoint deviations
31	X	X	X	X	X	PP44	NoFix	Event monitor counting discrepancies
32	X	X	X	X	X	PP45	NoFix	VERR type instructions causing page fault task switch with T bit set may corrupt CS:EIP
33	X	X	X		X	PP48	NoFix	Matched but disabled data breakpoint can be lost by STPCLK# assertion
34	X	X	X		X	PP49	NoFix	STPCLK# ignored in SMM when INIT or NMI pending
35	X	X	X	X	X	PP51	NoFix	A fault causing a page fault can cause an instruction to execute twice
36	X	X	X	X	X	PP53	NoFix	FBSTP stores BCD operand incorrectly if address wrap & FPU error both occur
37	X	X	X	X	X	PP54	NoFix	V86 interrupt routine at illegal privilege level can cause spurious pushes to stack
38	X	X	X	X	X	PP55	NoFix	Corrupted HLT flag can cause skipped or incorrect instruction, or CPU hang
39	X	X	X	X	X	PP56	NoFix	Benign exceptions can erroneously cause double fault
40	X	X	X	X	X	PP57	NoFix	Double fault counter may not increment correctly
41	X	X	X	X	X	PP59	NoFix	Short form of mov EAX/ AX/ AL may not pair
42	X	X	X	X	X	PP60	NoFix	Turning off paging may result in prefetch to random location
43	X	X	X		X	PP61	NoFix	STPCLK# or FLUSH# after STI
44	X	X	X		X	PP62	NoFix	REP string instruction not interruptable by STPCLK#
45	X	X	X	X	X	PP63	NoFix	Single step may not be reported on first instruction after FLUSH#
46	X	X	X	X	X	PP66	NoFix	STPCLK# on RSM to HLT causes non-standard behavior
47				X	X	PP15	NoFix	RUNBIST restrictions when run through boundary scan circuitry
48				X	X	PP46	NoFix	BUSCHK# interrupt has wrong priority
49					X	PP50	NoFix	STPCLK# pullup not engaged at RESET
50				X	X	PP52	NoFix	Machine check exception pending, then HLT, can cause skipped or incorrect instruction, or CPU hang

NO.	B1	B2	C0	tA0	aC0	Refer to:	Plans	ERRATA (Cont'd.)
51				X		PP64	NoFix	Double fault may generate illegal bus cycle
52				X	X	PP65	NoFix	TRST# not asynchronous
53				X			NoFix	I/O Buffer Leakage
54				X			NoFix	Break in Boundary Scan Chain
55				X	X		NoFix	Incorrect type field in CPUID
56	X	X	X	X	X	PP67	NoFix	Code cache dump may cause wrong IERR#
57	X	X	X	X	X	PP68	NoFix	Asserting TRST# pin or issuing JTAG instructions does not exit TAP Hi-Z state
58				X			Fix	CLK input capacitance exceeds specification
59	X	X	X	X	X	PP69	NoFix	ADS# may be delayed after HLDA deassertion
60	X	X	X	X	X	PP70	NoFix	Stack underflow in IRET gives #GP, not #SS
61				X	X	PP71	NoFix	Performance monitoring pins PM[1:0] may count the events incorrectly

NO.	B1	B2	C0	tA0	aC0	Refer to:	Plans	SPECIFICATION CLARIFICATIONS
1	X	X	X				Doc	CACHE#, KEN# and BLAST# Behavior
2	X	X	X				Doc	Behavior of writeback support pins in writethrough mode
3	X	X	X				Doc	HITM# deassertion behavior
4	X	X	X	X	X	PP6	Doc	Only one SMI# can be latched during SMM
5	X	X	X	X	X	PP11	Doc	SMIACT# handling during snoop writeback
6	X	X	X	X	X	PP15	Doc	Event monitor counters
7	X	X	X	X	X	PP17	Doc	NMI signal description
8	X	X	X	X	X	PP18	Doc	BTB behavior when entering SMM
9	X	X	X	X	X	PP19	Doc	SMI# activation may cause a nested NMI handling
10	X	X	X	X	X	PP20	Doc	Exit from shutdown
11	X	X	X	X	X	PP22	Doc	Code breakpoints set on meaningless prefixes not guaranteed to be recognized
12	X	X	X	X	X	PP23	Doc	Resume flag should be set by software
13	X	X	X	X	X	PP24	Doc	Data breakpoints on INS delayed one iteration
14	X	X	X	X	X	PP26	Doc	CPUID Feature flags
15				X	X	PP14	Doc	EADS# recognition

NO.	B1	B2	C0	tA0	aC0	Refer to:	Plans	SPECIFICATION CLARIFICATIONS (Cont'd.)
16				X	X	PP16	Doc	KEN# sets cacheability for restarted cycles
17	X	X	X	X	X	PP30	Doc	When L1 cache disabled, inquire cycles are blocked
18	X	X	X	X	X	PP32	Doc	Cache test registers are modified during FLUSH#
19	X	X	X	X	X	PP33	Doc	For correct translations, the TLB should be flushed after the PSE bit in CR4 is set
20	X	X	X	X	X	PP36	Doc	Extra code break can occur on I/O or HLT instruction if SMI coincides
21	X	X	X	X	X	PP37	Doc	4-MByte page extensions
22				X	X	PP5	Doc	Pullup/pulldown resistor values for some configuration pins
23	X	X	X	X	X	PP39	Doc	Recognizing INTR after its INTA cycle
24	X	X	X	X	X	PP40	Doc	Multiple interrupt delaying instructions may not delay more than one instruction
NO.	**B1**	**B2**	**C0**	**tA0**	**aC0**	**Refer to:**	**Plans**	**DOCUMENTATION CHANGES**
1	X	X	X	X	X	PP4	Doc	JMP cannot do a nested task switch, Volume 3, page 13-12
2	X	X	X	X	X	PP6	Doc	Interrupt sampling window, Volume 3, page 23-39
3	X	X	X	X	X	PP7	Doc	Errors in the detailed descriptions of FSUB, FSUBR, FDIV, FDIVR and related instructions
4	X	X	X	X	X	PP8	Doc	Figures of TR1 and TR12, Volume 1, page 33-13 and page 33-16
5	X	X	X	X	X	PP10	Doc	PUSHA, PUSHF, POPA & POPF can cause alignment faults
6	X	X	X	X	X	PP11	Doc	Corrections to BT, BTC, BTR and BTS instruction descriptions
7	X	X	X	X	X	PP12	Doc	Corrections for description of a stack overflow on interrupt to inner privilege case
8	X	X	X	X	X	PP13	Doc	FSETPM is like NOP, not like FNOP
9	X	X	X	X	X	PP14	Doc	Corrections in the pseudocode descriptions of the instructions CALL, IRET(D) & RET
10				X	X	PP3	Doc	BF1 pin location, volume 1, page 17-3 and 17-5
11	X	X	X	X	X	PP15	Doc	Errors in 3 tables of special descriptor types
12	X	X	X	X	X	PP16	Doc	Execution tracing - fast messages

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the Pentium OverDrive Processor datasheets. All Specification Changes will be incorporated into future versions of the appropriate document(s).

1. Clock Low Time (t_3) Change

The AC timings for the Pentium OverDrive processor CLK low time have been changed to better meet the needs of the installed base of Intel486 processor-based systems. The change to the Pentium OverDrive processor CLK low time specification applies to the Pentium OverDrive processor only, and does not affect any other clock specifications, such as CLK High time or rise/fall specifications.

NEW SPECIFICATIONS

Pentium® OverDrive® Processor 25 MHz AC Characteristics
V_{CC} = 5V ±5%; T_A(IN) = 10°C T_O +55°C; C_L = 50PF unless otherwise specified

Symbol	Parameter	Min	Max	Unit
t_3	CLK Low Time	11		nS

Pentium® OverDrive® Processor 33 MHz AC Characteristics
V_{CC} = 5V ±5%; T_A(IN) = 10°C T_O +55°C; C_L = 50PF unless otherwise specified

Symbol	Parameter	Min	Max	Unit
t_3	CLK Low Time	8		nS

2. Removal of V_{CC} Pin A4

To resolve compatibility concerns with a limited number of incorrectly designed motherboards, pin A4 (V_{CC}) will be removed from the Pentium OverDrive processors and will no longer be present on the package. The removal of this V_{CC} pin will not adversely affect the operation of the processor due to the large number of V_{SS} and V_{CC} pins remaining in the outer row.

3. Minimum Operating Temperature

The specification for the minimum operating temperature of the Pentium OverDrive processors has been changed. All other temperature specifications, such as the absolute maximum ratings, remained unchanged. The new temperature specification is detailed in the table below:

$T_{A(IN)}$ Old Specification (°C)	$T_{A(IN)}$ New Specification (°C)
0 to +55	10 to +55

4-5. Refer to Summary Table of Changes

6. Maximum I_{CC} Values

The maximum current requirements originally estimated for the Pentium Over Drive processors were found to be slightly lower than the actual values. The new values are listed in the table below.

Old max	New max	Notes
1.9 A	2.2 A	@ 63/25 MHz
2.6 A	2.8 A	@ 83/33 MHz

7-8. Refer to Summary Table of Changes

ERRATA

1 - 5. Refer to Summary Table of Changes

6. BIST Disabled

PROBLEM: The current production stepping of the Pentium OverDrive processor has disabled the Built-In Self Test (BIST) functionality.

IMPLICATION: If BIST is performed (AHOLD high at the falling edge of RESET), the processor will not actually perform the BIST, but will return a value of 'zero' in the EAX register to indicate that all tests have passed.

WORKAROUND: None required.

STATUS: This erratum has been fixed in the C-0 stepping.

7. Maximum I_{CC} Usage (Refer to Spec. Change 6.)

8 - 12. Refer to Summary Table of Changes

13. CLK Required for UP# to be Driven

PROBLEM: The Upgrade Present (UP#) output pin is intended to be driven low after power-up to indicate that an upgrade processor is present in the system. The UP# pin on the Pentium OverDrive processor requires that the CLK input toggle while the system is starting to insure that RESET reaches the UP# circuitry. If CLK does not toggle, UP# may or may not be driven low, depending on the initial state of the UP# circuitry.

IMPLICATION: If an Intel486 processor system is dependent on having UP# driven low before driving the CLK input on the Pentium OverDrive processor, it may never boot since CLK may never be driven to the processor if UP# remains high. This is generally only a potential issue in systems with two processor sites, such as those with a surface-mount Intel486 processor and a PGA upgrade processor site.

WORKAROUND: Most two site systems have the ability to disable the processor in the surface-mount location so that a different Intel486 (non-upgrade) processor may be used in the PGA socket location. If the system has a jumper or switch that is documented to specifically disable the surface mount processor (or the original processor in a two socket system), use it to disable the second processor and thereby route the CLK signal to the upgrade PGA socket location.

STATUS: This erratum has been fixed in the C-0 stepping.

14. Multiple Allocations Into Branch Target Buffer

PROBLEM: Please refer to Part II of this document. This errata entry for the Pentium OverDrive processor has been added for clarification of the issue as it applies to this processor specifically.

IMPLICATION: Please refer to Part II of this document.

WORKAROUND: Please refer to Part II of this document.

STATUS: This erratum has been fixed in the C-0 stepping. Although this erratum has been recreated in proprietary Intel test systems, it has not been observed on a Pentium OverDrive processor in an actual personal computer system.

15 - 23. Refer to Summary Table of Changes

24. Errors in Least Significant Bit of an FMUL Instruction Result With Specific Operands

PROBLEM: The result of multiplying two operands, one of which is typically close to infinity or close to the smallest representable normal number, may be incorrect in the Least Significant Bit (LSB) and may result in flags that are incorrectly set. This problem is limited to a small percentage of the first production units of the 63MHz Pentium OverDrive processor (specification number SZ953, Ver#: 1.0).

All precisions: single, double and extended are affected. All rounding modes: nearest, up, down and chop are affected. Only specific operand pairs result in errors.

IMPLICATION: Only a fraction of the Pentium OverDrive processors may produce errors in the least significant bit of the result when using the FMUL instruction with one of these specific input operands. Any of the exception flag bits in the FPU status word may also be incorrect.

WORKAROUND: None identified. Intel includes software with the Pentium OverDrive processor that can detect this erratum. Please see the following Status paragraph.

STATUS: The Pentium OverDrive processors currently in production are free from this defect. Only a fraction of the first production units at 63-MHz marked SZ953 may exhibit the defect. Owners of Pentium OverDrive processors should run the diagnostics on the disk supplied with the Pentium OverDrive processor. The diagnostic program tests the processor and will notify the user of pass/fail information. Should the processor fail, the diagnostic program will report that “The Floating Point Conformance Test has Failed” and will wait for a keystroke before continuing. Owners of the Pentium OverDrive processor should contact the Intel support line if there are any questions.

25. Second Assertion of FLUSH# Not Ignored

PROBLEM: Please refer to Part II of this document. This errata entry for the Pentium OverDrive processors has been added for clarification of the issue as it applies to these OverDrive processors specifically.

IMPLICATION: This erratum applies to the Pentium OverDrive processor in Enhanced (WB) bus mode only . Standard bus mode (WT) is not affected. Please refer to Part II of this document for more information.

WORKAROUND: Please refer to Part II of this document

STATUS: For the steppings affected, please see the Summary of Changes table at the beginning of this section.

26 - 28. Refer to Summary Table of Changes

29. Current in Stop Clock State Exceeds Specification

PROBLEM: Please refer to Part II of this document. This errata entry for the Pentium OverDrive processors has been added for clarification of the issue as it applies to these OverDrive processors specifically.

IMPLICATION: Please refer to Part II of this document for more information.

WORKAROUND: None Identified. The workaround as stated in Part II of this document is not valid for the Pentium OverDrive processor since the boundary scan pins are not available on production units.

STATUS: For the steppings affected, please see the Summary of Changes table at the beginning of this section.

30-38. Refer to Summary Table of Changes

39. Benign Exceptions Can Erroneously Cause Double Fault

PROBLEM: Please refer to Part II of this document. This errata entry for the Pentium OverDrive processors has been added for clarification of the issue as it applies to these OverDrive processors specifically.

IMPLICATION: This erratum applies to the Pentium OverDrive processor in Case 1 only. Case 2 “A machine check exception (INT 18) is generated” does not apply. Please refer to Part II of this document for more information.

WORKAROUND: Please refer to Part II of this document

STATUS: For the steppings affected, please see the Summary of Changes table at the beginning of this section.

40-42. Refer to Summary Table of Changes

43. STPCLK# or FLUSH# After STI

PROBLEM: Please refer to Part II of this document. This errata entry for the 63 and 83-MHz Pentium OverDrive processors has been added for clarification of the issue as it applies to these OverDrive processors specifically.

IMPLICATION: In the case of “FLUSH# After STI”, this erratum applies to the Pentium OverDrive processor in Enhanced (WB) bus mode only . Standard bus mode (WT) is not affected. Please refer to Part II of this document for more information.

WORKAROUND: Please refer to Part II of this document.

STATUS: For the steppings affected, please see the Summary of Changes table at the beginning of this section.

44. Refer to Summary Table of Changes

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

PROBLEM: Please refer to Part II of this document. This errata entry for the 63 and 83-MHz Pentium OverDrive processors has been added for clarification of the issue as it applies to these OverDrive processors specifically.

IMPLICATION: This erratum applies to the Pentium OverDrive processor in Enhanced (WB) bus mode only . Standard bus mode (WT) is not affected. Please refer to Part II of this document for more information.

WORKAROUND: Please refer to Part II of this document.

STATUS: For the steppings affected, please see the Summary of Changes table at the beginning of this section.

46-52. Refer to Summary Table of Changes

53. I/O Buffer Leakage

PROBLEM: When an I/O pin is driven high by the Pentium OverDrive processor and then tri-stated, the output is still weakly driven high until the line is driven low by either the processor or another source.

IMPLICATION: If a weak pull-down (greater than 1kΩ) is used on a signal line and the value of the line is expected at a given time after being tri-stated by the processor, the incorrect value can be read.

WORKAROUND: None identified at this time.

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

54. Break in Boundary Scan Chain

PROBLEM: The boundary scan problem can most easily be described as a broken link in the boundary scan chain. The effects of this broken link on the output testing are minimal, since the majority of the chain is before the break. This part of the chain can still be filled, and the data sent to the output pins. However, any pins after the break will not be able to send out the correct output data. The effects of this broken link on input testing are more drastic. Only the last four cells worth of data can be collected at the boundary scan output. With respect to operating with other devices, the boundary scan chain will not not be able to shift data through the processor to other devices.

IMPLICATION: This problem only affects operation of the component while in boundary scan mode.

WORKAROUND: The bypass mode for the processor does still work, so other devices in the boundary scan chain can be accessed using this method.

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

55. Incorrect Type Field in CPUID

PROBLEM: Execution of the CPUID with a ‘1’ in the EAX register does not return a value of ‘01’ (OverDrive Processor installed) in the type field (Bits 13.12). Instead it returns a ‘00’ (Primary Processor) in the type field.

IMPLICATION: BIOS or application software that relies on the type field to be '01' when an OverDrive processor is installed, may function improperly.

WORKAROUND: BIOS or application software should be written to function properly when '00' is returned in the type field and an OverDrive processor is present in the system.

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

56. Refer to Summary Table of Changes

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

PROBLEM: Please refer to Part II of this document. This errata entry for the Pentium OverDrive processor has been added for clarification of the issue as it applies to the 63- and 83-MHz Pentium OverDrive processors for Intel486™ processor-based systems specifically.

IMPLICATION: This erratum applies to the 63- and 83-MHz Pentium OverDrive processors only when issuing JTAG instructions. The 63- and 83-MHz Pentium OverDrive processors do not have a TRST# pin. Please refer to Part II of this document for more information.

WORKAROUND: Please refer to Part II of this document.

STATUS: For the steppings affected, please see the Summary of Changes table at the beginning of this section.

58. CLK Input Capacitance Exceeds Specification

PROBLEM: The input capacitance on the CLK pin is higher than the specification due to the ceramic package design. The specification for the 120/133-MHz Pentium OverDrive processor's input capacitance is 8pF whereas the measured value on the clock pin is 13pF. This added capacitance may cause excessive ringback on the CLK signal.

IMPLICATION: If the CLK signal going to the processor is also used as an input to another device, the ringback can cause the other device to detect an incorrect rising edge on CLK. This can cause the system to not boot or to hang. Only a small percentage of systems appear to be affected by the higher capacitance.

The only confirmed issue is related to some boards with an Intel 82430LX chipset. The 82434LX PCMC's HCLKA output is used to drive the clock to the processor. This clock signal may also be used as a feedback to the chipset to minimize the clock skew between the CPU and PCMC. The capacitance difference will create a reflection on the clock signal that, if sufficient in amplitude, will cause the PCMC to detect a false clock edge and the bus state machines to get out of synchronization causing the system to lock up. This condition is dependent upon not only the capacitance difference but also the physical board design and voltage threshold of the chipset.

WORKAROUND: If an end user experiences any problems, they should remove the Pentium OverDrive processor and install the original 60/66-MHz Pentium processor. The end user should then contact the Intel Customer Support Hotline listed on the OverDrive processor retail box for assistance.

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

59-61. Refer to Summary Table of Changes

SPECIFICATION CLARIFICATIONS

1. CACHE#, KEN# and BLAST# Behavior

If CACHE# is driven LOW during a read cycle, this implies that the processor intends to perform a linefill; however, this does not imply that BLAST# will be driven HIGH. As with the IntelDX2™ processor, KEN# controls the cacheability of data and can be used to refuse the linefill. If the processor requires only one transfer to get the required data, BLAST# will be driven LOW if KEN# is sampled HIGH.

2. Behavior of Writeback Support Pins in Writethrough Mode

Three new output pins are used to support the internal writeback cache of the Pentium OverDrive processor. These pins are HIT#, HITM# and CACHE#. The writeback mode of the processor can be enabled by driving the WB/WT# pin to the appropriate state during the falling edge of RESET. The clarification is as follows:

The three internal writeback support pins will be driven at all times regardless of the processor cache mode (enhanced (writeback) or standard (writethrough)). This implies that the HIT# and HITM# pins will always be driven HIGH in writethrough mode since there can be no writeback cycles. It also implies that in writethrough mode, the CACHE# pin will be toggled normally for reads and will be driven HIGH for all write cycles.

3. HITM# Deassertion Behavior

The HITM# pin is driven LOW in response to an external snoop that hits a modified line in the cache. It remains low until an appropriate action occurs (usually a writeback cycle) to maintain consistency with main memory. The clarification deals with the deassertion of HITM#.

Previously available documentation stated that HITM# will be driven inactive “after the last RDY#/BRDY# of the writeback cycle” and is “guaranteed to be deasserted before the next ADS#.” This means that HITM# can be deasserted anytime between these two points, and hardware should not assume that HITM# is deasserted on any specific CLK. It also implies that the system design should not depend on the Pentium OverDrive processor to deassert HITM# in the same clock the writeback Enhanced IntelDX2 processor.

4. Refer to Summary Table of Changes

5. SMIACT# Handling During Snoop Writeback

Please refer to Part II of this document. This errata entry for the 63 and 83-MHz Pentium OverDrive processors has been added for clarification of the issue as it applies to these OverDrive processors specifically. This erratum applies to the Pentium OverDrive processor in Enhanced (WB) bus mode only . Standard bus mode (WT) is not affected.

6-17. Refer to Summary Table of Changes

18. Cache Test Regs Are Modified During FLUSH#

Please refer to Part II of this document. This errata entry for the Pentium OverDrive processor has been added for clarification of the issue as it applies to these OverDrive processors specifically. This erratum applies to the the 63- and 83-MHz Pentium OverDrive processors in Enhanced (WB) bus mode only. Standard bus mode (WT) is not affected.

19-21. Refer to Summary Table of Changes

22. Pullup/Pulldown Resistor Values for Some Configuration Changes

Please refer to Part II of this document for more details. This specification clarification entry has been added for clarification as it applies to these OverDrive processors specifically. This specification clarification applies to the 120/133-MHz Pentium OverDrive processor only for the BUSCHK# signal. This specification clarification applies to the 125, 150, and 166-MHz Pentium OverDrive processors only for the BUSCHK# and BRDYC# signals.

23-24. Refer to Summary Table of Changes

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the Pentium OverDrive processor datasheets. All Documentation Changes will be incorporated into the appropriate documentation.

1-12. Refer to Summary Table of Changes

Appendix A

Pentium® Processor Related Technical Collateral

PUBLIC DOCUMENTATION

(Available from Intel Literature Center @ 1-800-548-4725)

Document Title	Order #
Pentium Processor Family Developer's Manual (3 Volume Set) - Volume 1: Pentium Processors (Databook) - Volume 2: 82496/82497/82498 Cache Controller and 82491/82492/82493 Cache SRAM (Databook) - Volume 3: Architecture and Programming Manual	241563 241428 241429 241430
Pentium® Processor Datasheet (75 MHz, 90 MHz, 100 MHz, 120 MHz, 133 MHz, 150 MHz, 166 MHz and 200 MHz)	241997
Pentium Processor Specification Update	242480
MultiProcessor Specification	242016
Pentium Processor Family Product Briefs	241561
Pentium Processor Performance Brief	241557
Pentium Processor Technical Overview	241610
Intel MMX™ Technology Overview	243081
Intel Architecture MMX™ Technology Programmer's Reference Manual	243007
Intel Architecture MMX™ Technology Developer's Manual	243006
Intel MMX™ Technology at a Glance	243100
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
Pentium Processor 3.3V Clock Driver Specifications	Contact your local Intel/Disti Sales Office
Pentium Processor 3.3V ASIC Interface Specification	Contact your local Intel/Disti Sales Office
Pentium Processor 3.3V Pipeline BSRAM Specification	Contact your local Intel/Disti Sales Office