Extended Temperature Pentium® Processor with MMX™ Technology

Advance Information Datasheet

Product Features

- Support for MMX™ Technology
- Low-Power 0.25 Micron Process Technology
  - —1.8 V Core Supply for HL-PBGA
  - —2.5 V I/O Interface
- 32-Bit CPU with 64-Bit Data Bus
- Fractional Bus Operation
  - —166-MHz Core/66-MHz Bus
- Superscalar Architecture
  - —Enhanced Pipelines
  - —Two Pipelined Integer Units Capable of Two Instructions/Clock
  - —Pipelined MMX Technology
  - —Pipelined Floating-Point Unit
- Separate Code and Data Caches
  - —16-Kbyte Code, 16-Kbyte Write-Back Data
  - —MESI Cache Protocol
- Compatible with Large Software Base
  - —MS-DOS*, Windows*, OS/2*, UNIX*
- 4-Mbyte Pages for Increased TLB Hit Rate
- IEEE 1149.1 Boundary Scan
- Advanced Design Features
  - —Deeper Write Buffers
  - —Enhanced Branch Prediction Feature
  - —Virtual Mode Extensions
- Internal Error Detection Features
- On-Chip Local APIC Controller
- Power Management Features
  - —System Management Mode
  - —Clock Control
- 352-ball HL-PBGA

Notice: This document contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice. Verify with your local Intel sales office that you have the latest datasheet before finalizing a design.

Order Number: 273232-001
February, 1999

Contents

1.0 Introduction 7

1.1 Processor Features 7

2.0 Architecture Overview 8

2.1 Pentium® Processor Family Architecture 8
2.2 Pentium® Processor with MMX™ Technology 11
2.2.1 Full Support for Intel MMX™ Technology 11
2.2.2 16-Kbyte Code and Data Caches 12
2.2.3 Improved Branch Prediction 12
2.2.4 Enhanced Pipeline 12
2.2.5 Deeper Write Buffers 12
2.3 0.25 Micron Technology 12

3.0 Extended Temperature Pentium® Processor with MMX™ Technology Packaging Information 13

3.1 Differences from Desktop Processors 13
3.2 HL-PBGA Pinout and Pin Descriptions 14
3.3 Design Notes 18
3.4 Pin Quick Reference 18
3.5 Bus Fraction (BF) Selection 24
3.6 The CPUID Instruction 25
3.7 Boundary Scan Chain List 27
3.8 Pin Reference Tables 28
3.9 Pin Grouping According to Function 30
3.10 Mechanical Specifications 31
3.10.1 HL-PBGA Package Mechanical Diagrams 31
3.11 Thermal Specifications 32
3.11.1 Measuring Thermal Values 32
3.11.2 Thermal Equations and Data 32
3.11.3 Airflow Calculations for Maximum and Typical Power 33
3.11.4 HL-PBGA Package Thermal Resistance Information 34

4.0 Extended Temperature Pentium® Processor with MMX™ Technology Electrical Specifications 35

4.1 Absolute Maximum Ratings 35
4.2 DC Specifications 35
4.2.1 Power Sequencing 35
4.3 AC Specifications 38
4.3.1 Power and Ground 38
4.3.2 Decoupling Recommendations 38
4.3.3 Connection Specifications 38
4.3.4 AC Timings 39
4.4 I/O Buffer Models 46
4.4.1 Buffer Model Parameters 48
4.5 Signal Quality Specifications 49
4.5.1 Overshoot 49

5.0 Extended Temperature Pentium® Processor with MMX™ Technology Errata Information 51
5.1 Nomenclature 51
5.2 Summary Table of Changes 52
5.2.1 Codes Used in Summary Table 52
5.2.2 Documentation Changes 55
6.0 Extended Temperature Pentium® Processor with MMX™ Technology Specification Changes 56
7.0 Extended Temperature Pentium® Processor with MMX™ Technology Errata 56
8.0 Extended Temperature Pentium® Processor with MMX™ Technology Specification Clarifications 80
9.0 Extended Temperature Pentium® Processor with MMX™ Technology Documentation Changes 86
10.0 Pentium® Processor Related Technical Collateral 88

Figures

1 Pentium® Processor with MMX™ Technology Block Diagram 10
2 HL-PBGA Package Top Side View 14
3 HL-PBGA Package Pin Side View 15
4 EAX Bit Assignments for CPUID 25
5 EDX Bit Assignments for CPUID 25
6 HL-PBGA Package Dimensions 31
7 Technique for Measuring T_C 33
8 Thermal Resistance vs. Airflow for HL-PBGA Package 34
9 Clock Waveform 44
10 Valid Delay Timings 44
11 Float Delay Timings 44
12 Setup and Hold Timings 45
13 Reset and Configuration Timings 45
14 Test Timings 46
15 Test Reset Timings 46
16 First Order Input Buffer Model 47
17 First Order Output Buffer Model 48
18 Pending Bus Cycle Timing Diagram 72
19 Snoop Writeback Cycle Timing Diagram 73

Tables

1 Signals Removed from the Extended Temperature Pentium® Processor with MMX™ Technology 13
2 Pin Cross Reference by Pin Name 16
3 No Connect, Power Supply and Ground Pin Cross Reference 17
4 Quick Pin Reference 18
5 Bus Frequency Selection 25
6 EDX Bit Assignment Definitions for CPUID 26
7 Output Pins 28
8 Input Pins 29
9 Input/Output Pins 30
10 Pin Functional Grouping 30
11 HL-PBGA Package Dimensions 31
12 Thermal Resistances for HL-PBGA Packages 34
13 Absolute Maximum Ratings 35
14 V_{CC} and T_{CASE} Specifications 36
15 DC Specifications 36
16 I_{CC} Specifications 36
17 Power Dissipation Requirements for Thermal Design 37
18 Input and Output Characteristics 37
19 Extended Temperature AC Specifications 40
20 APIC AC Specifications 43
21 Notes to Tables 19 and 20 43
22 Parameters Used in the Specification of the First Order Input Buffer Model 47
23 Parameters Used in the Specification of the First Order Output Buffer Model 48
24 Signal to Buffer Type 48
25 Preliminary Input, Output and Bidirectional Buffer Model Parameters for HL-PBGA Package 49
26 Input Buffer Model Parameters: D (Diodes) 49
27 Overshoot Specification Summary 50
28 Specification Changes 53
29 Errata 53
30 Specification Clarifications 55
31 Documentation Changes 55
32 Pentium® Processor Related Technical Collateral 88

1.0 Introduction

The Extended Temperature Pentium® Processor with MMX™ Technology extends the Pentium processor family, by providing additional performance for extended temperature applications. Furthermore, the Extended Temperature Pentium processor with MMX technology has superscalar architecture which can execute two instructions per clock cycle, and enhanced branch prediction and separate caches also increase performance. The pipelined floating-point unit delivers workstation level performance. Separate code and data caches reduce cache conflicts while remaining software transparent.

The Extended Temperature Pentium processor with MMX technology has 4.5 million transistors, is built on Intel's 0.25 micron manufacturing process technology and has full SL Enhanced power management features including System Management Mode (SMM) and clock control. The Extended Temperature Pentium processor with MMX technology is available in a 352-ball High-Thermal Low-Profile–Plastic Ball Grid Array (HL-PBGA). The HL-PBGA package allows designers to use surface mount technology to create small form-factor designs.

The additional SL Enhanced features and extended temperature HL-PBGA package make the Extended Temperature Pentium processor with MMX technology ideal for automotive multimedia designs.

1.1 Processor Features

The Extended Temperature Pentium processor with MMX technology has all the advanced architectural and internal features of the desktop version of the Pentium processor with MMX technology, except that several features have been eliminated. The differences are specified in "Differences from Desktop Processors" on page 13.

The Extended Temperature Pentium processor with MMX technology has several features which allow for automotive multimedia designs. These features include the following:

- 1.8 V core
- 2.5 V I/O buffer V_{CC3} inputs to reduce power consumption
- SL Enhanced feature set

This document should be used in conjunction with *Embedded Pentium® Processor Family Developer's Manual* (order number 273204).

2.0 Architecture Overview

The Extended Temperature Pentium processor with MMX technology extends the family of Pentium processors with MMX technology. It is binary compatible with the 8086/88, 80286, Intel386™ DX, Intel386 SX, Intel486™ SX, IntelDX2™, IntelDX4™, and Pentium processors with voltage reduction technology (75–150 MHz).

The Extended Temperature Pentium processor with MMX technology contains all of the features of previous Intel architecture processors and provides significant enhancements and additions, including the following:

- Support for MMX™ Technology
- Superscalar Architecture
- Enhanced Branch Prediction Algorithm
- Pipelined Floating-Point Unit
- Improved Instruction Execution Time
- Separate 16-Kbyte Code Cache and 16-Kbyte Data Cache
- Writeback MESI Protocol in the Data Cache
- 64-Bit Data Bus
- Enhanced Bus Cycle Pipelining
- Address Parity
- Internal Parity Checking
- Execution Tracing
- Performance Monitoring
- IEEE 1149.1 Boundary Scan
- System Management Mode
- Virtual Mode Extensions
- 0.25 Micron Process Technology
- SL Power Management Features
- Pool of Four Write Buffers Used by Both Pipes

2.1 Pentium® Processor Family Architecture

The application instruction set of the Pentium processor family includes the complete Intel486 CPU family instruction set with extensions to accommodate some of the additional functionality of the Pentium processors. All application software written for the Intel386 and Intel486 family microprocessors will run on the Pentium processors without modification. The on-chip memory management unit (MMU) is completely compatible with the Intel386 and Intel486 families of processors.

The Pentium processors implement several enhancements to increase performance. The two instruction pipelines and the floating-point unit on Pentium processors are capable of independent operation. Each pipeline issues frequently used instructions in a single clock. Together, the dual pipes can issue two integer instructions in one clock, or one floating-point instruction (under certain circumstances, two floating-point instructions) in one clock.

Branch prediction is implemented in the Pentium processors. To support this, Pentium processors implement two prefetch buffers, one that prefetches code in a linear fashion, and one that prefetches code according to the Branch Target Buffer (BTB) so that code is almost always prefetched before it is needed for execution.

The floating-point unit has been completely redesigned over the Intel486 processor. Faster algorithms provide up to 10x speed-up for common operations including add, multiply, and load.

Pentium processors include separate code and data caches integrated on-chip to meet performance goals. Each cache has a 32-byte line size and is 4-way set associative. Each cache has a dedicated Translation Lookaside Buffer (TLB) to translate linear addresses to physical addresses. The data cache is configurable to be writeback or writethrough on a line-by-line basis and follows the MESI protocol. The data cache tags are triple ported to support two data transfers and an inquire cycle in the same clock. The code cache is an inherently write-protected cache. The code cache tags are also triple ported to support snooping and split line accesses. Individual pages can be configured as cacheable or non-cacheable by software or hardware. The caches can be enabled or disabled by software or hardware.

The Pentium processors have increased the data bus to 64 bits to improve the data transfer rate. Burst read and burst writeback cycles are supported by the Pentium processors. In addition, bus cycle pipelining has been added to allow two bus cycles to be in progress simultaneously. The Pentium processors' MMU contains optional extensions to the architecture that allow 4-Kbyte and 4-Mbyte page sizes.

The Pentium processors have added significant data integrity and error detection capability. Data parity checking is still supported on a byte-by-byte basis. Address parity checking and internal parity checking features have been added along with a new exception, the machine check exception.

As more and more functions are integrated on-chip, the complexity of board level testing is increased. To address this, the Pentium processors have increased test and debug capability. The Pentium processors implement IEEE Boundary Scan (Standard 1149.1). In addition, the Pentium processors have specified four breakpoint pins that correspond to each of the debug registers and externally indicate a breakpoint match. Execution tracing provides external indications when an instruction has completed execution in either of the two internal pipelines, or when a branch has been taken.

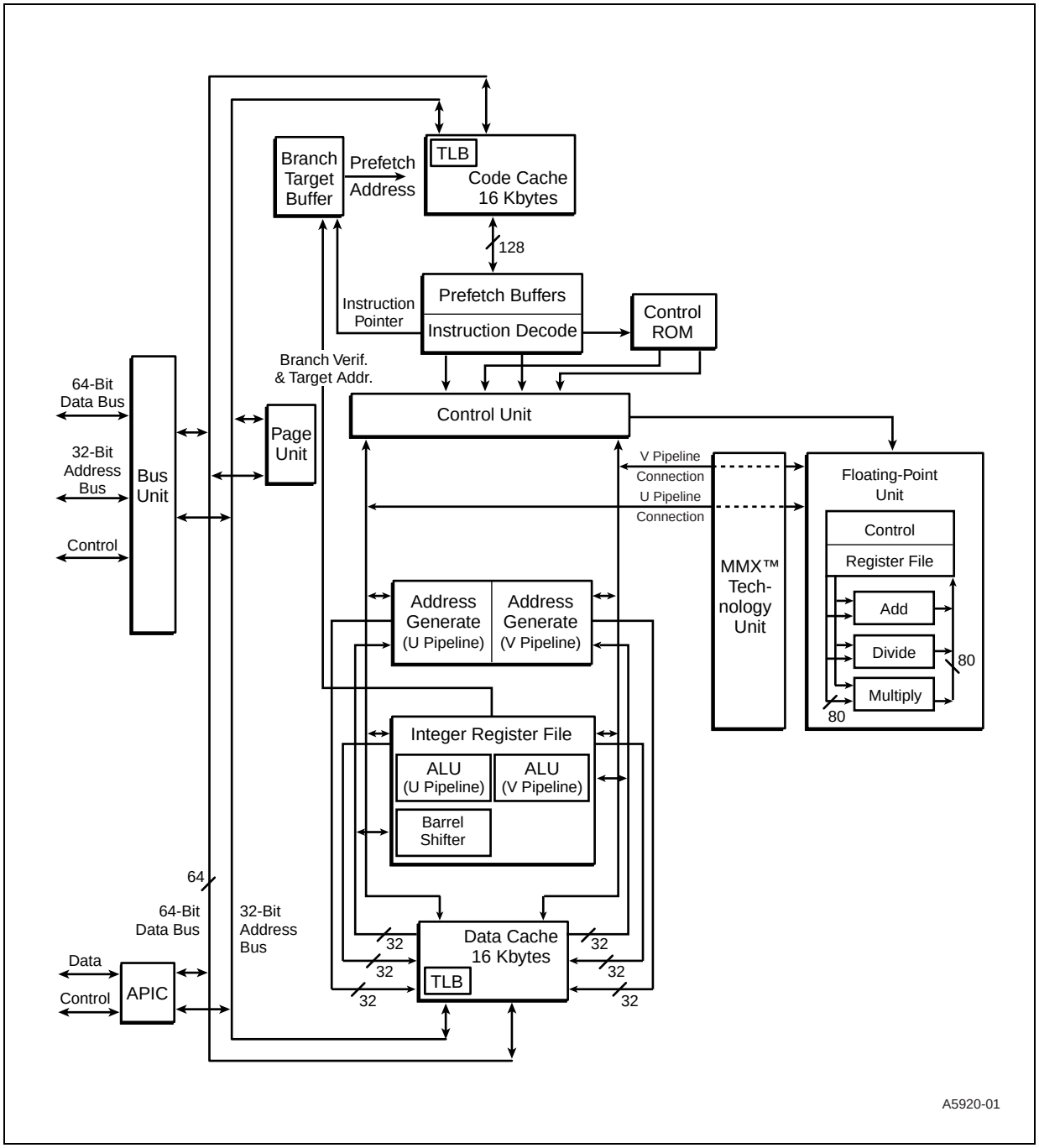
System Management Mode (SMM) has been implemented along with some extensions to the SMM architecture. Enhancements to virtual 8086 mode have been made to increase performance by reducing the number of times it is necessary to trap to a virtual 8086 monitor.

Figure 1 shows a block diagram of the Pentium processor with MMX technology.

The block diagram shows the two instruction pipelines, the "u" pipe and "v" pipe. The u-pipe can execute all integer and floating-point instructions. The v-pipe can execute simple integer instructions and the FXCH floating-point instructions.

The separate code and data caches are shown. The data cache has two ports, one for each of the two pipes (the tags are triple ported to allow simultaneous inquire cycles). The data cache has a dedicated Translation Lookaside Buffer (TLB) to translate linear addresses to the physical addresses used by the data cache.

Figure 1. Pentium® Processor with MMX™ Technology Block Diagram



The code cache, branch target buffer and prefetch buffers are responsible for getting raw instructions into the execution units of the Pentium processor. Instructions are fetched from the code cache or from the external bus. Branch addresses are remembered by the branch target buffer. The code cache TLB translates linear addresses to physical addresses used by the code cache.

The decode unit decodes the prefetched instructions so the Pentium processor can execute the instruction. The control ROM contains the microcode which controls the sequence of operations that must be performed to implement the Pentium processor architecture. The control ROM unit has direct control over both pipelines.

The Pentium processor contains a pipelined floating-point unit that provides a significant floating-point performance advantage over previous generations of processors.

In addition to the SMM features described above, the Pentium processor supports clock control. When the clock to the processor is stopped, power dissipation is virtually eliminated. The combination of these improvements makes the Pentium processor a good choice for low-power designs.

The Pentium processor supports fractional bus operation. This allows the internal processor core to operate at high frequencies, while communicating with the external bus at lower frequencies.

The Extended Temperature Pentium processor with MMX technology contains an on-chip advanced programmable interrupt controller (APIC). This function is reserved for future multi-processing function.

The architectural features introduced in this section are more fully described in the *Embedded Pentium® Processor Family Developer's Manual* (order number 273204).

2.2 Pentium® Processor with MMX™ Technology

The Pentium processor with MMX technology for high-performance extended temperature designs is a significant addition to the Pentium processor family. Available at 166 MHz, it is the first extended temperature microprocessor to support Intel MMX technology.

The Pentium processor with MMX technology is both software and pin compatible with previous members of the Pentium processor family. It contains 4.5 million transistors and is manufactured on 0.25 micron CMOS process, which allows voltage reduction technology for low power and high density.

In addition to the architecture described in the previous section for the Pentium processor family, the Pentium processor with MMX technology has several additional micro-architectural enhancements, which are described in the next section.

2.2.1 Full Support for Intel MMX™ Technology

MMX technology is based on the SIMD technique (Single Instruction, Multiple Data) which enables increased performance on a wide variety of multimedia and communications applications. Fifty-seven new instructions and four new 64-bit data types are supported in the Pentium processor with MMX technology. All existing operating system and application software are fully-compatible.

2.2.2 16-Kbyte Code and Data Caches

On-chip level-1 data and code cache sizes are 16 Kbytes each and are 4-way set associative on the Pentium processor with MMX technology. Large separate internal caches improve performance by reducing average memory access time and providing fast access to recently-used instructions and data. The instruction and data caches can be accessed simultaneously while the data cache supports two data references simultaneously. The data cache supports a write-back (or alternatively, write-through, on a line-by-line basis) policy for memory updates.

2.2.3 Improved Branch Prediction

Dynamic branch prediction uses the Branch Target Buffer (BTB) to boost performance by predicting the most likely set of instructions to be executed. The BTB has been improved on the Pentium processor with MMX technology to increase its accuracy. This processor has four prefetch buffers that can hold up to four successive code streams.

2.2.4 Enhanced Pipeline

An additional pipeline stage has been added and the pipeline has been enhanced to improve performance. The integration of the MMX technology pipeline with the integer pipeline is very similar to that of the floating-point pipeline. Under some circumstances, two MMX instructions or one integer and one MMX instruction can be paired and issued in one clock cycle to increase throughput. The enhanced pipeline is described in more detail in the *Embedded Pentium® Processor Family Developer's Manual* (order number 273204).

2.2.5 Deeper Write Buffers

A pool of four write buffers is now shared between the dual pipelines to improve memory write performance.

2.3 0.25 Micron Technology

The 0.25 micron technology is the state-of-the-art CMOS manufacturing process Intel unveiled on April 12, 1997, enabling the use of lower core supply to sub-2 V. As a result, the Extended Temperature Pentium processor with MMX technology consumes significantly less power at even higher speeds.

3.0 Extended Temperature Pentium® Processor with MMX™ Technology Packaging Information

3.1 Differences from Desktop Processors

The following features have been eliminated in the Extended Temperature Pentium processor with MMX technology: Upgrade, Dual Processing (DP), and Master/Checker functional redundancy.

Table 1 lists the corresponding pins that exist on the Pentium processor with MMX technology but have been removed on the Extended Temperature Pentium processor with MMX technology.

Table 1. Signals Removed from the Extended Temperature Pentium® Processor with MMX™ 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.

3.2 HL-PBGA Pinout and Pin Descriptions

Figure 2. HL-PBGA Package Top Side View

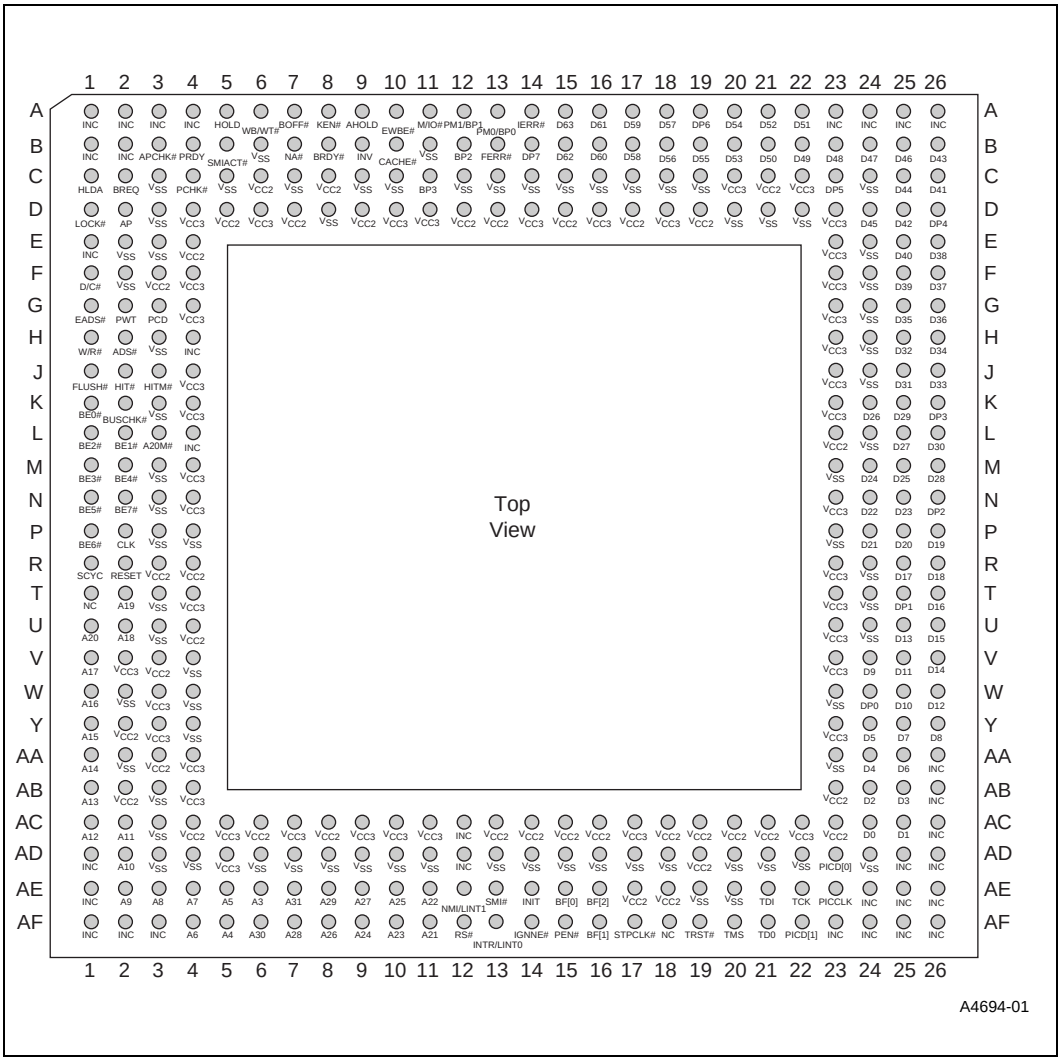


Figure 3. HL-PBGA Package Pin Side View

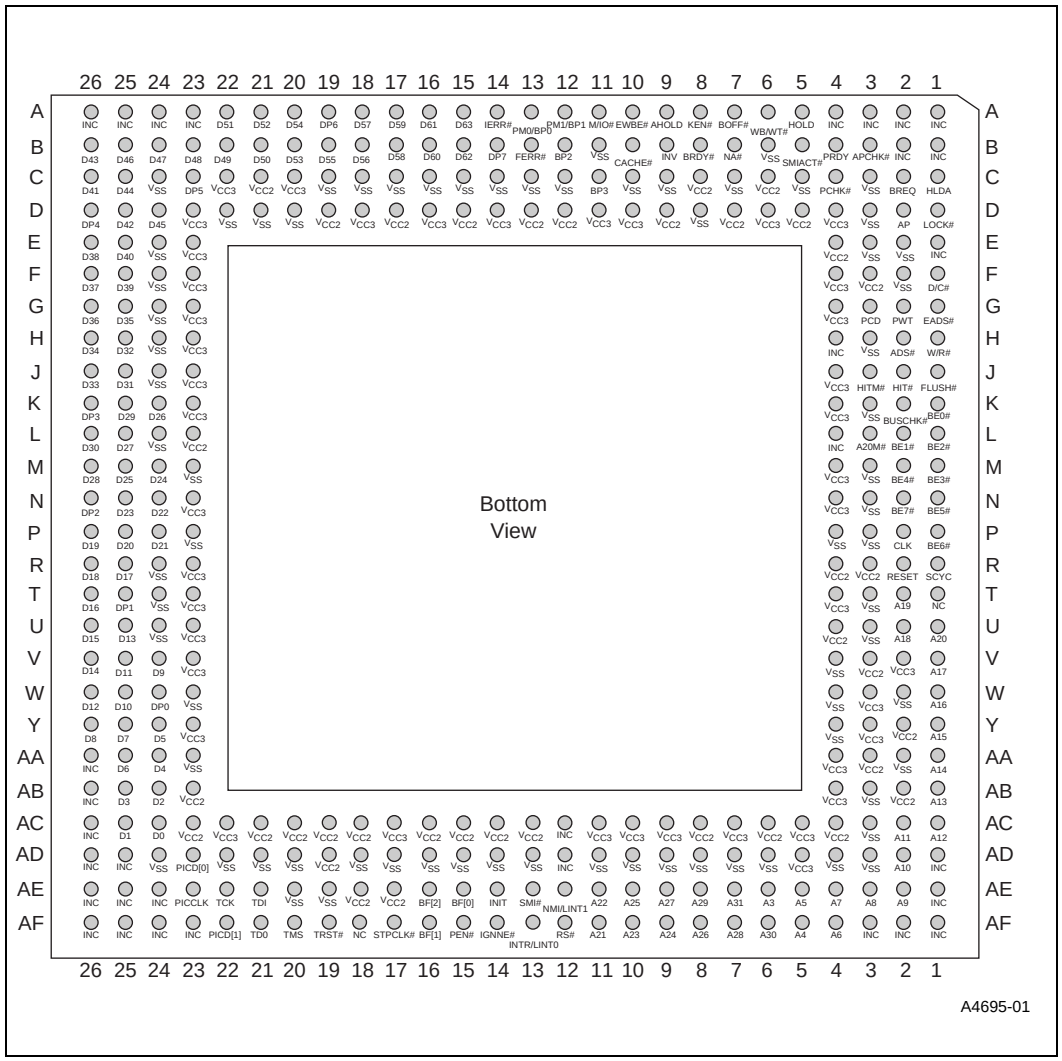


Table 2. Pin Cross Reference by Pin Name (Sheet 1 of 2)

Pin	Location	Pin	Location	Pin	Location	Pin	Location
Address							
A3	AE6	A11	AC2	A19	T2	A27	AE9
A4	AF5	A12	AC1	A20	U1	A28	AF7
A5	AE5	A13	AB1	A21	AF11	A29	AE8
A6	AF4	A14	AA1	A22	AE11	A30	AF6
A7	AE4	A15	Y1	A23	AF10	A31	AE7
A8	AE3	A16	W1	A24	AF9		
A9	AE2	A17	V1	A25	AE10		
A10	AD2	A18	U2	A26	AF8		
Data							
D0	AC24	D16	T26	D32	H25	D48	B23
D1	AC25	D17	R25	D33	J26	D49	B22
D2	AB24	D18	R26	D34	H26	D50	B21
D3	AB25	D19	P26	D35	G25	D51	A22
D4	AA24	D20	P25	D36	G26	D52	A21
D5	Y24	D21	P24	D37	F26	D53	B20
D6	AA25	D22	N24	D38	E26	D54	A20
D7	Y25	D23	N25	D39	F25	D55	B19
D8	Y26	D24	M24	D40	E25	D56	B18
D9	V24	D25	M25	D41	C26	D57	A18
D10	W25	D26	K24	D42	D25	D58	B17
D11	V25	D27	L25	D43	B26	D59	A17
D12	W26	D28	M26	D44	C25	D60	B16
D13	U25	D29	K25	D45	D24	D61	A16
D14	V26	D30	L26	D46	B25	D62	B15
D15	U26	D31	J25	D47	B24	D63	A15
Control							
A20M#	L3	BREQ	C2	HITM#	J3	PM1/BP1	A12
ADS#	H2	BUSCHK#	K2	HLDA	C1	PRDY	B4
AHOLD	A9	CACHE#	B10	HOLD	A5	PWT	G2
AP	D2	D/C#	F1	IERR#	A14	R/S#	AF12
APCHK#	B3	DP0	W24	IGNNE#	AF14	RESET	R2
BE0#	K1	DP1	T25	INIT	AE14	SCYC	R1
BE1#	L2	DP2	N26	INTR/ LINT0	AF13	SMI#	AE13
BE2#	L1	DP3	K26	INV	B9	SMIACT#	B5
BE3#	M1	DP4	D26	KEN#	A8	TCK	AE22
BE4#	M2	DP5	C23	LOCK#	D1	TDI	AE21
BE5#	N1	DP6	A19	M/IO#	A11	TDO	AF21
BE6#	P1	DP7	B14	NA#	B7	TMS	AF20
BE7#	N2	EADS#	G1	NMI/LINT1	AE12	TRST#	AF19
BOFF#	A7	EWBE#	A10	PCD	G3	W/R#	H1
BP2	B12	FERR#	B13	PCHK#	C4	WB/WT#	A6

Table 2. Pin Cross Reference by Pin Name (Sheet 2 of 2)

Pin	Location	Pin	Location	Pin	Location	Pin	Location
BP3	C11	FLUSH#	J1	PEN#	AF15		
BRDY#	B8	HIT#	J2	PM0/BP0	A13		
APIC							
PICCLK	AE23	PICD0	AD23	PICD1 [APICEN]	AF22		
Clock Control							
BF0	AE15	BF1	AF16	BF2	AE16	CLK	P2
STPCLK#	AF17						

Table 3. No Connect, Power Supply and Ground Pin Cross Reference

V_{CC2}							
C6	D9	D19	R4	AB2	AC13	AC19	AE17
C8	D12	E4	U4	AB23	AC14	AC20	AE18
C21	D13	F3	V3	AC4	AC15	AC21	
D5	D15	L23	Y2	AC6	AC16	AC23	
D7	D17	R3	AA3	AC8	AC18	AD19	
V_{CC3}							
C20	D14	F23	J23	N23	V2	AA4	AC10
C22	D16	G4	K4	R23	V23	AB4	AC11
D4	D18	G23	K23	T4	W3	AC5	AC17
D6	D23	H23	M4	T23	Y3	AC7	AC22
D10	E23	J4	N4	U23	Y23	AC9	AD5
D11	F4						
V_{SS}							
B6	C14	D20	H3	P4	W4	AD6	AD16
B11	C15	D21	H24	P23	W23	AD7	AD17
C3	C16	D22	J24	R24	Y4	AD8	AD18
C5	C17	E2	K3	T3	AA2	AD9	AD20
C7	C18	E3	L24	T24	AA23	AD10	AD21
C9	C19	E24	M3	U3	AB3	AD11	AD22
C10	C24	F2	M23	U24	AC3	AD13	AD24
C12	D3	F24	N3	V4	AD3	AD14	AE19
C13	D8	G24	P3	W2	AD4	AD15	AE20
No Connect (NC)							
AF18	T1						
Internal No Connect (INC)							
A1	A23	B1	L4	AC26	AD26	AE26	AF23
A2	A24	B2	AA26	AD1	AE1	AF1	AF24
A3	A25	E1	AB26	AD12	AE24	AF2	AF25
A4	A26	H4	AC12	AD25	AE25	AF3	AF26

3.3 Design Notes

For reliable operation, always connect unused inputs to an appropriate signal level. Unused active low inputs should be connected to V_{CC3}. Unused active high inputs should be connected to GND (V_{SS}).

No Connect (NC) pins must remain unconnected. Connection of NC pins may result in component failure or incompatibility with processor steppings.

3.4 Pin Quick Reference

This section gives a brief functional description of each pin. For a detailed description, see the Hardware Interface chapter in the *Embedded Pentium® Processor Family Developer's Manual*.

Note: All input pins must meet their AC/DC specifications to guarantee proper functional behavior.

The # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When a # symbol is not present after the signal name, the signal is active, or asserted at the high voltage level. Square brackets around a signal name indicate that the signal is defined only at RESET.

The pins are classified as Input or Output based on their function in Master Mode. See the Error Detection chapter of the *Embedded Pentium® Processor Family Developer's Manual* (order number 273204) for further information.

Table 4. Quick Pin Reference (Sheet 1 of 6)

Symbol	Type	Name and Function
A20M#	I	When the **address bit 20 mask** pin is asserted, the Pentium® processor with MMX™ technology emulates the address wraparound at 1 Mbyte, which occurs on the 8086. When A20M# is asserted, the processor masks physical address bit 20 (A20) before performing a lookup to the internal caches or driving a memory cycle on the bus. The effect of A20M# is undefined in protected mode. A20M# must be asserted only when the processor is in real mode.
A31–A3	I/O	As outputs, the **address** lines of the processor along with the byte enables define the physical area of memory or I/O accessed. The external system drives the inquire address to the processor on A31–A5.
ADS#	O	The **address status** indicates that a new valid bus cycle is currently being driven by the processor.
AHOLD	I	In response to the assertion of **address hold**, the processor will stop driving the address lines (A31–A3) and AP in the next clock. The rest of the bus will remain active so data can be returned or driven for previously issued bus cycles.
AP	I/O	**Address parity** is driven by the processor with even parity information on all processor generated cycles in the same clock that the address is driven. Even parity must be driven back to the processor during inquire cycles on this pin in the same clock as EADS# to ensure that correct parity check status is indicated.
APCHK#	O	The **address parity check** status pin is asserted two clocks after EADS# is sampled active if the processor has detected a parity error on the address bus during inquire cycles. APCHK# will remain active for one clock each time a parity error is detected.
BE7#–BE5# BE4#–BE0#	O I/O	The **byte enable** pins are used to determine which bytes must be written to external memory, or which bytes were requested by the CPU for the current cycle. The byte enables are driven in the same clock as the address lines (A31-3).

Table 4. Quick Pin Reference (Sheet 2 of 6)

Symbol	Type	Name and Function
BF2–BF0	I	The **Bus Frequency** pins determine the bus-to-core frequency ratio. BF [2:0] are sampled at RESET, and cannot be changed until another non-warm (1 ms) assertion of RESET. Additionally, BF[2:0] must not change values while RESET is active. See Table 5 for Bus Frequency Selection. In order to override the internal defaults and guarantee that the BF[2:0] inputs remain stable while RESET is active, these pins should be strapped directly to or through a pullup/pulldown resistor to V_{CC3} or ground. Driving these pins with active logic is not recommended unless stability during RESET can be guaranteed. During power up, RESET should be asserted prior to or ramped simultaneously with the core voltage supply to the processor.
BOFF#	I	The **backoff** input is used to abort all outstanding bus cycles that have not yet completed. In response to BOFF#, the processor will float all pins normally floated during bus hold in the next clock. The processor remains in bus hold until BOFF# is negated, at which time the processor restarts the aborted bus cycle(s) in their entirety.
[APICEN] PICD1	I	**Advanced Programmable Interrupt Controller Enable** enables or disables the on-chip APIC interrupt controller. If sampled high at the falling edge of RESET, the APIC is enabled. APICEN shares a pin with the PICD1 signal.
BP3–BP2 PM/BP1–BP0	O	The **breakpoint** pins (BP3–0) correspond to the debug registers, DR3–DR0. These pins externally indicate a breakpoint match when the debug registers are programmed to test for breakpoint matches. BP1 and BP0 are multiplexed with the **performance monitoring** pins (PM1 and PM0). The PB1 and PB0 bits in the Debug Mode Control Register determine if the pins are configured as breakpoint or performance monitoring pins. The pins come out of RESET configured for performance monitoring.
BRDY#	I	The **burst ready** input indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the processor data in response to a write request. This signal is sampled in the T2, T12 and T2P bus states.
BREQ	O	The **bus request** output indicates to the external system that the processor has internally generated a bus request. This signal is always driven whether or not the processor is driving its bus.
BUSCHK#	I	The **bus check** input allows the system to signal an unsuccessful completion of a bus cycle. If this pin is sampled active, the processor will latch the address and control signals in the machine check registers. If, in addition, the MCE bit in CR4 is set, the processor will vector to the machine check exception. To assure that BUSCHK# will always be recognized, STPCLK# must be deasserted any time BUSCHK# is asserted by the system, before the system allows another external bus cycle. If BUSCHK# is asserted by the system for a snoop cycle while STPCLK# remains asserted, usually (if MCE=1) the processor will vector to the exception after STPCLK# is deasserted. But if another snoop to the same line occurs during STPCLK# assertion, the processor can lose the BUSCHK# request.
CACHE#	O	For processor-initiated cycles, the **cache** pin indicates internal cacheability of the cycle (if a read), and indicates a burst writeback cycle (if a write). If this pin is driven inactive during a read cycle, the processor will not cache the returned data, regardless of the state of the KEN# pin. This pin is also used to determine the cycle length (number of transfers in the cycle).
CLK	I	The **clock** input provides the fundamental timing for the processor. Its frequency is the operating frequency of the processor external bus and requires TTL levels. All external timing parameters except TDI, TDO, TMS, TRST# and PICD0–1 are specified with respect to the rising edge of CLK. This pin is 2.5 V-tolerant-only on the Extended Temperature Pentium processor with MMX technology. It is recommended that CLK begin 150 ms after V_{CC} reaches its proper operating level. This recommendation is only to assure the long term reliability of the device.

Table 4. Quick Pin Reference (Sheet 2 of 6)

Symbol	Type	Name and Function
BF2–BF0	I	The **Bus Frequency** pins determine the bus-to-core frequency ratio. BF [2:0] are sampled at RESET, and cannot be changed until another non-warm (1 ms) assertion of RESET. Additionally, BF[2:0] must not change values while RESET is active. See Table 5 for Bus Frequency Selection. In order to override the internal defaults and guarantee that the BF[2:0] inputs remain stable while RESET is active, these pins should be strapped directly to or through a pullup/pulldown resistor to V_{CC3} or ground. Driving these pins with active logic is not recommended unless stability during RESET can be guaranteed. During power up, RESET should be asserted prior to or ramped simultaneously with the core voltage supply to the processor.
BOFF#	I	The **backoff** input is used to abort all outstanding bus cycles that have not yet completed. In response to BOFF#, the processor will float all pins normally floated during bus hold in the next clock. The processor remains in bus hold until BOFF# is negated, at which time the processor restarts the aborted bus cycle(s) in their entirety.
[APICEN] PICD1	I	**Advanced Programmable Interrupt Controller Enable** enables or disables the on-chip APIC interrupt controller. If sampled high at the falling edge of RESET, the APIC is enabled. APICEN shares a pin with the PICD1 signal.
BP3–BP2 PM/BP1–BP0	O	The **breakpoint** pins (BP3–0) correspond to the debug registers, DR3–DR0. These pins externally indicate a breakpoint match when the debug registers are programmed to test for breakpoint matches. BP1 and BP0 are multiplexed with the **performance monitoring** pins (PM1 and PM0). The PB1 and PB0 bits in the Debug Mode Control Register determine if the pins are configured as breakpoint or performance monitoring pins. The pins come out of RESET configured for performance monitoring.
BRDY#	I	The **burst ready** input indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the processor data in response to a write request. This signal is sampled in the T2, T12 and T2P bus states.
BREQ	O	The **bus request** output indicates to the external system that the processor has internally generated a bus request. This signal is always driven whether or not the processor is driving its bus.
BUSCHK#	I	The **bus check** input allows the system to signal an unsuccessful completion of a bus cycle. If this pin is sampled active, the processor will latch the address and control signals in the machine check registers. If, in addition, the MCE bit in CR4 is set, the processor will vector to the machine check exception. To assure that BUSCHK# will always be recognized, STPCLK# must be deasserted any time BUSCHK# is asserted by the system, before the system allows another external bus cycle. If BUSCHK# is asserted by the system for a snoop cycle while STPCLK# remains asserted, usually (if MCE=1) the processor will vector to the exception after STPCLK# is deasserted. But if another snoop to the same line occurs during STPCLK# assertion, the processor can lose the BUSCHK# request.
CACHE#	O	For processor-initiated cycles, the **cache** pin indicates internal cacheability of the cycle (if a read), and indicates a burst writeback cycle (if a write). If this pin is driven inactive during a read cycle, the processor will not cache the returned data, regardless of the state of the KEN# pin. This pin is also used to determine the cycle length (number of transfers in the cycle).
CLK	I	The **clock** input provides the fundamental timing for the processor. Its frequency is the operating frequency of the processor external bus and requires TTL levels. All external timing parameters except TDI, TDO, TMS, TRST# and PICD0–1 are specified with respect to the rising edge of CLK. This pin is 2.5 V-tolerant-only on the Extended Temperature Pentium processor with MMX technology. It is recommended that CLK begin 150 ms after V_{CC} reaches its proper operating level. This recommendation is only to assure the long term reliability of the device.

Table 4. Quick Pin Reference (Sheet 3 of 6)

Symbol	Type	Name and Function
D/C#	O	The **data/code** output is one of the primary bus cycle definition pins. It is driven valid in the same clock as the ADS# signal is asserted. D/C# distinguishes between data and code or special cycles.
D63–D0	I/O	These are the 64 **data lines** for the processor. Lines D7–D0 define the least significant byte of the data bus; lines D63–D56 define the most significant byte of the data bus. When the CPU is driving the data lines, they are driven during the T2, T12 or T2P clocks for that cycle. During reads, the CPU samples the data bus when BRDY# is returned.
DP7–DP0	I/O	These are the **data parity** pins for the processor. There is one for each byte of the data bus. They are driven by the processor with even parity information on writes in the same clock as write data. Even parity information must be driven back to the Pentium processor with voltage reduction technology on these pins in the same clock as the data to ensure that the correct parity check status is indicated by the processor. DP7 applies to D63–D56; DP0 applies to D7–D0.
EADS#	I	This signal indicates that a valid **external address** has been driven onto the processor address pins to be used for an inquire cycle.
EWBE#	I	The **external write buffer empty** input, when inactive (high), indicates that a write cycle is pending in the external system. When the processor generates a write and EWBE# is sampled inactive, the processor will hold off all subsequent writes to all E- or M-state lines in the data cache until all write cycles have completed, as indicated by EWBE# being active.
FERR#	O	The **floating-point error** pin is driven active when an unmasked floating-point error occurs. FERR# is similar to the ERROR# pin on the Intel387™ math coprocessor. FERR# is included for compatibility with systems using MS-DOS type floating-point error reporting.
FLUSH#	I	When asserted, the **cache flush** input forces the processor to write back all modified lines in the data cache and invalidate its internal caches. A Flush Acknowledge special cycle will be generated by the processor indicating completion of the writeback and invalidation. If FLUSH# is sampled low when RESET transitions from high to low, three-state test mode is entered.
HIT#	O	The **hit** indication is driven to reflect the outcome of an inquire cycle. If an inquire cycle hits a valid line in either the data or instruction cache, this pin is asserted two clocks after EADS# is sampled asserted. If the inquire cycle misses the cache, this pin is negated two clocks after EADS#. This pin changes its value only as a result of an inquire cycle and retains its value between the cycles.
HITM#	O	The **hit to a modified line** output is driven to reflect the outcome of an inquire cycle. It is asserted after inquire cycles which resulted in a hit to a modified line in the data cache. It is used to inhibit another bus master from accessing the data until the line is completely written back.
HLDA	O	The **bus hold acknowledge** pin goes active in response to a hold request driven to the processor on the HOLD pin. It indicates that the processor has floated most of the output pins and relinquished the bus to another local bus master. When leaving bus hold, HLDA will be driven inactive and the processor will resume driving the bus. If the processor has a bus cycle pending, it will be driven in the same clock that HLDA is de-asserted.
HOLD	I	In response to the **bus hold request**, the processor will float most of its output and input/output pins and assert HLDA after completing all outstanding bus cycles. The processor will maintain its bus in this state until HOLD is de-asserted. HOLD is not recognized during LOCK cycles. The processor will recognize HOLD during reset.
IERR#	O	The **internal error** pin is used to indicate internal parity errors. If a parity error occurs on a read from an internal array, the processor will assert the IERR# pin for one clock and then shutdown.

Table 4. Quick Pin Reference (Sheet 4 of 6)

Symbol	Type	Name and Function
IGNNE#	I	This is the **ignore numeric error** input. This pin has no effect when the NE bit in CR0 is set to 1. When the CR0.NE bit is 0, and the IGNNE# pin is asserted, the processor will ignore any pending unmasked numeric exception and continue executing floating-point instructions for the entire duration that this pin is asserted. When the CR0.NE bit is 0, IGNNE# is not asserted, a pending unmasked numeric exception exists (SW.ES = 1), and the floating-point instruction is one of FINIT, FCLEX, FSTENV, FSAVE, FSTSW, FSTCW, FENI, FDISI, or FSETPM, the processor will execute the instruction in spite of the pending exception. When the CR0.NE bit is 0, IGNNE# is not asserted, a pending unmasked numeric exception exists (SW.ES = 1), and the floating-point instruction is one other than FINIT, FCLEX, FSTENV, FSAVE, FSTSW, FSTCW, FENI, FDISI, or FSETPM, the processor will stop execution and wait for an external interrupt.
INIT	I	The processor **initialization** input pin forces the processor to begin execution in a known state. The processor state after INIT is the same as the state after RESET except that the internal caches, write buffers, and floating-point registers retain the values they had prior to INIT. INIT may NOT be used in lieu of RESET after power up. If INIT is sampled high when RESET transitions from high to low, the processor will perform built-in self test prior to the start of program execution.
INTR	I	An active **maskable interrupt** input indicates that an external interrupt has been generated. If the IF bit in the EFLAGS register is set, the processor will generate two locked interrupt acknowledge bus cycles and vector to an interrupt handler after the current instruction execution is completed. INTR must remain active until the first interrupt acknowledge cycle is generated to assure that the interrupt is recognized.
INV	I	The **invalidation** input determines the final cache line state (S or I) in case of an inquire cycle hit. It is sampled together with the address for the inquire cycle in the clock EADS# is sampled active.
KEN#	I	The **cache enable** pin is used to determine whether the current cycle is cacheable or not and is consequently used to determine cycle length. When the processor generates a cycle that can be cached (CACHE# asserted) and KEN# is active, the cycle will be transformed into a burst line fill cycle.
LOCK#	O	The **bus lock** pin indicates that the current bus cycle is locked. The processor will not allow a bus hold when LOCK# is asserted (but AHOLD and BOFF# are allowed). LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after the BRDY# is returned for the last locked bus cycle. LOCK# is guaranteed to be de-asserted for at least one clock between back-to-back locked cycles.
M/IO#	O	The **memory/input-output** is one of the primary bus cycle definition pins. It is driven valid in the same clock as the ADS# signal is asserted. M/IO# distinguishes between memory and I/O cycles.
NA#	I	An active **next address** input indicates that the external memory system is ready to accept a new bus cycle although all data transfers for the current cycle have not yet completed. The processor will issue ADS# for a pending cycle two clocks after NA# is asserted. The processor supports up to two outstanding bus cycles.
NMI	I	The **non-maskable interrupt** request signal indicates that an external non-maskable interrupt has been generated.
PCD	O	The **page cache disable** pin reflects the state of the PCD bit in CR3; Page Directory Entry or Page Table Entry. The purpose of PCD is to provide an external cacheability indication on a page-by-page basis.
PCHK#	O	The **parity check** output indicates the result of a parity check on a data read. It is driven with parity status two clocks after BRDY# is returned. PCHK# remains low one clock for each clock in which a parity error was detected. Parity is checked only for the bytes on which valid data is returned.

Table 4. Quick Pin Reference (Sheet 5 of 6)

Symbol	Type	Name and Function
PEN#	I	The **parity enable** input (along with CR4.MCE) determines whether a machine check exception will be taken as a result of a data parity error on a read cycle. If this pin is sampled active in the clock, a data parity error is detected. The processor will latch the address and control signals of the cycle with the parity error in the machine check registers. If, in addition, the machine check enable bit in CR4 is set to "1", the processor will vector to the machine check exception before the beginning of the next instruction.
PICCLK	I	The APIC interrupt controller serial data bus clock is driven into the **programmable interrupt controller clock** input of the Pentium processor with MMX technology.
PICD0–PICD1 [APICEN]	I/O	**Programmable interrupt controller data lines 0–1** of the Pentium processor with MMX technology comprise the data portion of the APIC 3-wire bus. They are open-drain outputs that require external pull-up resistor. These signals are multiplexed with APICEN.
PM/BP[1:0]	O	These pins function as part of the performance monitoring feature. The breakpoint 1–0 pins are multiplexed with the **performance monitoring 1-0** pins. The PB1 and PB0 bits in the Debug Mode Control Register determine if the pins are configured as breakpoint or performance monitoring pins. The pins come out of RESET configured for performance monitoring.
PRDY	O	The **probe ready** output pin indicates that the processor has stopped normal execution in response to the R/S# pin going active or Probe Mode being entered.
PWT	O	The **page writethrough** pin reflects the state of the PWT bit in CR3, the page directory entry, or the page table entry. The PWT pin is used to provide an external writeback indication on a page-by-page basis.
R/S#	I	The **run/stop** input is provided for use with the Intel debug port. Please refer to the *Embedded Pentium® Processor Family Developer's Manual* (Order Number 273204) for more details.
RESET	I	**RESET** forces the processor to begin execution at a known state. All the processor internal caches will be invalidated upon the RESET. Modified lines in the data cache are not written back. FLUSH# and INIT are sampled when RESET transitions from high to low to determine if three-state test mode will be entered or if BIST will be run.
SCYC	O	The **split cycle** output is asserted during misaligned LOCKed transfers to indicate that more than two cycles will be locked together. This signal is defined for locked cycles only. It is undefined for cycles which are not locked.
SMI#	I	The **system management interrupt** causes a system management interrupt request to be latched internally. When the latched SMI# is recognized on an instruction boundary, the processor enters System Management Mode.
SMIACT#	O	An active **system management interrupt active** output indicates that the processor is operating in System Management Mode.
STPCLK#	I	Assertion of the **stop clock** input signifies a request to stop the internal clock of the Pentium processor with voltage reduction technology thereby causing the core to consume less power. When the CPU recognizes STPCLK#, the processor will stop execution on the next instruction boundary, unless superseded by a higher priority interrupt, and generate a Stop Grant Acknowledge cycle. When STPCLK# is asserted, the processor will still respond to external snoop requests.
TCK	I	The **testability clock** input provides the clocking function for the processor boundary scan in accordance with the IEEE Boundary Scan interface (Standard 1149.1). It is used to clock state information and data into and out of the processor during boundary scan.
TDI	I	The **test data input** is a serial input for the test logic. TAP instructions and data are shifted into the processor on the TDI pin on the rising edge of TCK when the TAP controller is in an appropriate state.

Table 4. Quick Pin Reference (Sheet 6 of 6)

Symbol	Type	Name and Function
TDO	O	The **test data output** is a serial output of the test logic. TAP instructions and data are shifted out of the processor on the TDO pin on TCK's falling edge when the TAP controller is in an appropriate state.
TMS	I	The value of the **test mode select** input signal sampled at the rising edge of TCK controls the sequence of TAP controller state changes.
TRST#	I	When asserted, the **test reset** input allows the TAP controller to be asynchronously initialized.
VCC2DET#	N/A	Differentiate between the Pentium Processor with MMX technology and the Extended Temperature Pentium processor with MMX technology. This is an Internal No Connect (INC) pin on the Extended Temperature Pentium processor with MMX technology. This pin is not defined on the HL-PBGA package.
V_{CC2}	I	These pins are the power inputs to the core: 1.9 V input for 166/266 MHz PPGA; 1.8 V for 166 MHz HL-PBGA; 2.0 V for 166 MHz HL-PBGA.
V_{CC3}	I	These pins are the 2.5 V power inputs to the I/O.
V_{SS}	I	These pins are the ground inputs.
W/R#	O	**Write/read** is one of the primary bus cycle definition pins. It is driven valid in the same clock as the ADS# signal is asserted. W/R# distinguishes between write and read cycles.
WB/WT#	I	The **writeback/writethrough** input allows a data cache line to be defined as writeback or writethrough on a line-by-line basis. As a result, it determines whether a cache line is initially in the S or E state in the data cache.

3.5 Bus Fraction (BF) Selection

Each Extended Temperature Pentium processor with MMX technology must be externally configured with the BF2–BF0 pins to operate in the specified bus fraction mode. Operation out of the specification is not supported. For example, a 166 MHz Extended Temperature Pentium processor with MMX technology supports only 2/5 mode.

The BF configuration pins are provided to select the allowable bus/core ratio of 2/5. The Extended Temperature Pentium processor with MMX technology multiplies the input CLK to achieve the higher internal core frequencies. The internal clock generator requires a constant frequency CLK input to within ±250 ps; therefore, the CLK input cannot be changed dynamically.

The external bus frequency is set during power-up Reset through the CLK pin. The Extended Temperature Pentium processor with MMX technology samples the BF0, BF1 and BF2 pins on the falling edge of RESET to determine which bus/core ratio to use.

Table 5 summarizes the operation of the BF pins on the Extended Temperature Pentium processor with MMX technology.

Note: BF pins must meet a 1 ms setup time to the falling edge of RESET and *must not change value while RESET is active*. Once a frequency is selected, it may not be changed with a warm reset. Changing this speed or ratio requires a "power on" RESET pulse initialization.

Table 5. Bus Frequency Selection

BF2	BF1	BF0	Bus/Core Ratio	Max Bus/Core Frequency (MHz)
0	0	0	2/5	66/166

NOTE: All other BF2–BF0 settings are reserved on the Extended Temperature Pentium processor with MMX technology.

3.6 The CPUID Instruction

The CPUID instruction allows software to determine the type and features of the processor on which it is executing. When executing CPUID, the Extended Temperature Pentium processor with MMX technology behaves like the Pentium processor and the Pentium processor with MMX technology as follows:

- If the value in EAX is '0', then the 12-byte ASCII string "Genuine Intel" (little endian) is returned in EBX, EDX and ECX. Also, a '1' is returned to EAX.
- If the value in EAX is '1', then the processor version is returned in EAX and the processor capabilities are returned in EDX. The values of EAX and EDX for the Extended Temperature Pentium processor with MMX technology are given below.
- If the value in EAX is neither '0' nor '1', the Extended Temperature Pentium processor with MMX technology writes '0' to all registers.

The following EAX and EDX values are defined for the CPUID instruction executed with EAX = '1'. The processor version EAX bit assignments are given in Figure 4. The EDX bit assignments are shown in Figure 5.

Figure 4. EAX Bit Assignments for CPUID

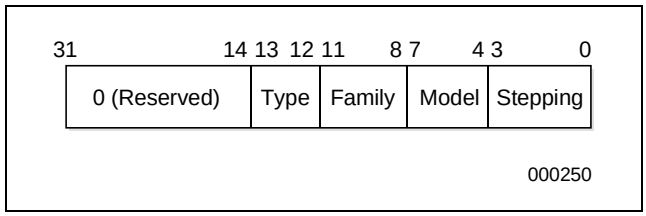
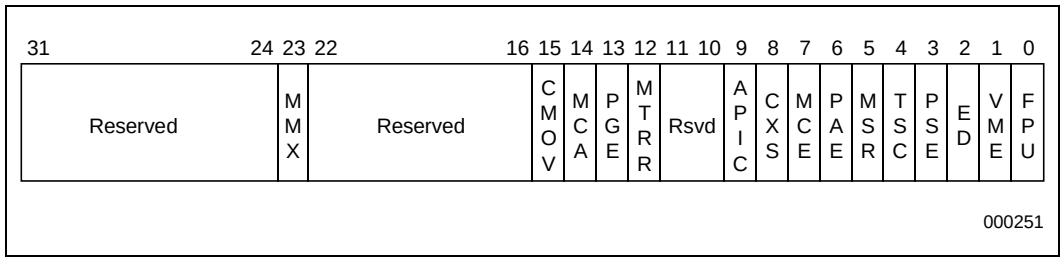


Figure 5. EDX Bit Assignments for CPUID



The type field for Extended Temperature Pentium processor with MMX technology is the same as Pentium processor with MMX technology (type = 00H). The family field is the same as all other Pentium processors (family = 5H). However, the model field is different: the Pentium processor model number is 2H, the Pentium processor with MMX technology model number is 4H, and the Extended Temperature Pentium processor with MMX technology model number is 8H. The stepping field indicates the revision number of a model. The stepping ID of A-step for the Extended Temperature Pentium processor with MMX technology is 1H. Stepping ID will be documented in the Extended Temperature Pentium processor with MMX technology stepping information.

After masking the reserve bits, all Extended Temperature Pentium processor with MMX technology-based products will get a value of 0x008003BF (assuming the APIC is enabled at boot), or 0x008001BF (when the APIC is disabled, using the APICEN boot pin) in EDX upon completion of the CPUID instruction.

Table 6. EDX Bit Assignment Definitions for CPUID

Bit	Value	Comments
0	1	FPU: Floating-point Unit on-chip
1	1	VME: Virtual-8086 Mode Enhancements
2	1	DE: Debugging Extensions
3	1	PSE: Page Size Extension
4	1	TSC: Time Stamp Counter
5	1	MSR Pentium® Processor MSR
6	0	PAE: Physical Address Extension
7	1	MCE: Machine Check Exception
8	1	CX8: CMPXCHG8B Instruction
9	1	APIC: APIC on-chip†
10–11	R	Reserved – Do not write to these bits or rely on their values
12	0	MTRR: Memory Type Range Registers
13	0	PGE: Page Global Enable
14	0	MCA: Machine Check Architecture
15–22	R	Reserved – Do not write to these bits or rely on their values
23	1	Intel Architecture with MMX™ technology supported
24–31	R	Reserved – Do not write to these bits or rely on their values

† Indicates that APIC is present and hardware enabled (software disabling does not affect this bit).

3.7 Boundary Scan Chain List

The boundary scan chain list for the Extended Temperature Pentium processor with MMX technology is different than the Pentium processor with MMX technology due to the removal of some pins. The boundary scan register for the Extended Temperature Pentium processor with MMX technology contains a cell for each pin. Following is the bit order of the Extended Temperature Pentium processor with MMX technology boundary scan register (left to right, top to bottom):

TDI → disapsba†, PICD1, PICD0, Reserved, PICCLK, D0, D1, D2, D3, D4, D5, D6, D7, DP0, D8, D9, D10, D11, D12, D13, D14, D15, DP1, D16, D17, D18, D19, D20, D21, D22, D23, DP2, D24, D25, D26, D27, D28, D29, D30, D31, DP3, D32, D33, D34, D35, D36, D37, D38, D39, DP4, D40, D41, D42, D43, D44, D45, D46, diswr†, D47, DP5, D48, D49, D50, D51, D52, D53, D54, D55, DP6, D56, D57, D58, D59, D60, D61, D62, D63, DP7, IERR#, FERR#, PM0BP0, PM1BP1, BP2, BP3, MIO#, CACHE#, EWBE#, INV, AHOLD, KEN#, BRDYC#, BRDY#, BOFF#, NA#, WBWT#, HOLD, disbus†, disbusl†, dismisc†, dismisca†, SMIACT#, PRDY, PCHK#, APCHK#, BREQ, HLDA, AP, LOCK#, PCD, PWT, DC#, EADS#, ADS#, HITM#, HIT#, WR#, BUSCHK#, FLUSH#, A20M#, BE0#, BE1#, BE2#, BE3#, BE4#, BE5#, BE6#, BE7#, SCYC, CLK, RESET, disabus†, A20, A19, A18, A17, A16, A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A31, A30, A29, A28, A27, A26, A25, A24, A23, A22, A21, NMI, RS#, INTR, SMI#, IGNNE#, INIT, PEN#, Reserved, BF0, BF1, BF2, STPCLK#, Reserved, Reserved, Reserved, Reserved → TDO

“Reserved” includes the no connect “NC” signals on the Extended Temperature Pentium processor with MMX technology.

The cells marked with a dagger (†) are control cells that are used to select the direction of bi-directional pins or three-state the output pins. If “1” is loaded into the control cell, the associated pin(s) are three-stated or selected as input. The following lists the control cells and their corresponding pins:

Disabus:	A31–A3, AP
Disbus:	BE7#–BE0#, CACHE#, SCYC, M/IO#, D/C#, W/R#, PWT, PCD
Disbusl:	ADS#, LOCK#, ADSC#
Dismisc:	APCHK#, PCHK#, PRDY, BP3, BP2, PM1/BP1, PM0/BP0, FERR#, SMIACT#, BREQ, HLDA, HIT#, HITM#
Dismisca:	IERR#
Diswr:	D63–D0, DP7–DP0
Disapsba:	PICD1–PICD0

3.8 Pin Reference Tables

Table 7. Output Pins

Name[1]	Active Level	When Floated
ADS#	Low	Bus Hold, BOFF#
APCHK#	Low	
BE7#–BE4#	Low	Bus Hold, BOFF#
BREQ	High	
CACHE#	Low	Bus Hold, BOFF#
FERR#	Low	
HIT#	Low	
HITM#[2]	Low	
HLDA	High	
IERR#	Low	
LOCK#	Low	Bus Hold, BOFF#
M/IO#, D/C#, W/R#	N/A	Bus Hold, BOFF#
PCHK#	Low	
BP3–BP2, PM1/BP1, PM0/BP0	High	
PRDY	High	
PWT, PCD	High	Bus Hold, BOFF#
SCYC	High	Bus Hold, BOFF#
SMIACT#	Low	
TDO	N/A	All states except Shift-DR and Shift-IR
VCC2DET#[3]	N/A	Differentiates between the Pentium® processor with MMX™ technology and the Extended Temperature Pentium processor with MMX technology

NOTE:
1. All output and input/output pins are floated during three-state test mode (except TDO).
2. HITM# pin has an internal pull-up resistor.
3. This pin is not on the HL-PBGA pinout.

Table 8. Input Pins

Name	Active Level	Synchronous/ Asynchronous	Internal Resistor	Qualified
A20M#	LOW	Asynchronous		
AHOLD	HIGH	Synchronous		
BF0	N/A	Synchronous/RESET	Pulldown	
BF1	N/A	Synchronous/RESET	Pullup	
BF2	N/A	Synchronous/RESET	Pulldown	
BOFF#	LOW	Synchronous		
BRDY#	LOW	Synchronous	Pullup	Bus State T2,T12,T2P
BUSCHK#	LOW	Synchronous	Pullup	BRDY#
CLK	N/A			
EADS#	LOW	Synchronous		
EWBE#	LOW	Synchronous		BRDY#
FLUSH#	LOW	Asynchronous		
HOLD	HIGH	Synchronous		
IGNNE#	LOW	Asynchronous		
INIT	HIGH	Asynchronous		
INTR	HIGH	Asynchronous		
INV	HIGH	Synchronous		EADS#
KEN#	LOW	Synchronous		First BRDY#/NA#
NA#	LOW	Synchronous		Bus State T2,TD,T2P
NMI	HIGH	Asynchronous		
PEN#	LOW	Synchronous		BRDY#
PICCLK	HIGH	Asynchronous	Pullup	
R/S#	N/A	Asynchronous	Pullup	
RESET	HIGH	Asynchronous		
SMI#	LOW	Asynchronous	Pullup	
STPCLK#	LOW	Asynchronous	Pullup	
TCK	N/A		Pullup	
TDI	N/A	Synchronous/TCK	Pullup	TCK
TMS	N/A	Synchronous/TCK	Pullup	TCK
TRST#	LOW	Asynchronous	Pullup	
WB/WT#	N/A	Synchronous		First BRDY#/NA#

Table 9. Input/Output Pins

Name	Active Level	When Floated(1)	Qualified (when an input)	Internal Resistor
A31–A3	N/A	Address Hold, Bus Hold, BOFF#	EADS#	
AP	N/A	Address Hold, Bus Hold, BOFF#	EADS#	
BE3#–BE0#	LOW	Bus Hold, BOFF#	RESET	Pulldown(2)
D63–D0	N/A	Bus Hold, BOFF#	BRDY#	
DP7–DP0	N/A	Bus Hold, BOFF#	BRDY#	
PICD0	N/A			Pullup
PICD1[APICEN]	N/A			Pulldown

NOTE:
1. All output and input/output pins are floated during three-state test mode (except TDO).
2. BE3#–BE0# have pulldowns during RESET only.

3.9 Pin Grouping According to Function

Table 10. Pin Functional Grouping

Function	Pins
Clock	CLK
Initialization	RESET, INIT, BF[2:0]
Address Bus	A31–A3, BE7#–BE0#
Address Mask	A20M#
Data Bus	D63–D0
Address Parity	AP, APCHK#
APIC Support	PICCLK, PICD0–PICD1
Data Parity	DP7–DP0, PCHK#, PEN#
Internal Parity Error	IERR#
System Error	BUSCHK#
Bus Cycle Definition	M/IO#, D/C#, W/R#, CACHE#, SCYC, LOCK#
Bus Control	ADS#, BRDY#, NA#
Page Cacheability	PCD, PWT
Cache Control	KEN#, WB/WT#
Cache Snooping/Consistency	AHOLD, EADS#, HIT#, HITM#, INV
Cache Flush	FLUSH#
Write Ordering	EWBE#
Bus Arbitration	BOFF#, BREQ, HOLD, HLDA
Interrupts	INTR, NMI
Floating-point Error Reporting	FERR#, IGNNE#
System Management Mode	SMI#, SMIACT#
TAP Port	TCK, TMS, TDI, TDO, TRST#
Breakpoint/Performance Monitoring	PM0/BP0, PM1/BP1, BP3–BP2
Clock Control	STPCLK#
Debugging	R/S#, PRDY

3.10 Mechanical Specifications

The HL-PBGA package of the Extended Temperature Pentium processor with MMX technology is a new package type for the Pentium processor family. Package summary information for the HL-PBGA device is provided in Table 11. Figure 6 shows the package dimensions.

3.10.1 HL-PBGA Package Mechanical Diagrams

Figure 6 shows the HL-PBGA package. The dimensions are listed in Table 11.

Figure 6. HL-PBGA Package Dimensions

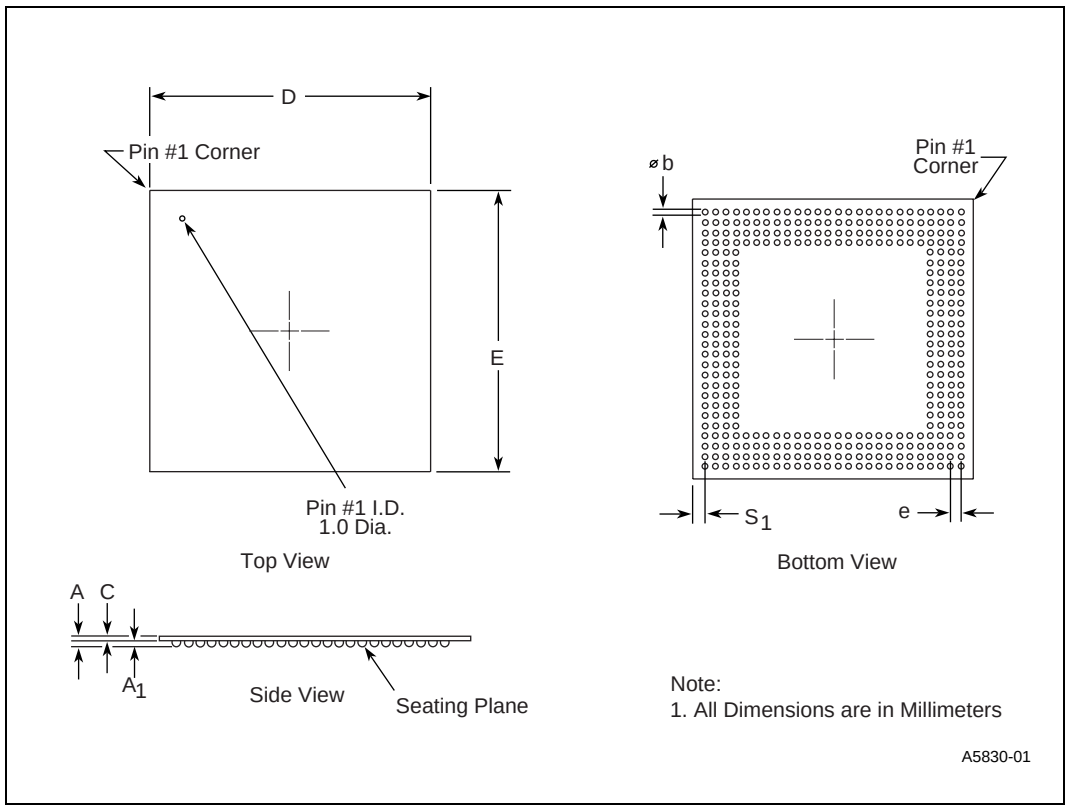


Table 11. HL-PBGA Package Dimensions (Sheet 1 of 2)

Symbol	Millimeters	
	Min	Max
A	1.41	1.67
A_1	0.56	0.70
b	0.60	0.90
c	0.85	0.97
D	34.90	35.10

Table 11. HL-PBGA Package Dimensions (Sheet 2 of 2)

Symbol	Millimeters	
	Min	Max
E	34.90	35.10
e	1.27	
S_1	1.63 REF	

3.11 Thermal Specifications

The Extended Temperature Pentium processor with MMX technology in the HL-PBGA package is specified for proper operation with a case temperature, T_{CASE} (T_C), range of – 40° C to 115° C .

3.11.1 Measuring Thermal Values

To verify that the proper T_C is maintained, it should be measured at the center of the package top surface (opposite of the pins). The measurement is made in the same way with or without a heatsink attached. When a heatsink is attached, a hole (smaller than 0.150" diameter) should be drilled through the heatsink to allow probing the center of the package. See Figure 7 for an illustration of how to measure T_C.

To minimize the measurement errors, it is recommended to use the following approach:

- Use 36-gauge or finer diameter K, T, or J type thermocouples. The laboratory testing was done using a thermocouple made by Omega* (part number 5TC-TTK-36-36).
- Attach the thermocouple bead or junction to the center of the package top surface using high thermal conductivity cements. The laboratory testing was done by using Omega Bond (part number OB-101).
- The thermocouple should be attached at a 90-degree angle as shown in Figure 7.
- The hole size should be smaller than 0.150" in diameter.
- Make sure there is no contact between thermocouple cement and heatsink base. The contact will affect the thermocouple reading.

3.11.2 Thermal Equations and Data

For the Extended Temperature Pentium processor with MMX technology, an ambient temperature, T_A (air temperature around the processor), is not specified directly. The only restriction is that T_C is met.

The equation used to calculate θ_{CA} is:

$$\theta_{CA} = \frac{T_C - T_A}{P}$$

Where:

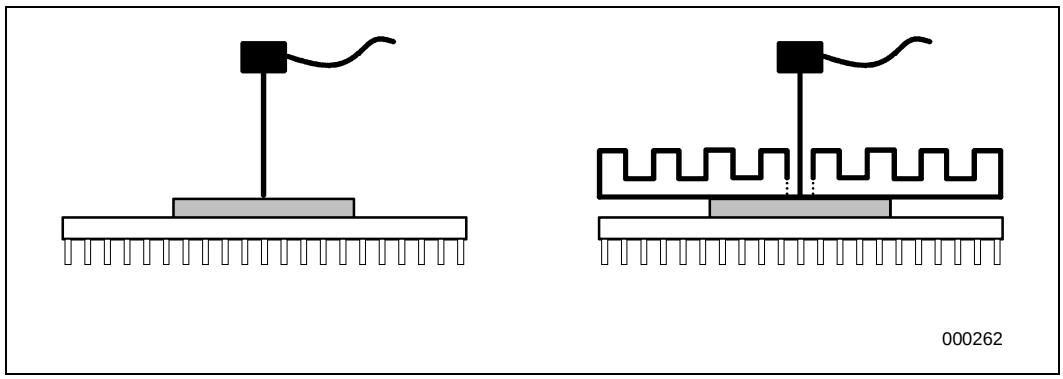
T_A and T_C = Ambient and case temperature (°C)

θ_{CA} = Case-to-ambient thermal resistance (°C/Watt)

P = Maximum power consumption (Watt)

θ_{JC} is thermal resistance from die to package case. θ_{JC} values shown in Table 12 are typical values. The actual θ_{JC} values depend on actual thermal conductivity and process of die attach. θ_{CA} is thermal resistance from package case to the ambient. θ_{CA} values shown in these tables are typical values. The actual θ_{CA} values depend on the heatsink design, interface between heatsink and package, airflow in the system, and thermal interactions between processor and surrounding components through PCB and the ambient.

Figure 7. Technique for Measuring T_C



3.11.3 Airflow Calculations for Maximum and Typical Power

Below is an example of determining the airflow required during maximum power consumption for the 166 MHz Extended Temperature Pentium processor with MMX technology assuming an ambient air temperature of +85° C:

T_C = 115° C

T_A = 85° C

$P_{HL\text{-}PBGA}$ = 4.1 W

θ_{CA} (HL-PBGA, without heat sink) = 7.3° C/W

Figure 8 indicates that this example would require about 600 LFM without a heat sink, and about 150 LFM with a heat sink in the vertical orientation.

Below is an example of determining the airflow required during typical power consumption for the 166 MHz Extended Temperature Pentium processor with MMX technology assuming an ambient air temperature of +85° C:

T_C = 115° C

T_A = 85° C

$P_{HL\text{-}PBGA}$ = 2.9 W

θ_{CA} (HL-PBGA, without heat sink) = 10.34° C/W

Figure 8 indicates that this example would require about 200 LFM without a heat sink, and about 25 LFM with a heat sink in vertical orientation, for typical power and 85° C ambient conditions.

3.11.4 HL-PBGA Package Thermal Resistance Information

Table 12 lists the θ_{JC} values for the Extended Temperature Pentium processor with MMX technology in the HL-PBGA package.

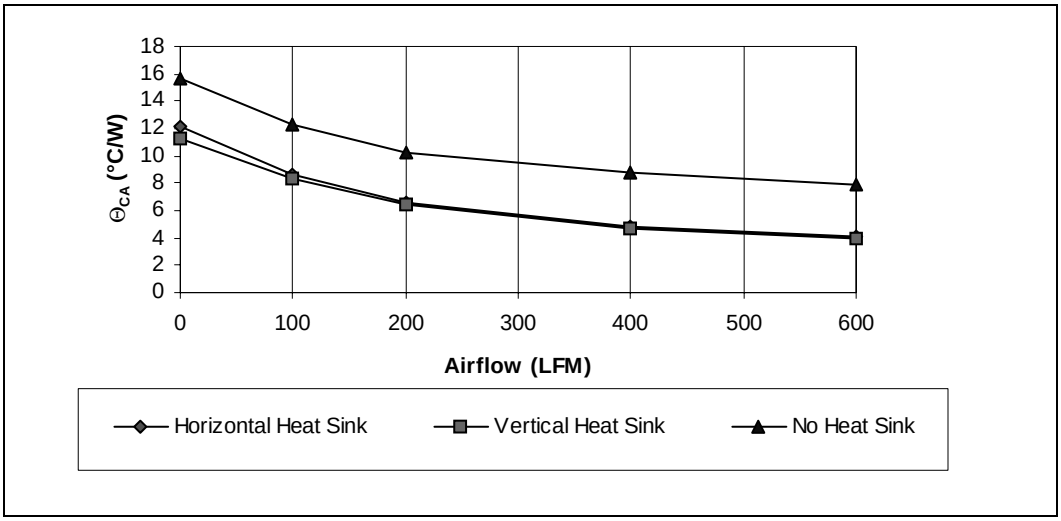
The thermal data collection conditions were:

- A bidirectional anodized aluminum alloy heat sink was used.
- Heat sink height was 7mm.
- In the horizontal orientation the component was mounted flush with the motherboard.
- In the vertical orientation the component was mounted on an add-in card perpendicular to the motherboard.

Table 12. Thermal Resistances for HL-PBGA Packages

Heatsink/ Orientation	θ_{JC} (°C/watt)	θ_{CA} (°C/watt) vs. Laminar Airflow (linear ft/min)				
		0	100	200	400	600
No Heat Sink	0.76	15.66	12.33	10.3	8.85	7.89
Horizontal	0.76	12.09	8.57	6.52	4.82	4.06
Vertical	0.76	11.33	8.34	6.38	4.69	3.95

Figure 8. Thermal Resistance vs. Airflow for HL-PBGA Package



4.0 Extended Temperature Pentium® Processor with MMX™ Technology Electrical Specifications

This section contains preliminary information on new products in production. The specifications are subject to change without notice.

4.1 Absolute Maximum Ratings

Warning: The following values are stress ratings only. Functional operation at the maximum ratings is not implied nor guaranteed. Functional operating conditions are given in the AC and DC specification tables. Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage.

Extended exposure to the maximum ratings may affect device reliability. Furthermore, although the Pentium processor with MMX technology contains protective circuitry to resist damage from Electrostatic Discharge (ESD), always take precautions to avoid high static voltages or electric fields.

Table 13. Absolute Maximum Ratings

Parameter	Maximum Rating
Case temperature under bias	–65° C to 125° C
Storage temperature	–65° C to 150° C
V_{CC3} supply voltage with respect to V_{SS}	–0.5 V to +3.2 V
V_{CC2} supply voltage with respect to V_{SS}	–0.5 V to +2.8 V
2.5 V only buffer DC input voltage	–0.5 V to V_{CC3}+0.5 V (not to exceed V_{CC3} max)

4.2 DC Specifications

Tables 15, 16, 17 and 18 list the DC specifications which apply to the Extended Temperature Pentium processor with MMX technology.

4.2.1 Power Sequencing

There is no specific sequence required for powering up or powering down the V_{CC2} and V_{CC3} power supplies. However, it is recommended that the V_{CC2} and V_{CC3} power supplies be either both ON or both OFF within one second of each other.

The I/O voltage V_{CC3} is 2.5 V. The core voltage V_{CC2} for the HL-PBGA package type is 1.8 V (166 MHz).

Table 14. V_{CC} and T_{CASE} Specifications

Package	T_{CASE}	Supply	Min Voltage	Max Voltage	Voltage Tolerance	Frequency
HL-PBGA	-40°C to 115°C	V_{CC2}	1.665 V	1.935 V	1.8 V ± 7.5%	166 MHz
		V_{CC3}	2.375 V	2.625 V	2.5 V ± 5%	166 MHz

Table 15. DC Specifications

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL3}	Input Low Voltage	–0.3	0.5	V	
V_{IH3}	Input High Voltage	$V_{CC3} - 0.7$	$V_{CC3} + 0.3$	V	TTL Level
V_{OL3}	Output Low Voltage		0.4	V	TTL Level, (1)
V_{OH3}	Output High Voltage	$V_{CC3} - 0.4$ $V_{CC3} - 0.2$		V V	TTL Level, (2) TTL Level, (3)

NOTES:
1. Parameter measured at -4 mA.
2. Parameter measured at 3 mA.
3. Parameter measured at 1 mA; not 100% tested, guaranteed by design.

The values in Table 16 should be used for power supply design. The values were determined using a worst case instruction mix and maximum V_{CC}. Power supply transient response and decoupling capacitors must be sufficient to handle the instantaneous current changes occurring during transitions from Stop Clock to full Active modes.

Table 16. I_{CC} Specifications

Symbol	Parameter	Min	Max	Unit	Notes
I_{CC2}	Power Supply Current		2.35 (HL-PBGA)	A	166 MHz
I_{CC3}	Power Supply Current		0.38	A	166 MHz

Table 17. Power Dissipation Requirements for Thermal Design

Parameter	Typical(1)	Max(2)	Unit	Frequency
Thermal Design Power		4.1	Watts	166 MHz
Active Power(3)	2.9		Watts	166 MHz
Stop Grant/Auto Halt Powerdown Power Dissipation(4)		0.70	Watts	166 MHz
Stop Clock Power(5)		0.06	Watts	166 MHz

NOTES:
1. This is the typical power dissipation in a system. This value is expected to be the average value that will be measured in a system using a typical device at the specified voltage running typical applications. This value is dependent upon the specific system configuration. Typical power specifications are not tested.
2. Systems must be designed to thermally dissipate the maximum thermal design power unless the system uses thermal feedback to limit processor's maximum power. The maximum thermal design power is determined using a worst-case instruction mix and also takes into account the thermal time constant of the package.
3. Active power is the average power measured in a system using a typical device running typical applications under normal operating conditions at nominal V_{CC} and room temperature.
4. Stop Grant/Auto Halt Powerdown Power Dissipation is determined by asserting the STPCLK# pin or executing the HALT instruction. When in this mode, the processor has a new feature which allows it to power down additional circuitry to enable lower power dissipation. This is the power without snooping at the specified voltage and with TR12 bit 21 set. In order to enable this feature, TR12 bit 21 must be set to 1 (the default is 0 or disabled). Stop grant/Auto Halt Powerdown power dissipation without TR12 bit 21 set may be higher. The Max rating may be changed in future specification updates.
5. Stop Clock Power Dissipation is determined by asserting the STPCLK# pin and then removing the external CLK input. This is specified at a T_{CASE} of 50 °C.

Table 18. Input and Output Characteristics

Symbol	Parameter	Min	Max	Unit	Notes
C_{IN}	Input Capacitance		15	pF	(4)
C_O	Output Capacitance		20	pF	(4)
$C_{I/O}$	I/O Capacitance		25	pF	(4)
C_{CLK}	CLK Input Capacitance		15	pF	(4)
C_{TIN}	Test Input Capacitance		15	pF	(4)
C_{TOUT}	Test Output Capacitance		20	pF	(4)
C_{TCK}	Test Clock Capacitance		15	pF	(4)
I_{LI}	Input Leakage Current		±15	µA	$0<V_{IN}<V_{IL}$, $V_{IH}< V_{IN}<V_{CC3}$, (1)
I_{LO}	Output Leakage Current		±15	µA	$0<V_{IN}<V_{IL}$, $V_{IH}< V_{IN}<V_{CC3}$, (1)
I_{IH}	Input High Leakage Current		200	µA	$V_{IN} = V_{CC3} - 0.4$ V, (3)
I_{IL}	Input Low Leakage Current		–400	µA	V_{IN} = 0.4 V (2, 5)

NOTES:
1. This parameter is for inputs/outputs without an internal pull up or pull down.
2. This parameter is for inputs with an internal pull up.
3. This parameter is for inputs with an internal pull down.
4. Guaranteed by design.
5. This specification applies to the HITM# pin when it is driven as an input (e.g., in JTAG mode).

4.3 AC Specifications

The AC specifications of the Extended Temperature Pentium processor with MMX technology consist of setup times, hold times, and valid delays at 0 pF.

4.3.1 Power and Ground

For clean on-chip power distribution, there are 42 V_{CC3}, 37 V_{CC2} and 72 V_{SS} inputs.

Power and ground connections must be made to all external V_{CC2}, V_{CC3} and V_{SS} pins. On the circuit board, all V_{CC2} pins must be connected to a proper voltage V_{CC2} plane or island (core voltage determined by package type/frequency). All V_{CC3} pins must be connected to a 2.5 V V_{CC3}, plane. All V_{SS} pins must be connected to a V_{SS} plane. Please refer to Table 2 on page 16 for the list of V_{CC2}, V_{CC3} and V_{SS} pins.

4.3.2 Decoupling Recommendations

Liberal decoupling capacitance should be placed near the processor. The processor's large address and data buses can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the processor and decoupling capacitors as much as possible. These capacitors should be evenly distributed around each component on the power plane. Capacitor values should be chosen to ensure they eliminate both low and high frequency noise components.

Power transients also occur as the processor rapidly transitions from a low level power consumption to a high level one (or high to low power transition). A typical example would be entering or exiting the Stop Grant state. Another example would be executing a HALT instruction, causing the processor to enter the Auto HALT Powerdown state, or transitioning from HALT to the Normal state. All of these examples may cause abrupt changes in the power being consumed by the processor.

Note that the Auto HALT Powerdown feature is always enabled even when other power management features are not implemented.

Bulk storage capacitors with a low ESR (Effective Series Resistance) in the 10 to 100 µf range are required to maintain a regulated supply voltage during the interval between the time the current load changes and the point that the regulated power supply output can react to the change in load. In order to reduce the ESR, it may be necessary to place several bulk storage capacitors in parallel.

These capacitors should be placed near the processor on both V_{CC2} plane and V_{CC3} plane to ensure that the supply voltages stay within specified limits during changes in the supply current during operation.

4.3.3 Connection Specifications

All NC/INC pins must remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Unused active low inputs should be connected to V_{CC3}. Unused active high inputs should be connected to ground.

4.3.4 AC Timings

The AC specifications given in Table 19 consist of output delays, input setup requirements and input hold requirements for the standard 66 MHz external bus. All AC specifications (with the exception of those for the TAP signals and APIC signals) are relative to the rising edge of the CLK input.

All timings are referenced to V_{CC3}/V_{CC2} for both “0” and “1” logic levels unless otherwise specified. Within the sampling window, asynchronous inputs must be stable for correct operation.

Each valid delay is specified for a 0 pF load. The system designer should use I/O buffer modeling to account for signal flight time delays. Do not select a bus fraction and clock speed which will cause the processor to exceed its internal maximum frequency.

The following specifications apply to all standard TTL signals used with the Pentium processor family:

- TTL input test waveforms are assumed to be 0 to 2.5 V transitions with 1.0 V/ns rise and fall times
- 0.3 V/ns ≤ input rise/fall time ≤ 5 V/ns
- All TTL timings are referenced from V_{CC3}/V_{CC2}

Table 19. Extended Temperature Pentium® Processor with MMX™ Technology AC Specifications† (Sheet 1 of 3)

Symbol	Parameter	Min	Max	Unit	Figure	Notes (see Table 21)
	CLK Frequency	33.33	66.6	MHz		(1)
t_{1a}	CLK Period	15.0	30.0	ns	9	
t_{1b}	CLK Period Stability		±250	ps		(2, 3)
t_2	CLK High Time	4.0		ns	9	@V_{CC3} – 0.7 V, (2)
t_3	CLK Low Time	4.0		ns	9	@0.5 V, (2)
t_4	CLK Fall Time	0.15	1.5	ns	9	V_{CC3} – 0.7 V to 0.5 V, (2, 4)
t_5	CLK Rise Time	0.15	1.5	ns	9	0.5 V to V_{CC3} –0.7 V, (2, 4)
t_{6a}	PWT, PCD, CACHE# Valid Delay	1.0	7.0	ns	10	
t_{6b}	AP Valid Delay	1.0	8.5	ns	10	
t_{6c}	LOCK#, Valid Delay	0.9	7.0	ns	10	
t_{6d}	ADS# Valid Delay	1.0	6.2	ns	10	
t_{6e}	A31–A3 Valid Delay	0.8	6.4	ns	10	
t_{6f}	M/IO# Valid Delay	0.8	6.2	ns	10	
t_{6g}	BE7#–BE0#, D/C#, W/R#, SCYC Valid Delay	0.8	7.0	ns	10	
t_7	ADS#, AP, A31–A3, PWT, PCD, BE7#–BE0#, M/IO#, D/C#, W/R#, CACHE#, SCYC, LOCK# Float Delay		10.0	ns	11	(2)
t_{8a}	APCHK#, IERR#, FERR# Valid Delay	1.0	8.3	ns	10	(5)
t_{8b}	PCHK# Valid Delay	1.0	7.0	ns	10	(5)
t_{9a}	BREQ Valid Delay	1.0	8.0	ns	10	(5)
t_{9b}	SMIACT# Valid Delay	1.0	7.3	ns	10	(5)
t_{9c}	HLDA Valid Delay	1.0	6.8	ns	10	(5)
t_{10a}	HIT# Valid Delay	1.0	6.8	ns	10	
t_{10b}	HITM# Valid Delay	0.9	6.0	ns	10	
t_{11a}	PM1–PM0, BP3–BP0 Valid Delay	1.0	10.0	ns	10	
t_{11b}	PRDY Valid Delay	1.0	8.0	ns	10	
t_{12}	D63–D0, DP7–DP0 Write Data Valid Delay	1.0	7.7	ns	10	
t_{13}	D63–D0, DP3–0 Write Data Float Delay		10.0	ns	11	(2)
t_{14}	A31–A5 Setup Time	6.0		ns	12	(6)
t_{15}	A31–A5 Hold Time	1.0		ns	12	
t_{16a}	INV, AP Setup Time	5.0		ns	12	
t_{16b}	EADS# Setup Time	5.0		ns	12	

† Refer to Table 14 for VCC and TCASE assumptions.

Table 19. Extended Temperature Pentium® Processor with MMX™ Technology AC Specifications† (Sheet 2 of 3)

Symbol	Parameter	Min	Max	Unit	Figure	Notes (see Table 21)
t_{17}	EADS#, INV, AP Hold Time	1.0		ns	12	
t_{18a}	KEN# Setup Time	5.0		ns	12	
t_{18b}	NA#, WB/WT# Setup Time	4.5		ns	12	
t_{19}	KEN#, WB/WT#, NA# Hold Time	1.0		ns	12	
t_{20}	BRDY# Setup Time	4.75		ns	12	
t_{21}	BRDY# Hold Time	1.0		ns	12	
t_{22}	AHOLD, BOFF# Setup Time	5.5		ns	12	
t_{23}	AHOLD, BOFF# Hold Time	1.0		ns	12	
t_{24a}	BUSCHK#, EWBE#, HOLD, Setup Time	5.0		ns	12	
t_{24b}	PEN# Setup Time	4.8		ns	12	
t_{25a}	BUSCHK#, EWBE#, PEN# Hold Time	1.0		ns	12	
t_{25b}	HOLD Hold Time	1.5		ns	12	
t_{26}	A20M#, INTR, STPCLK# Setup Time	5.0		ns	12	(7, 8)
t_{27}	A20M#, INTR, STPCLK# Hold Time	1.0		ns	12	(9)
t_{28}	INIT, FLUSH#, NMI, SMI#, IGNNE# Setup Time	5.0		ns	12	(8, 10, 17)
t_{29}	INIT, FLUSH#, NMI, SMI#, IGNNE# Hold Time	1.0		ns	12	(9)
t_{30}	INIT, FLUSH#, NMI, SMI#, IGNNE# Pulse Width, Async	2.0		CLKs	12	(10, 11)
t_{31}	R/S# Setup Time	5.0		ns	12	(7, 8, 10)
t_{32}	R/S# Hold Time	1.0		ns	12	(9)
t_{33}	R/S# Pulse Width, Async.	2.0		CLKs	12	(10, 11)
t_{34}	D63–D0, DP7–0 Read Data Setup Time	2.8		ns	12	
t_{35}	D63–D0, DP7–0 Read Data Hold Time	1.5		ns	12	
t_{36}	RESET Setup Time	5.0		ns	12	(7, 8)
t_{37}	RESET Hold Time	1.0		ns	13	(9)
t_{38}	RESET Pulse Width, V_{CC} & CLK Stable	15.0		CLKs	13	(10)
t_{39}	RESET Active After V_{CC} & CLK Stable	1.0		mS	13	Power up

† Refer to Table 14 for VCC and TCASE assumptions.

Table 19. Extended Temperature Pentium® Processor with MMX™ Technology AC Specifications† (Sheet 3 of 3)

Symbol	Parameter	Min	Max	Unit	Figure	Notes (see Table 21)
t_{40}	Reset Configuration Signals (INIT, FLUSH#) Setup Time	5.0		ns	13	(7, 8, 10)
t_{41}	Reset Configuration Signals (INIT, FLUSH#) Hold Time	1.0		ns	13	(9)
t_{42a}	Reset Configuration Signals (INIT, FLUSH#) Setup Time, Async.	2.0		CLKs	13	To RESET falling edge, (8)
t_{42b}	Reset Configuration Signals (INIT, FLUSH#) Setup Time, Async.	2.0		CLKs	13	To RESET falling edge
t_{43a}	BF2–BF0 Setup Time	1.0		mS	13	To RESET falling edge, (10)
t_{43b}	BF2–BF0 Hold Time	2.0		CLKs	13	To RESET falling edge, (12)
t_{43c}	APICEN, BE4# Setup Time	2.0		CLKs	13	To RESET falling edge
t_{43d}	APICEN, BE4# Hold Time	2.0		CLKs	13	To RESET falling edge
t_{44}	TCK Frequency	—	16.0	MHz		
t_{45}	TCK Period	62.5		ns	9	
t_{46}	TCK High Time	25.0		ns	9	@V_{CC3} –0.7 V, (2)
t_{47}	TCK Low Time	25.0		ns	9	@0.5 V, (2)
t_{48}	TCK Fall Time		5.0	ns	9	V_{CC3} –0.7 V to 0.5 V (2, 13, 14)
t_{49}	TCK Rise Time		5.0	ns	9	0.5 V to V_{CC3} –0.7 V, (2, 13, 14)
t_{50}	TRST# Pulse Width	40.0		ns	15	Asynchronous, (2)
t_{51}	TDI, TMS Setup Time	5.0		ns	15	(15)
t_{52}	TDI, TMS Hold Time	13.0		ns	15	(15)
t_{53}	TDO Valid Delay	3.0	20.0	ns	15	(13)
t_{54}	TDO Float Delay		25.0	ns	15	(2, 13)
t_{55}	All Non-Test Outputs Valid Delay	3.0	20.0	ns	15	(13, 16, 17)
t_{56}	All Non-Test Outputs Float Delay		25.0	ns	15	(2, 13, 16, 17)
t_{57}	All Non-Test Inputs Setup Time	5.0		ns	15	(15, 16, 17)
t_{58}	All Non-Test Inputs Hold Time	13.0		ns	15	(15, 16, 17)

† Refer to Table 14 for VCC and TCASE assumptions.

Table 20. APIC AC Specifications

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{60a}	PICCLK Frequency	2	16.66	MHz		
t_{60b}	PICCLK Period	60	500	ns	9	
t_{60c}	PICCLK High Time	15		ns	9	
t_{60d}	PICCLK Low Time	15		ns	9	
t_{60e}	PICCLK Rise Time	0.15	2.5	ns	9	
t_{60f}	PICCLK Fall Time	0.15	2.5	ns	9	
t_{60g}	PICD0–1 Setup Time	3		ns	12	To PICCLK
t_{60h}	PICD0–1 Hold Time	2.5		ns	12	To PICCLK
t_{60i}	PICD0–1 Valid Delay (L to H)	4	38	ns	10	From PICCLK, (18)
t_{60j}	PICD0–1 High Time (H to L)	4	22	ns	10	From PICCLK, (18)
t_{61}	PICCLK Setup Time	5.0			12	To CLK
t_{62}	PICCLK Hold Time	2.0			12	To CLK
t_{63}	PICCLK Ratio (CLK/PICCLK)	4				(19)

Table 21. Notes to Tables 19 and 20

1. CLK input frequency must be either 33.33 MHz (+1 MHz) or 66.6 MHz (–1 MHz). Operation in the range between 33.33 MHz and 66.6 MHz is not supported.
2. Not 100 percent tested. Guaranteed by design.
3. These signals are measured on the rising edge of adjacent CLKs at V_{CC3}/V_{CC2}. To ensure a 1:1 relationship between the amplitude of the input jitter and the internal and external clocks, the jitter frequency spectrum should not have any power spectrum peaking between 500 KHz and 1/3 of the CLK operating frequency. The amount of jitter present must be accounted for as a component of CLK skew between devices. The internal clock generator requires a constant frequency CLK input to within +250 ps, and therefore the CLK input cannot be changed dynamically.
4. 0.87 V/ns ≤ CLK input rise/fall time ≤ 8.7 V/ns.
5. APCHK#, FERR#, HLDA, IERR#, LOCK#, and PCHK# are glitch-free outputs. Glitch-free signals monotonically transition without false transitions.
6. Timing (t_{14}) is required for external snooping (e.g., address setup to the CLK in which EADS# is sampled active).
7. Setup time is required to guarantee recognition on a specific clock.
8. This input may be driven asynchronously.
9. Hold time is required to guarantee recognition on a specific clock.
10. When driven asynchronously, RESET, NMI, FLUSH#, R/S#, INIT, and SMI# must be de-asserted (inactive) for a minimum of two clocks before being returned active.
11. To guarantee proper asynchronous recognition, the signal must have been de-asserted (inactive) for a minimum of two clocks before being returned active and must meet the minimum pulse width.
12. BF2–BF0 should be strapped to V_{CC3} or V_{SS}.
13. Referenced to TCK falling edge.
14. 1 ns can be added to the maximum TCK rise and fall times for every 10 MHz of frequency below 33 MHz.
15. Referenced to TCK rising edge.
16. Non-test outputs and inputs are the normal output or input signals (besides TCK, TRST#, TDI, TDO, and TMS). These timings correspond to the response of these signals due to boundary scan operations.
17. During probe mode operation, do not use the boundary scan timings (t_{55}–t_{58}).
18. This assumes an external pull-up resistor to V_{CC} and a lumped capacitive load. The pull-up resistor must be between 300 Ω and 1 KΩ, the capacitance must be between 20 pF and 120 pF, and the RC product must be between 6 ns and 36 ns.
19. The CLK to PICCLK ratio has to be an integer and the ratio (CLK/PICCLK) cannot be smaller than 4.

Figure 9. Clock Waveform

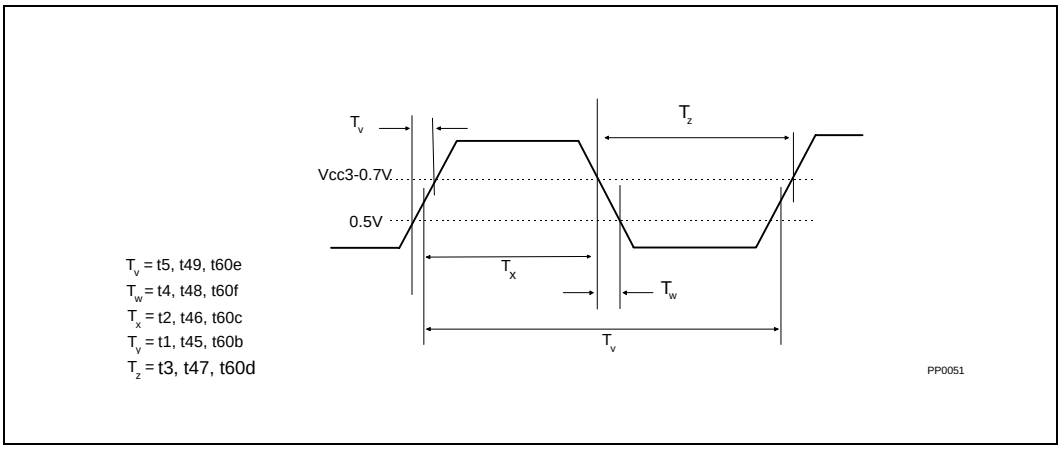


Figure 10. Valid Delay Timings

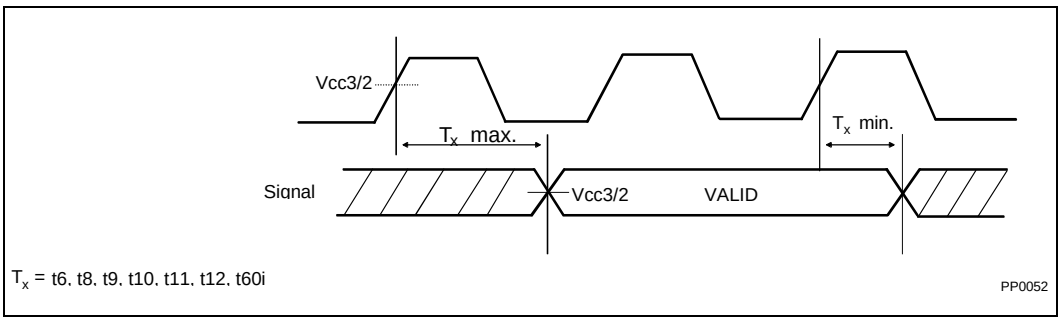


Figure 11. Float Delay Timings

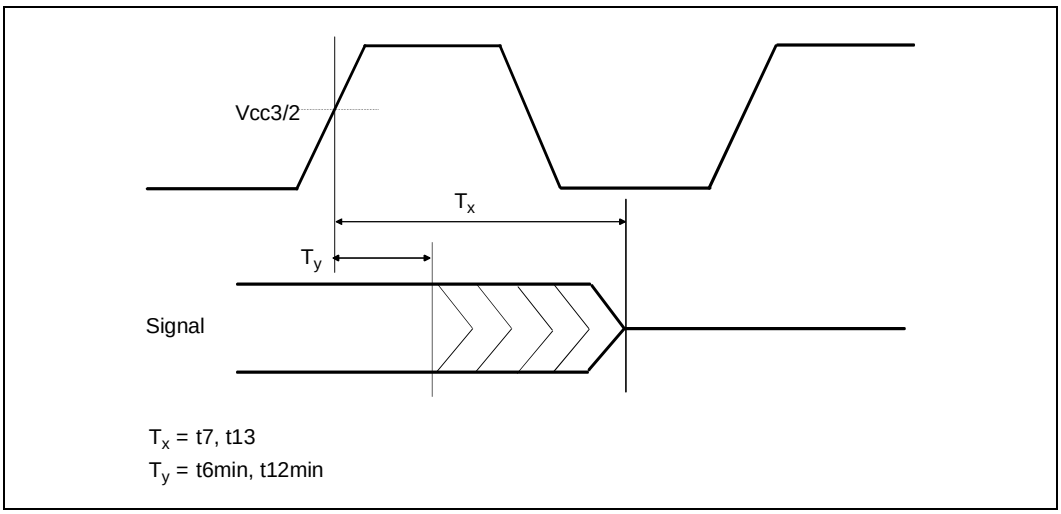
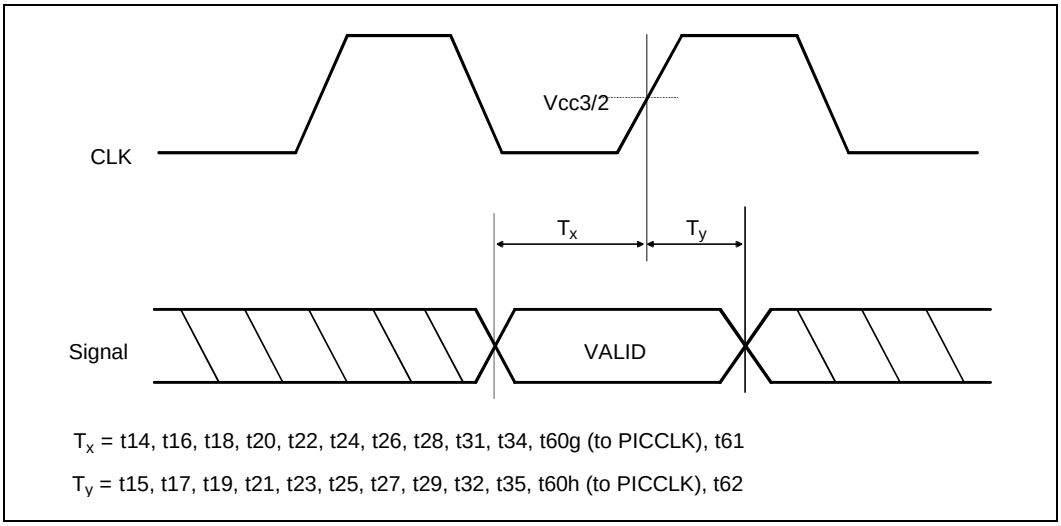


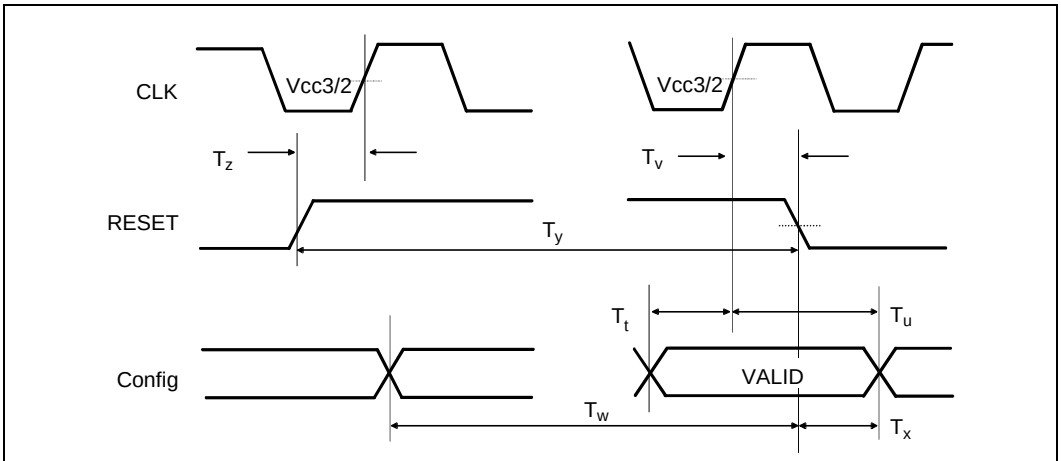
Figure 12. Setup and Hold Timings



T_x = t14, t16, t18, t20, t22, t24, t26, t28, t31, t34, t60g (to PICCLK), t61

T_y = t15, t17, t19, t21, t23, t25, t27, t29, t32, t35, t60h (to PICCLK), t62

Figure 13. Reset and Configuration Timings



T_t = t40

T_u = t41

T_v = t37

T_w = t42, t43c, t43e, t87

T_x = t43b, t43d, t43f, t88

T_y = t38, t39

T_z = t36

Figure 14. Test Timings

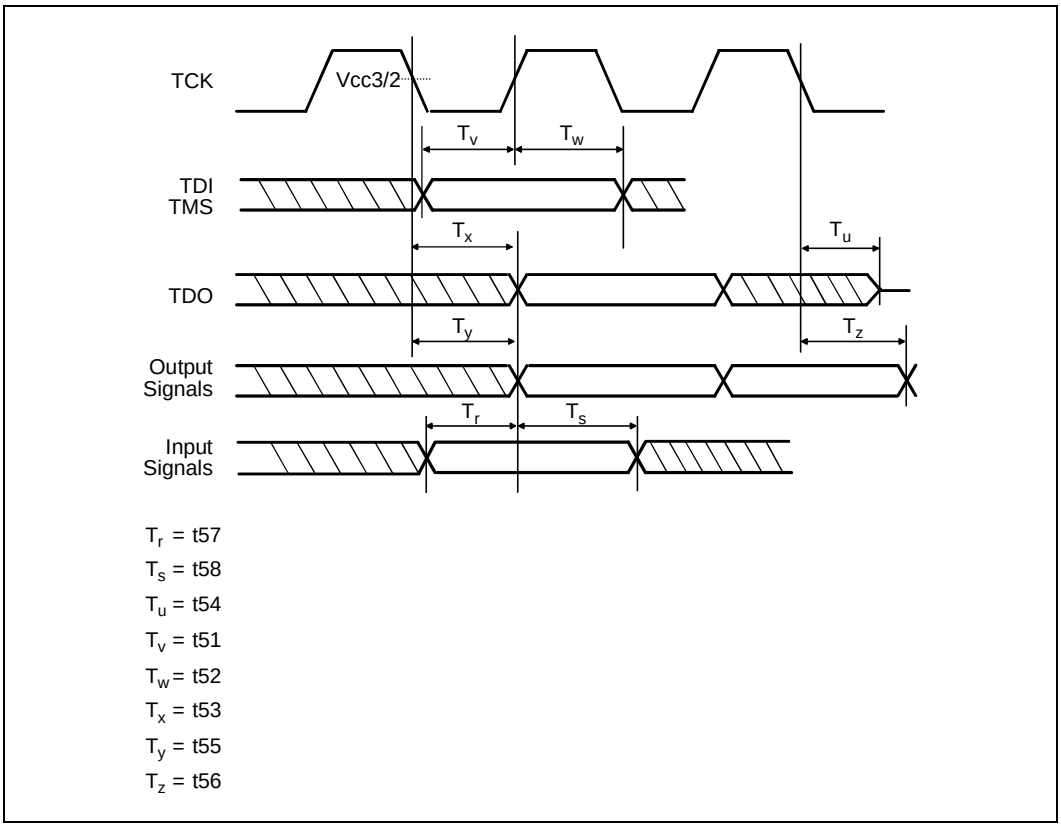
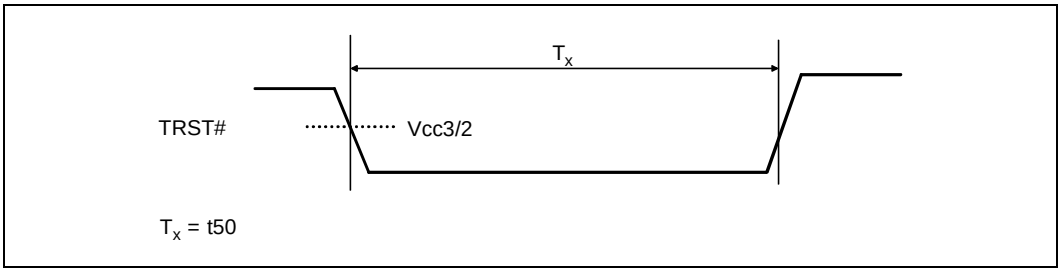


Figure 15. Test Reset Timings



4.4 I/O Buffer Models

This section describes the I/O buffer models of the Extended Temperature Pentium processor with MMX technology.

The first order I/O buffer model is a simplified representation of the complex input and output buffers used. Figure 16 shows the structure of the input buffer model and Figure 17 shows the output buffer model. Table 22 and 23 show the parameters used to specify these models.

Although simplified, these buffer models will accurately model flight time and signal quality. For these parameters, there is very little added accuracy in a complete transistor model.

In addition to the input and output buffer parameters, input protection diode models are provided for added accuracy. These diodes have been optimized to provide ESD protection and provide some level of clamping. Although the diodes are not required for simulation, it may be more difficult to meet specifications without them.

Note, however, some signal quality specifications require that the diodes be removed from the input model. The series resistors (Rs) are a part of the diode model. Remove these when removing the diodes from the input model.

Figure 16. First Order Input Buffer Model

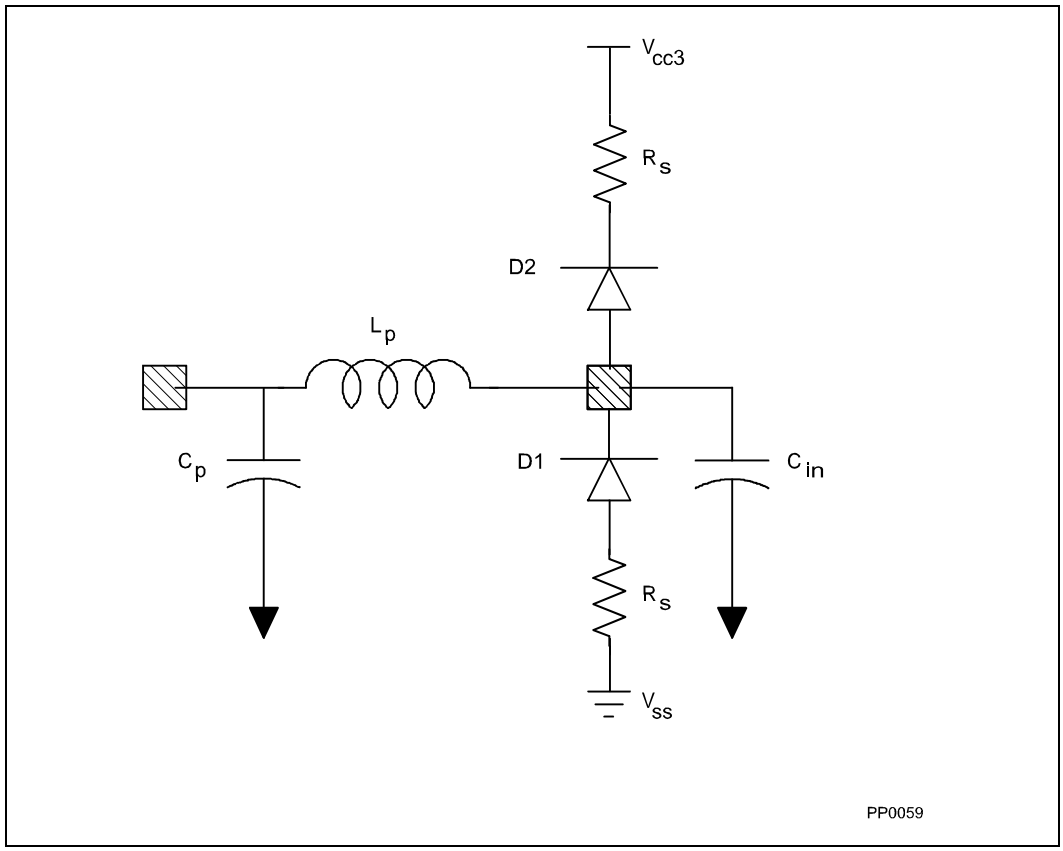


Table 22. Parameters Used in the Specification of the First Order Input Buffer Model

Parameter	Description
C_{in}	Minimum and Maximum value of the capacitance of the input buffer model
L_p	Minimum and Maximum value of the package inductance
C_p	Minimum and Maximum value of the package capacitance
R_S	Diode Series Resistance
D1, D2	Ideal Diodes

Figure 17. First Order Output Buffer Model

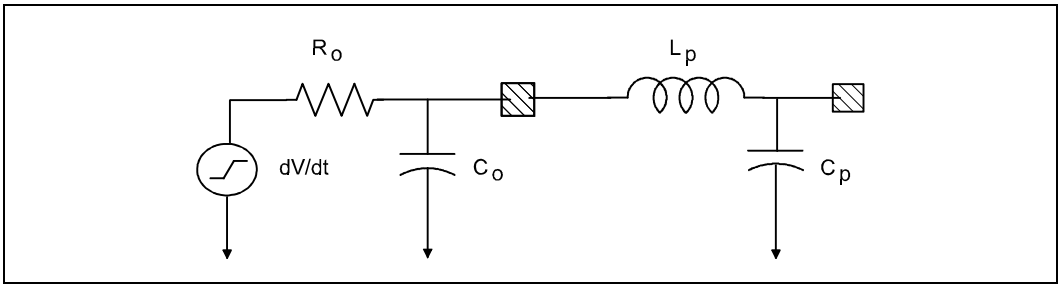


Table 23. Parameters Used in the Specification of the First Order Output Buffer Model

Parameter	Description
dV/dt	Minimum and maximum value of the rate of change of the open circuit voltage source used in the output buffer model
R_O	Minimum and maximum value of the output impedance of the output buffer model
C_O	Minimum and Maximum value of the capacitance of the output buffer model
L_P	Minimum and Maximum value of the package inductance
C_P	Minimum and Maximum value of the package capacitance

4.4.1 Buffer Model Parameters

This section gives the parameters for each input, output and bidirectional buffers.

Tables 24, 25, and 26 contain listings for all three buffer model types; do not get them confused during simulation. When a bidirectional pin is operating as an input, use the C_{IN}, C_P and L_P values; if it is operating as a driver, use all of the data parameters.

Please refer to Table 24 for the groupings of the buffers.

The input, output and bidirectional buffer's values are listed below. These tables contain listings for all three types. When a bidirectional pin is operating as an input, just use the C_{IN}, C_P and L_P values, if it is operating as a driver use all the data parameters.

Table 24. Signal to Buffer Type

Signals	Type	Driver Buffer Type	Receiver Buffer Type
A20M#, AHOLD, BF, BOFF#, BRDY#, BUSCHK#, EADS#, EWBE#, FLUSH#, HOLD, IGNNE#, INIT, INTR, INV, KEN#, NA#, NMI, PEN#, PICCLK, R/S#, RESET, SMI#, STPCLK#, TCK, TDI, TMS, TRST#, WB/WT#	I		ER1
APCHK#, BE7–BE5#, BP3–BP2, BREQ, FERR#, IERR#, PCD, PCHK#, PM0/BP0, PM1/BP1, PRDY, PWT, SMIACT#, TDO	O	ED1	
A31–A3, AP, BE4#–BE0#, CACHE#, D/C#, D63–D0, DP8–DP0, HLDA, LOCK#, M/IO#, SCYC, ADS#, HITM#, HIT#, W/R#, PICD0, PICD1	I/O	EB1	EB1

Table 25. Preliminary Input, Output and Bidirectional Buffer Model Parameters for HL-PBGA Package

Buffer Type	Transition	dV/dt (V/nsec)		R_O (Ohms)		C_P (pF)		L_P (nH)		C_O/C_{IN} (pF)	
		Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
ER1	Rising					0.2	0.4	6.4	11.3	0.8	1.2
(input)	Falling					0.2	0.4	6.4	11.3	0.8	1.2
ED1	Rising	2.2/2.2	2.7/0.15	21.6	65	0.2	0.5	5.4	11.7	2.0	2.6
(output)	Falling	2.2/2.9	2.7/0.22	17.5	75	0.2	0.5	5.4	11.7	2.0	2.6
EB1	Rising	2.2/2.2	2.7/0.15	21.6	65	0.2	0.4	5.2	10.3	2.0	2.6
(bidir)	Falling	2.2/2.9	2.7/0.22	17.5	75	0.2	0.4	5.2	10.3	2.0	2.6

NOTE: The data in this table is based on preliminary design information. Input, output and bidirectional buffer values are being characterized at this time.

Table 26. Input Buffer Model Parameters: D (Diodes)

Symbol	Parameter	D1	D2
I_S	Saturation Current	1.4e–14A	2.78e–16A
N	Emission Coefficient	1.19	1.00
R_S	Series Resistance	6.5 Ω	6.5 Ω
T_T	Transit Time	3 ns	6 ns
V_J	PN Potential	0.983 V	0.967 V
C_{J0}	Zero Bias PN Capacitance	0.281 pF	0.365 pF
M	PN Grading Coefficient	0.385	0.376

4.5 Signal Quality Specifications

Signals driven by the system into the Extended Temperature Pentium processor with MMX technology must meet signal quality specifications to guarantee that the components read data properly and to ensure that incoming signals do not affect the reliability of the component.

4.5.1 Overshoot

The maximum overshoot and overshoot threshold duration specifications for inputs to the Extended Temperature Pentium processor with MMX technology are described as follows:

- Maximum overshoot specification: The maximum overshoot of the CLK/PICCLK signals should not exceed V_{CC2}, nominal +0.6 V. The maximum overshoot of all other input signals should not exceed V_{CC3}, nominal +1.0 V.
- Overshoot threshold duration specification: The overshoot threshold duration is defined as the sum of all time during which the input signal is above V_{CC3}, nominal +0.3 V, within a single clock period. The overshoot threshold duration must not exceed 20% of the period.

Refer to Table 27 for a summary of the overshoot specifications for the Extended Temperature Pentium processor with MMX technology.

Table 27. Overshoot Specification Summary

Specification Name	Value	Units	Notes
Threshold Level	V_{CC3}, nominal +0.3	V	(1)
Maximum Overshoot Level (CLK and PICCLK)	V_{CC3}, nominal +0.6	V	(1)
Maximum Overshoot Level (all other inputs)	V_{CC3}, nominal +1.0	V	(1)
Maximum Threshold Duration	20% of clock period above threshold voltage	ns	
Maximum Ringback	V_{CC3}, nominal –0.7	V	(1)

NOTES:

1. V_{CC3}, nominal refers to the voltage measured at the V_{CC3} pins.

5.0 Extended Temperature Pentium® Processor with MMX™ Technology Errata Information

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

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

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

5.1 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.

5.2 Summary Table of Changes

The following table indicates the Specification Changes, S-Specs, Errata, Specification Clarifications or Documentation Changes, which apply to the 166 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:

5.2.1 Codes Used in Summary Table

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.
AP:	APIC related errata.

Table 28. Specification Changes

No.	Plans	Specification Changes
1	Doc	IDT limit violation causes GP fault, not interrupt 8

Table 29. Errata (Sheet 1 of 2)

No.	Plans	Errata
1	NoFix	NMI or INIT during HALT within SMM may cause large amount of bus activity
2	NoFix	Single-step debug exception breaks out of HALT
3	NoFix	Second assertion of FLUSH# not ignored
4	NoFix	Segment limit violation by FPU operand may corrupt FPU state
5	NoFix	FP exception inside SMM with pending NMI hangs system
6	NoFix	Data breakpoint deviations
7	NoFix	Event monitor counting discrepancies
8	NoFix	VERR type instructions causing page fault task switch with T bit set may corrupt CS:EIP
9	NoFix	BUSCHK# interrupt has wrong priority
10	NoFix	Matched but disabled data breakpoint can be lost by STPCLK# assertion
11	NoFix	STPCLK# ignored in SMM when INIT or NMI pending
12	NoFix	A fault causing a page fault can cause an instruction to execute twice
13	NoFix	Machine check exception pending, then HLT, can cause skipped or incorrect instruction, or CPU hang
14	NoFix	FBSTP stores BCD operand incorrectly If address wrap and FPU error both occur
15	NoFix	V86 interrupt routine at illegal privilege level can cause spurious pushes to stack
16	NoFix	Corrupted HLT flag can cause skipped or incorrect instruction, or CPU hang
17	NoFix	Benign exceptions can erroneously cause double fault
18	NoFix	Double fault counter may not increment correctly
19	NoFix	Short form of mov EAX / AX / AL may not pair
20	NoFix	Turning off paging may result in prefetch to random location
21	NoFix	STPCLK#, FLUSH# or SMI# after STI
22	NoFix	REP string instruction not interruptible by STPCLK#
23	NoFix	Single step may not be reported on first instruction after FLUSH#
24	NoFix	Double fault may generate illegal bus cycle
25	NoFix	TRST# not asynchronous
26	NoFix	STPCLK# on RSM to HLT causes non-standard behavior
27	NoFix	Code cache dump may cause wrong IERR#
28	NoFix	Asserting TRST# pin or issuing JTAG instructions does not exit TAP Hi-Z state
29	NoFix	ADS# may be delayed after HLDA deassertion
30	NoFix	Stack underflow in IRET gives #GP, not #SS
31	NoFix	Performance monitoring pins PM[1:0] may count the events incorrectly

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

Table 29. Errata (Sheet 2 of 2)

No.	Plans	Errata
32	NoFix	Branch trace messages may cause system hang
33	Fixed	Event monitor counting discrepancies (fix)
34	NoFix	Event monitor counting discrepancies (Nofix)
35	NoFix	TLB update is blocked after a specific sequence of events with a misaligned descriptor
36	NoFix	Erroneous debug exception on POPF/IRET instructions with a GP fault
37	NoFix	CR2 and CR4 Content upon Return from SMM
38	Fix	Invalid operand with locked CMPXCHG8B instruction
39	Fix	Event monitor counting discrepancy
40	NoFix	FBSTP instruction incorrectly sets Accessed and Dirty bits of Page Table entry
1AP.	NoFix	INIT and SMI via the APIC three-wire bus may be lost
2AP.	NoFix	PICCLK must toggle for at least twenty cycles before RESET

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

Table 30. Specification Clarifications

No.	Plans	Specification Clarifications
1	Doc	Only one SMI# can be latched during SMM
2	Doc	APIC 8-bit access
3	Doc	LOCK prefix excludes APIC memory space
4	Doc	SMI# activation may cause a nested NMI handling
5	Doc	Code breakpoints set on meaningless prefixes not guaranteed to be recognized
6	Doc	Resume flag should be set by software
7	Doc	Data breakpoints on INS delayed one iteration
8	Doc	When L1 cache disabled, inquire cycles are blocked
9	Doc	Serializing operation required when one CPU modifies another CPU's code
10	Doc	For correct translations, the TLB should be flushed after the PSE bit in CR4 is set
11	Doc	Extra code break can occur on I/O or HLT instruction if SMI coincides
12	Doc	FYL2XP1 does not generate exceptions for X out of range
13	Doc	Enabling NMI inside SMM
14	Doc	BF[1:0] must not change values while RESET is active
15	Doc	Active A20M# during SMM
16	Doc	POP[ESP] with 16-bit stack size
17	Doc	Line fill order optimization revision
18	Doc	Test Parity Check Mechanism Clarification

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

5.2.2 Documentation Changes

Table 31. Documentation Changes

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

6.0 Extended Temperature Pentium® Processor with MMX™ Technology Specification Changes

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

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

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

7.0 Extended Temperature Pentium® Processor with MMX™ Technology Errata

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

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

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

Workaround: Use one of the following:

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

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

2. 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.

3. 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.

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

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

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

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

Note that the *Intel Architecture Software Developer's Manual*, Volume 3, Chapter 17 warns in general that the Pentium processor may store only part of operands which generate a memory fault by crossing either a segment or page limit. This erratum is just one case of that general problem, and all cases will be avoided by following the recommended programming practice of never

straddling segment or page boundaries with operands. Note also that the handling of operands which straddle such boundaries is processor specific, so code which uses such straddling will behave differently when run on different Intel Architecture processors.

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

Workaround:

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

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

5. 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.

6. Data Breakpoint Deviations

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

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

handler, but instead it goes to the data breakpoint handler. This is incorrect because the data access that should trigger the breakpoint has not occurred yet.

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

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

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

Workaround: Use one of the following:

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

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

7. 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.

8. 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.

9. BUSCHK# Interrupt Has Wrong Priority

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

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

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

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

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

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

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

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

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

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

Another problem can occur due to this erratum if the system is using the SMM I/O instruction restart feature. This problem requires an improbable coincidence: the SMI# signal caused by an I/O restart event must occur essentially simultaneously with BUSCHK#, such that the SMI# interrupt hits the narrow window (as described above) just before the first instruction of the BUSCHK# handler begins execution. This could happen if the same I/O instruction that triggers SMI# (usually to turn back on a device that's been turned off to save power) also generates a bus failure due to the system suddenly going defective, thus signaling BUSCHK#. The result is that the SMI# interrupt is serviced after the EIP has already been switched to point to the first instruction of the BUSCHK# handler, instead of the I/O instruction. The SMM code that services the I/O restart feature may well use the image of EIP in the SMRAM state save memory to inspect the I/O instruction, for example to determine what I/O address it's trying to access. In this case, the I/O restart part of SMM code will not find the correct instruction. If it is well written, it will execute RSM when it determines there is no valid I/O access to service. Then execution returns to the BUSCHK# handler with no deleterious impact. But less robust code might turn on the wrong I/O device, hang up, or begin executing from a random location.

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

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

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

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

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

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

Workaround: Use one of the following:

1. Use only *enabled* data breakpoints when STPCLK# may be asserted.

2. Disable the assertion of STPCLK# while this debug feature is being used.

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

11. 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

Workaround: The problems caused by this erratum will be prevented if the Machine Check Exception handler (if invoked) always forces a CPU RESET or INIT (which it should do anyway, for reasons given

above). Since the problems can occur *inside* the MCE handler, the IF should be left zero to prevent INTR from interrupting. Also, NMI, SMI and FLUSH could be blocked inside the MCE handler. The most secure strategy is to force INIT immediately upon entrance to the MCE handler.

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

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

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

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

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

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

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

Workaround: Use one of the following:

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

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

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

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

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

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

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

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

16. 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.

17. Benign Exceptions Can Erroneously Cause Double Fault

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

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

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

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

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

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

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

18. 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.

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

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

The Intel Architecture includes special instructions to MOV EAX /AX /AL to a memory offset (opcodes 0A2H & 0A3H). These instructions don't have a MOD/RM byte (and so are shortened by one byte). Instead, the opcode is followed immediately by 1/2/4 bytes giving the memory offset (displacement). This erratum occurs specifically when a MOV mem, EAX /AX /AL instruction using opcode 0A2H or 0A3H is followed by an instruction that uses the EAX /AX /AL register as a source (register source, or as base or index for the address of a memory source) or a destination register. Then the instruction unit detects a (false) dependency and it doesn't allow pairing. For example, the following two instructions are not paired:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Workaround: None identified at this time.

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

22. REP String Instruction Not Interruptible by STPCLK#

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

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

Workaround: None identified at this time.

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

23. 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.

24. 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.

25. TRST# Not Asynchronous

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

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

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

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

26. 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.

27. 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.

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

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

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

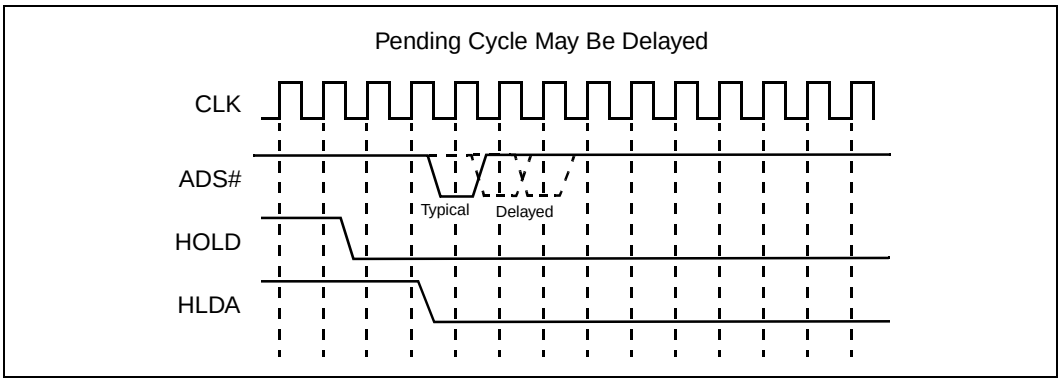
Workaround: To exit TAP Hi-Z state, in addition to the methods described above, the processor needs to be RESET as well.

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

29. ADS# May be Delayed After HLDA Deassertion

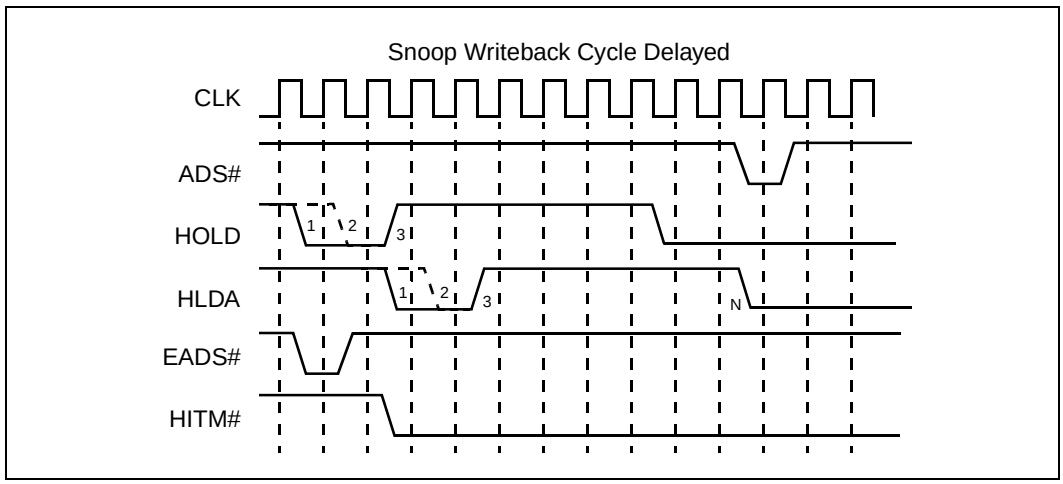
Problem: The Pentium processor typically starts a pending bus cycle on the same clock that HLDA is deasserted and the Pentium processor with MMX technology typically starts the cycle one clock after HLDA is deasserted. However, in both processors it may be delayed by as many as two clocks. See the diagram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

Figure 18. Pending Bus Cycle Timing Diagram



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.

Figure 19. Snoop Writeback Cycle Timing Diagram



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.

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

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

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

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

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

31. 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.

32. Branch Trace Messages May Cause System Hang

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

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

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

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

33. Event Monitor Counting Discrepancies (Fix)

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

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

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

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

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

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

34. Event Monitor Counting Discrepancies (NoFix)

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

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

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

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

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

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

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

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

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

When the TLB miss discussed in condition 5 above occurs, the processor starts a split locked read-modify-write sequence to update the descriptor access or busy bit. During this split locked cycle, the address of the low bytes of the descriptor is loaded into a slot in the TLB. The address of the high bytes of the descriptor is then put into the same slot of the TLB causing the address of the low bytes to be overwritten (this is caused by conditions 1-3 above). The address of the low bytes of the descriptor then needs to be re-read from memory. However, since the bus is now locked, this cannot occur and the processor hangs waiting for the sequence to complete.

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

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

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

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

Problem: An erroneous debug exception can occur due to execution of a POPF or IRET instruction in virtual 8086 mode, if there is a data breakpoint set on the address pointed to by SS:ESP, and the POPF or IRET triggers a general protection fault. This occurs in virtual 8086 mode when the IOPL < 3,

causing POPF and IRET to trap to the GP fault without accessing the stack. The data breakpoint set on the stack should not be triggered, but in fact it is incorrectly triggered as soon as the GP fault handler is entered.

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

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

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

37. CR2 and CR4 Content upon Return from SMM

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

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

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

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

38. Invalid Operand with Locked CMPXCHG8B Instruction

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

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

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

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

1. The first part of this workaround sets the first 7 entries (0-6) of the Interrupt Descriptor Table (IDT) in a non-writeable page. When the invalid opcode exception (exception 6) occurs due to

the locked CMPXCHG8B instruction with an invalid register destination (and only then), the processor will generate a page fault if it does not have write access to the page containing entry 6 of the IDT. The second part of this workaround modifies the page fault handler to recognize and correctly dispatch the invalid opcode exceptions that are now routed through the page fault handler.

Part I, IDT Page Access:

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

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

Part II, Page Fault Handler Modifications:

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

OR

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

   Part I, IDT Alignment:

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

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

   Part II, Page Fault Handler Modifications:

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

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

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

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

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

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

39. Event Monitor Counting Discrepancy

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

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

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

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

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

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

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

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

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

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

Workaround: None.

1AP. 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.

2AP. PICCLK Must Toggle For at Least Twenty Cycles Before RESET

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

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

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

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

8.0 Extended Temperature Pentium® Processor with MMX™ Technology Specification Clarifications

1. Only One SMI# Can Be Latched During SMM

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

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

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

2. 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.

3. 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.

4. SMI# Activation May Cause a Nested NMI Handling

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

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

5. 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.

6. 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.

7. 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."

8. 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."

9. 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.

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

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

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

11. 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.

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

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

13. Enabling NMI Inside SMM

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

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

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

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

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

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

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

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

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

15. Active A20M# During SMM

Section 14.3.3. of the 1997 *Pentium® Processor Family Developer's Manual* describes two considerations when using the A20M# input and SMM when SMRAM is relocated above 1 Megabyte. The system designer must ensure that A20M# is de-asserted on entry into SMM.

A20M# must be driven inactive before the first cycle of the SMM state save, and must be returned to its original level after the last cycle of the SMM state restore. This can be done by blocking the assertion of A20M# whenever SMIACT# is active.

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

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

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

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

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

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

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

It should read as follows:

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

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

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

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

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

The result of the memory write is processor family specific. For example, in Pentium II and Pentium Pro processors, the result of the memory write is to SS:0h plus any scaled index and displacement. In Pentium and Intel486 processors, the result of the memory write may be either a

stack fault (real mode or protected mode with a stack segment size of 64 Kbytes), or write to SS:10000h plus any scaled index and displacement (protected mode and stack segment size exceeds 64 Kbytes).

17. Line Fill Order Optimization Revision

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

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

Instead, it should read as follows:

"For Pentium processors with MMX technology, Pentium Pro and Pentium II processors, data is available for use in the order that it arrives from memory. On these processors, if an array of data is being read serially, it is preferable to access it in sequential order so that each data item will be used as it arrives from memory. On Pentium processors with MMX technology, the first 8-byte section is available immediately after it arrives from memory (after 5 I/O cycles), but the rest of the cache line is not available until the entire line is read from memory. If an array of data is being read serially, it is often preferable to pre-fetch portions of the array mapping to other cache lines. The first 8-byte section of the other cache line will be available for use immediately after it arrives from memory. This technique can be used to effectively hide the latency associated with accessing the last three 8-byte sections of the cache line.access it in sequential order so that each data item will be used as it arrives from memory. If a request of reading the next cache line is made after the first quad-word of the first cache line is available, memory will provide the first 8-byte section of the second cache line, while the processor loads rest of the three quad-words of the first cache line from the in-line buffer. It take three I/O cycles to load the first section of the second cache line instead of five I/O cycles. Notice the order for each data item for these two cache line to arrive from memory will still be in a sequential order, that is, the data items for the second cache line will not be available until the first cache line is completely loaded. This technique is frequently used in modern compiler technology.

18. Test Parity Check Mechanism Clarification

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

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

9.0 Extended Temperature Pentium® Processor with MMX™ Technology Documentation Changes

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

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

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

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

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

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

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

3. FSETPM Is Like NOP, Not Like FNOP

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

4. Errors in Three Tables of Special Descriptor Types

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

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

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

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

6. Incorrect Sequence of Registers Stored in PUSHA/PUSHAD

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

7. One-Byte Opcode Map Correction

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

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

10.0 Pentium® Processor Related Technical Collateral

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

Table 32. Pentium® Processor Related Technical Collateral

Document Title	Order Number
Embedded Pentium® Processor with MMX™ Technology Datasheet	273214
Embedded Pentium® Processor with MMX™ Technology Flexible Motherboard Design Guidelines	273206
Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture	243190
Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference	243191
Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide	243192
Pentium® Processor Specification Update	242480
Pentium® Processor Family Product Brief	241561
Pentium® Processor Performance Brief	241557
Pentium® Processor Technical Overview	241610
AP-579: *Pentium® Processor Flexible Motherboard Design Guidelines*	243187
AP-479: *Pentium® Processor Clock Design*	241574
AP-480: *Pentium® Processor Thermal Design Guidelines*	241575
AP-485: *Intel Processor Identification with the CPUID Instruction*	241618
AP-500: *Optimizations for Intel's 32-Bit Processors*	241799
AP-578: *Software and Hardware Considerations in Handling FPU Exceptions*	242415