Embedded Pentium® Processor Family

Developer's Manual

December 1998

Order Number: 273204-001

Contents

Figures

Tables

Guide to this Manual 1

1.1 Manual Contents

This manual contains 14 chapters and an index. This section summarizes the contents of the remaining chapters. The remainder of this chapter describes notation conventions and special terminology used throughout the manual and provides references to related documentation.

Chapter 2, “Architectural Features”	This chapter provides an overview of the embedded Pentium® processor, including product features, system components, system architecture, and applications.
Chapter 3, “Component Operation”	This chapter describes the Pentium processor internal architecture, with an overview of the processor’s functional units.
Chapter 4, “Microprocessor Initialization and Configuration”	This chapter details the Pentium processor register set, including the base architecture registers, system-level registers, and debug and test registers.
Chapter 5, “Hardware Interface”	This chapter describes the signals for the Pentium processor family.
Chapter 6, “Bus Functional Description”	This chapter describes the features of the processor bus, including bus cycle handling, interrupt and reset signals, cache control, and floating-point error control.
Chapter 7, “Electrical Differences Between Family Members”	This section describes the electrical differences between the embedded Pentium processor (at 100/133/166 MHz) and the embedded Pentium processor with MMX™ technology .
Chapter 8, “I/O Buffer Models”	This chapter describes the 3.3 V I/O buffer models of the embedded Pentium processor.
Chapter 9, “Testability”	This chapter describes the features which are included in the embedded Pentium processor for the purpose of enhancing testability. This chapter describes component testing using the Built-In Self-Test (BIST) feature, three-state test mode, and the IEEE 1149.1 “Test Access Port and Boundary Scan” mechanism.
Chapter 10, “Error Detection”	This chapter describes data integrity features that are focused on the detection and limited recovery of errors. The data integrity features provide capabilities for error detection of the internal devices and the external interface.
Chapter 11, “Execution Tracing”	This chapter describes the special bus cycles used to support execution tracing. Execution tracing allows the external hardware to track the flow of instructions as they execute inside the processor.

Chapter 12, “Power Management”	The embedded Pentium processor family implements Intel’s System Management Mode (SMM) architecture. This chapter describes the hardware interface to SMM and Clock Control.
Chapter 13, “Debugging”	This chapter describes the Pentium processor debugging support, including the breakpoint instruction, single-step trap, and debug registers.
Chapter 14, “Model Specific Registers and Functions”	This chapter introduces the model specific registers (MSRs) as they are implemented on the embedded Pentium processor family. Model specific registers are used to provide access to features that are generally tied to implementation-dependent aspects of a particular processor.

1.2 Notation Conventions

The following notations are used throughout this manual.

The pound symbol (#) appended to a signal name indicates that the signal is active low.

Variables Variables are shown in italics. Variables must be replaced with correct values.

Instructions Instruction mnemonics are shown in upper case. When you are programming, instructions are not case-sensitive. You may use either upper or lower case.

Numbers Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character H. A zero prefix is added to numbers that begin with A through F. (For example, FF is shown as 0FFH.) Decimal and binary numbers are represented by their customary notations. (That is, 255 is a decimal number and 1111 1111 is a binary number. In some cases, the letter B is added for clarity.)

Units of Measure The following abbreviations are used to represent units of measure:

A	amps, amperes
mA	milliamps, milliamperes
µA	microamps, microamperes
Mbyte	megabytes
Kbyte	kilobytes
Gbyte	gigabyte
W	watts
KW	kilowatts
mW	milliwatts
µW	microwatts
MHz	megahertz
ms	milliseconds

ns	nanoseconds
µs	microseconds
µF	microfarads
pF	picofarads
V	volts

Bit and Signal Ranges When the text refers to a range of register bits or signals, the range is represented by the highest and lowest number, separated by a dash (example: A15–A8). For register bits, the first bit shown is the most-significant and the second bit shown is the least-significant.

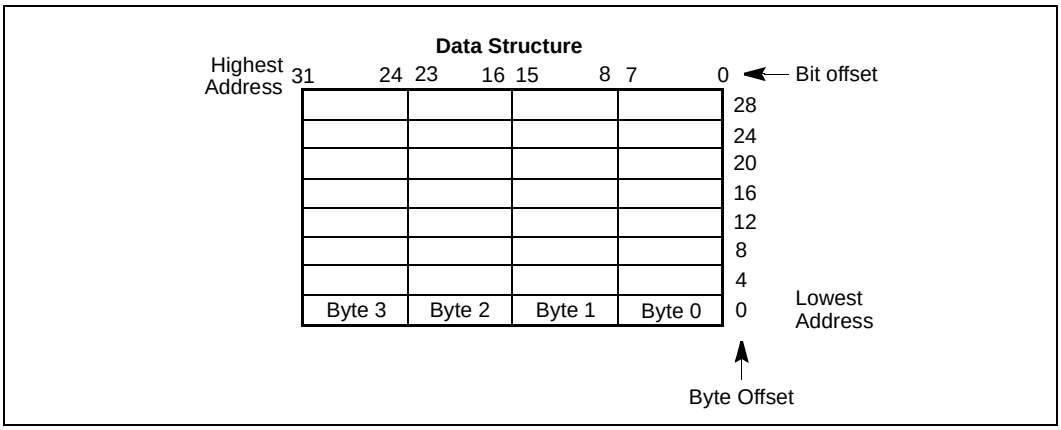
Register Names Register names are shown in upper case. When a register name contains a lower case, italic character, the name represents more than one register. For example, CR*n* represents these registers: CR0, CR1, CR2, etc.

Signal Names Signal names are shown in upper case. A pound symbol (#) appended to a signal name identifies an active-low signal.

1.2.1 Bit and Byte Order

In illustrations of data structures in memory, smaller addresses appear toward the bottom of the figure; addresses increase toward the top. Bit positions are numbered from right to left. The numerical value of a set bit is equal to two raised to the power of the bit position. Intel Architecture processors is a “little endian” machines; this means the bytes of a word are numbered starting from the least significant byte. Figure 1-1 illustrates these conventions.

Figure 1-1. Bit and Byte Order



1.2.2 Reserved Bits and Software Compatibility

In many register and memory layout descriptions, certain bits are marked as **reserved**. When bits are marked as reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as not only undefined, but unpredictable. Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing the values of registers which contain such bits. Mask out the reserved bits before testing.

- Do not depend on the states of any reserved bits when storing to memory or to a register.
- Do not depend on the ability to retain information written into any reserved bits.
- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

Note: Avoid any software dependence upon the state of reserved bits in Intel Architecture registers. Depending upon the values of reserved register bits will make software dependent upon the unspecified manner in which the processor handles these bits. Depending upon reserved values risks incompatibility with future processors.

1.2.3 Instruction Operands

When instructions are represented symbolically, a subset of the Intel Architecture assembly language is used. In this subset, an instruction has the following format:

```
label: mnemonic argument1, argument2, argument3
```

where:

- A **label** is an identifier which is followed by a colon.
- A **mnemonic** is a reserved name for a class of instruction opcodes which have the same function.
- The operands *argument1*, *argument2*, and *argument3* are optional. There may be from zero to three operands, depending on the opcode. When present, they take the form of either literals or identifiers for data items. Operand identifiers are either reserved names of registers or are assumed to be assigned to data items declared in another part of the program (which may not be shown in the example).

When two operands are present in an arithmetic or logical instruction, the right operand is the source and the left operand is the destination.

For example:

```
LOADREG: MOV EAX, SUBTOTAL
```

In this example LOADREG is a label, MOV is the mnemonic identifier of an opcode, EAX is the destination operand, and SUBTOTAL is the source operand. Some assembly languages put the source and destination in reverse order.

1.2.4 Hexadecimal and Binary Numbers

Base 16 (hexadecimal) numbers are represented by a string of hexadecimal digits followed by the character H (for example, F82EH). A hexadecimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Base 2 (binary) numbers are represented by a string of 1s and 0s, sometimes followed by the character B (for example, 1010B). The “B” designation is only used in situations where confusion as to the type of number might arise.

1.2.5 Segmented Addressing

The processor uses byte addressing. This means memory is organized and accessed as a sequence of bytes. Whether one or more bytes are being accessed, a byte address is used to locate the byte or bytes memory. The range of memory that can be addressed is called an **address space**.

The processor also supports segmented addressing. This is a form of addressing where a program may have many independent address spaces, called **segments**. For example, a program can keep its code (instructions) and stack in separate segments. Code addresses would always refer to the code space, and stack addresses would always refer to the stack space. The following notation is used to specify a byte address within a segment:

Segment-register:Byte-address

For example, the following segment address identifies the byte at address FF79H in the segment pointed by the DS register:

```
DS:FF79H
```

The following segment address identifies an instruction address in the code segment. The CS register points to the code segment and the EIP register contains the address of the instruction.

```
CS:EIP
```

1.2.6 Exceptions

An exception is an event that typically occurs when an instruction causes an error. For example, an attempt to divide by zero generates an exception. However, some exceptions, such as breakpoints, occur under other conditions. Some types of exceptions may provide error codes. An error code reports additional information about the error. An example of the notation used to show an exception and error code is shown below.

```
#PF(fault code)
```

This example refers to a page-fault exception under conditions where an error code naming a type of fault is reported. Under some conditions, exceptions which produce error codes may not be able to report an accurate code. In this case, the error code is zero, as shown below for a general-protection exception.

```
#GP(0)
```

See Chapter 5, *Interrupt and Exception Handling*, in the *Intel Architecture Software Developer's Manual, Volume 3,* for a list of exception mnemonics and their descriptions.

1.3 Special Terminology

The general terms "processor," "embedded Pentium processor," and "embedded Pentium processor family" are used throughout this manual to refer to the embedded Pentium processor, the embedded Pentium processor with Voltage Reduction Technology, the embedded Pentium processor with MMX technology, and the low-power embedded Pentium processor with MMX technology together. Some of the features or functions described using these terms, however, may not be available on each processor type. Refer to the datasheet for each product to determine whether a specific feature is offered.

In some instances, the names “embedded Pentium processor,” “embedded Pentium processor with Voltage Reduction Technology,” “embedded Pentium processor with MMX technology,” and “low-power embedded Pentium processor with MMX technology” are used in this manual to distinguish between processors when specific differences exist.

See “Related Documents” on page 1-8 for a list of datasheets and other documents that describe the operation of Pentium processors.

The following terms have special meanings in this manual.

Assert and Deassert The terms assert and deassert refer to the acts of making a signal active and inactive, respectively. The active polarity (high/low) is defined by the signal name. Active-low signals are designated by the pound symbol (#) suffix; active-high signals have no suffix. To assert FLUSH# is to drive it low; to assert HOLD is to drive it high; to deassert FLUSH# is to drive it high; to deassert HOLD is to drive it low.

DOS I/O Address Peripherals that are compatible with PC/AT system architecture can be mapped into DOS (or PC/AT) addresses 0H–03FFH. In this manual, the terms *DOS address* and *PC/AT address* are synonymous.

Expanded I/O Address All peripheral registers reside at I/O addresses 0F000H–0FFFFH. PC/AT-compatible integrated peripherals can also be mapped into DOS (or PC/AT) address space (0H–03FFH).

PC/AT Address Integrated peripherals that are compatible with PC/AT system architecture can be mapped into PC/AT (or DOS) addresses 0H–03FFH. In this manual, the terms *DOS address* and *PC/AT address* are synonymous.

Set and Clear The terms set and clear refer to the value of a bit or the act of giving it a value. If a bit is set, its value is “1”; setting a bit gives it a “1” value. If a bit is clear, its value is “0”; clearing a bit gives it a “0” value.

1.4 Technical Support

1.4.1 Electronic Support Systems

Intel’s site on the World Wide Web (http://www.intel.com/) provides up-to-date technical information and product support. This information is available 24 hours a day, 7 days a week, providing technical information whenever you need it.

1.4.1.1 Online Documents

Product documentation is provided online in a variety of web-friendly formats at:

http://developer.intel.com/design/litcentr/index.htm

1.4.1.2 Intel Product Forums

Intel provides technical expertise through electronic messaging. With publicly accessible forums, you have all of the benefits of email technical support, with the added benefit of the option of viewing previous messages written by other participants, and providing suggestions and tips that can help others.

Each of Intel's technical support forums is based on a single product or product family. Questions and replies are limited to the topic of the particular forum. Intel also provides several non-technical support related forums.

Complete information on Intel forums is available at:

http://support.intel.com/newsgroups/index.htm

1.4.2 Telephone Technical Support

In the U.S. and Canada, technical support representatives are available to answer your questions between 5 a.m. and 5 p.m. PST. You can also fax your questions to us. (Please include your voice telephone number and indicate whether you prefer a response by phone or by fax). Outside the U.S. and Canada, please contact your local distributor.

1-800-628-8686	U.S. and Canada
916-356-7599	U.S. and Canada
916-356-6100 (fax)	U.S. and Canada

1.5 Product Literature

You can order product literature from the following Intel literature centers.

1-800-548-4725	U.S. and Canada
708-296-9333	U.S. (from overseas)
44(0)1793-431155	Europe (U.K.)
44(0)1793-421333	Germany
44(0)1793-421777	France
81(0)120-47-88-32	Japan (fax only)

1.6 Related Documents

Table 1-1. Related Resources

Document Title	Order Number
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
Embedded Pentium® Processor datasheet	273202
Embedded Pentium® Processor with Voltage Reduction Technology datasheet	273203
Embedded Pentium® Processor with MMX™ Technology datasheet	273214
Low-Power Embedded Pentium® Processor with MMX™ Technology datasheet	273184
Pentium® Processor Specification Update	242480
Pentium® Processor for Embedded Applications Specification Update	273183
Optimizing for Intel's 32-Bit Processors	241799
MultiProcessor Specification	242016
Embedded Pentium® Processor Flexible Motherboard Design Guidelines	273206
Implementation Guidelines for 3.3 V Pentium® Processors with VR/VRE Specifications	242687
Voltage Guidelines for Pentium® Processors with MMX™ Technology	243186

Architectural Features 2

This volume describes the basic features and operation of embedded Pentium® processors:

- Embedded Pentium processors with maximum operating frequencies of 100, 133, and 166 MHz
- Embedded Pentium processors with Voltage Reduction Technology with a maximum operating frequency of 133 MHz
- Embedded Pentium processors with MMX™ technology with maximum operating frequencies of 200 and 233 MHz
- Low-power embedded Pentium processors with MMX™ technology with maximum operating frequencies of 166 and 266 MHz

The general terms “processor,” “embedded Pentium processor,” and “embedded Pentium processor family” are used throughout this manual to refer to the embedded Pentium processor, the embedded Pentium processor with Voltage Reduction Technology, the embedded Pentium processor with MMX technology, and the low-power embedded Pentium processor with MMX technology together. Some of the features or functions described using these terms, however, may not be available on each processor type. Refer to the datasheet for each product to determine whether a specific feature is offered.

In some instances, the names “embedded Pentium processor (at 100/133/166 MHz),” “embedded Pentium processor with Voltage Reduction Technology,” “embedded Pentium processor with MMX technology,” and “low-power embedded Pentium processor with MMX technology” are used in this manual to distinguish between processors when specific differences exist.

See “Related Documents” on page 1-8 for a list of datasheets and other documents that describe the operation of Pentium processors.

2.1 Processor Features Overview

The embedded Pentium processor supports the features of previous Intel® architecture processors and provides significant enhancements, including the following (refer to the datasheet for a specific list of features supported by each processor):

- Superscalar architecture
- Dynamic branch prediction
- Pipelined Floating-Point Unit
- Improved instruction execution time
- Separate code and data caches
- Writeback MESI protocol in the data cache
- 64-bit data bus
- Bus cycle pipelining
- Address parity

- Internal parity checking
- Functional redundancy checking and lock-step operation
- Execution tracing
- Performance monitoring
- IEEE 1149.1 boundary scan
- System Management Mode
- Virtual Mode extensions
- Dual processing support
- Advanced SL power management features
- Fractional bus operation
- On-chip local APIC device

In addition, the embedded Pentium processor with MMX technology offers the following enhancements over the embedded Pentium processor:

- Support for Intel MMX technology
- Dual power supplies—separate V_{CC2} (core) and V_{CC3} (I/O) voltage inputs
- Separate 16-Kbyte, 4-way set-associative code and data caches, each with improved fully associative TLBs
- Pool of four write buffers used by both execution pipelines
- Enhanced branch prediction algorithm
- New Fetch pipeline stage between Prefetch and Instruction Decode

The following features are supported by the embedded Pentium processor, but are not supported by the embedded Pentium processor with MMX technology:

- Functional redundancy checking and lock-step operation
- Support for the Intel 82498/82493 and 82497/82492 cache chipset products
- Split line accesses to the code cache

The following feature is supported by the embedded Pentium processor with MMX technology, but is not supported by the low-power embedded Pentium processor with MMX technology:

- Dual processing support

2.2 Component Introduction

The application instruction set of the embedded Pentium processor family includes the complete instruction set of existing Intel Architecture processors to ensure backward compatibility, with extensions to accommodate the additional functionality of the embedded Pentium processor. All application software written for the Intel386™ and Intel486™ microprocessors runs on the embedded Pentium processor without modification. The on-chip Memory Management Unit (MMU) is completely compatible with Intel386 and Intel486 processors.

The embedded Pentium processor with MMX technology adds 57 new instructions and four new data types to accelerate the performance of multimedia and communications software. 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. To take advantage of the MMX instructions, software modifications must be made. When the MMX instructions are not used, no hardware or software modifications are needed.

The two instruction pipelines and the floating-point unit on the embedded Pentium processor 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.

The embedded Pentium processor with MMX technology adds the Fetch pipeline stage between the Prefetch and Instruction decode stages, which increases the performance capability of the processor. The embedded Pentium processor with MMX technology doubles the number of write buffers available to be used by the dual pipelines.

Branch prediction is implemented in the embedded Pentium processor. To support this, the processor has two prefetch buffers, one to prefetch code in a linear fashion, and one that prefetches code according to the Branch Target Buffer (BTB) so the needed code is almost always prefetched before it is needed for execution. The branch prediction algorithm has been enhanced on the embedded Pentium processor with MMX technology for increased accuracy.

The embedded Pentium processor includes separate code and data caches integrated on chip to meet its performance goals. Each cache on the embedded Pentium processor with MMX technology is 16 Kbytes in size, and is four-way set associative. The caches on the embedded Pentium processor (at 100/133/166 MHz) are each 8 Kbytes and two-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 of the embedded Pentium processor (at 100/133/166 MHz) are also triple ported to support snooping and split-line accesses. The embedded Pentium processor with MMX technology does not support split line accesses to the code cache. As such, its code cache tags are dual ported. 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 embedded Pentium processor has a 64-bit data bus and supports burst read and burst writeback cycles. In addition, bus cycle pipelining has been added to allow two bus cycles to be in progress simultaneously. The Memory Management Unit contains optional extensions to the architecture that allow four-Mbyte page sizes.

The embedded Pentium processor has 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.

The embedded Pentium processor features functional redundancy checking to provide maximum error detection of the processor and the interface to the processor. When functional redundancy checking is used, a second processor, the "checker" is used to execute in lockstep with the "master" processor. The checker samples the master's outputs, compares those values with the values it computes internally, and asserts an error signal when a mismatch occurs. The embedded Pentium processor with MMX technology does not support functional redundancy checking.

As more and more functions are integrated on-chip, the complexity of board-level testing is increased. To address this, the embedded Pentium processor has increased test and debug capability by implementing IEEE Boundary Scan (Standard 1149.1).

System management mode (SMM) has been implemented along with some extensions to the SMM architecture. Enhancements to the 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 2-1 is a block diagram overview of the embedded Pentium processor with MMX technology including the two instruction pipelines, the "u" pipe and the "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 instruction.

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 TLB to translate linear addresses to the physical addresses used by the data cache.

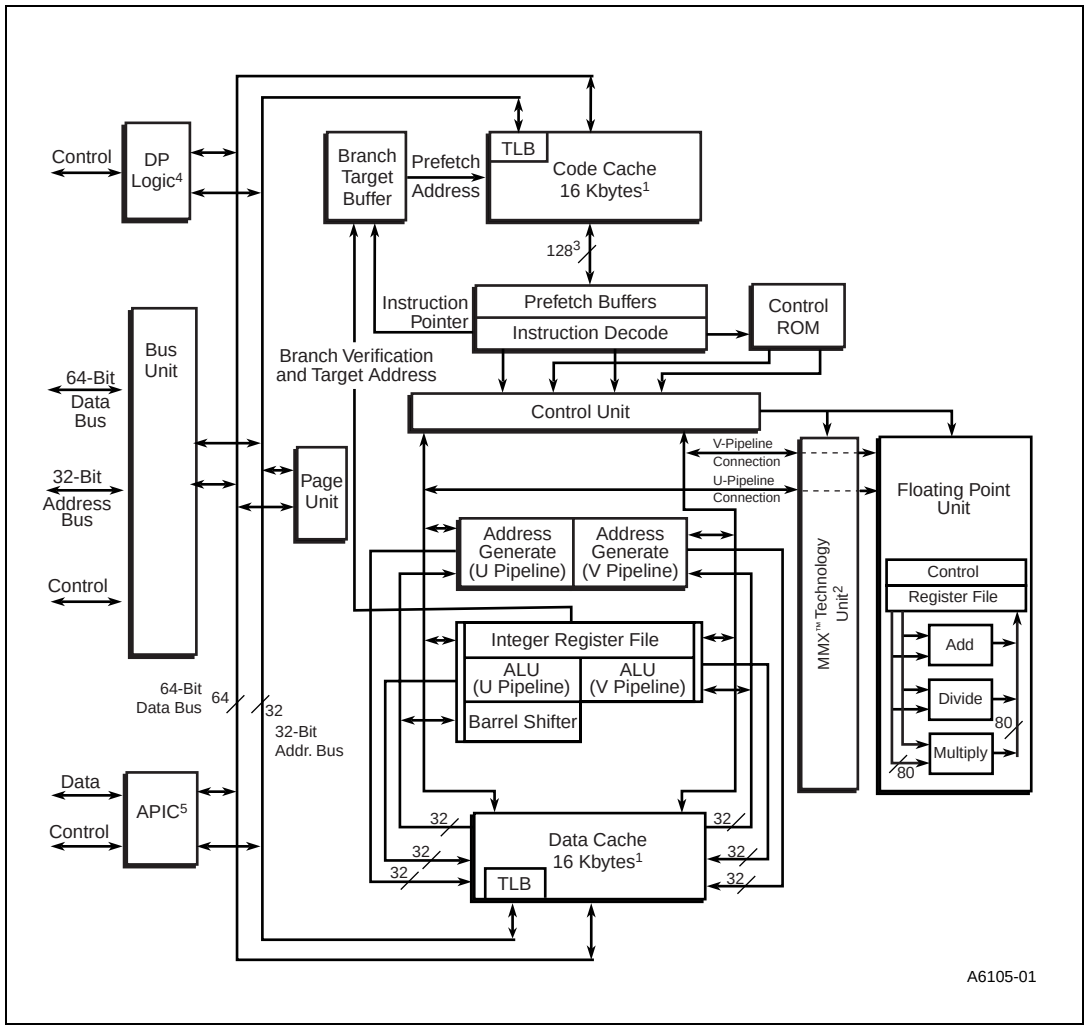
The code cache, branch target buffer and prefetch buffers are responsible for getting raw instructions into the execution units of the embedded Pentium processor. Instructions are fetched from the code cache or from the external bus. Branch addresses are remembered by the branch target buffer. The code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

The decode unit contains two parallel decoders which decode and issue up to the next two sequential instructions into the execution pipeline. The control ROM contains the microcode that controls the sequence of operations performed by the processor. The control unit has direct control over both pipelines.

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

The embedded Pentium processor includes features to support multi-processor systems, namely an on-chip Advanced Programmable Interrupt Controller (APIC). This APIC implementation supports multiprocessor interrupt management (with symmetric interrupt distribution across all processors), multiple I/O subsystem support, 8259A compatibility, and inter-processor interrupt support.

Figure 2-1. Embedded Pentium® Processor Block Diagram



NOTES:
1. The Code and Data caches are each 8 Kbytes in size on the embedded Pentium® processor (at 100/133/166 MHz).
2. The MMX Technology Unit is present only on the embedded Pentium processor with MMX™ technology.
3. The internal instruction bus is 256 bits wide on the embedded Pentium processor.
4. Dual processing is not present on the embedded Pentium processor with Voltage Reduction Technology or the low-power embedded Pentium processor with MMX technology.
5. The APIC is not present on the embedded Pentium processor with Voltage Reduction Technology.

The dual processor configuration allows two embedded Pentium processors to share a single L2 cache for a low-cost symmetric multi-processor system. The two processors appear to the system as a single embedded Pentium processor. Multiprocessor operating systems properly schedule computing tasks between the two processors. This scheduling of tasks is transparent to software applications and the end-user. Logic built into the processors support a “glueless” interface for easy system design. Through a private bus, the two embedded Pentium processors arbitrate for the external bus and maintain cache coherency. The embedded Pentium processor can also be used in a conventional multi-processor system in which one L2 cache is dedicated to each processor.

In this document, in order to distinguish between two embedded Pentium processors in dual processing mode, one processor is referred to as the Primary processor and the other as the Dual processor. Note that this is a different concept than that of “master” and “checker” processors described in the discussion on functional redundancy.

Dual processing is supported in a system only when both processors are operating at identical core and bus frequencies and are the same type of processor. Within these restrictions, two processors of different steppings may operate together in a system. See Chapter 3, “Component Operation” for more details about Dual processing.

The embedded Pentium processor is produced on Intel’s advanced silicon technology. The embedded Pentium processor also includes SL enhanced power management features. When the clock to the embedded Pentium processor is stopped, power dissipation is virtually eliminated. The low V_{CC} operating voltages and SL enhanced power management features make the embedded Pentium processor a good choice for energy-efficient designs.

The embedded 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. See the datasheet for the bus-to-core frequency ratios supported by a specific member of the embedded Pentium processor family.

Component Operation 3

The embedded Pentium® processor has an optimized superscalar micro-architecture capable of executing two instructions in a single clock. A 64-bit external bus, separate data and instruction caches, write buffers, branch prediction, and a pipelined floating-point unit combine to sustain the high execution rate. These architectural features and their operation are discussed in this chapter.

3.1 Pipeline and Instruction Flow

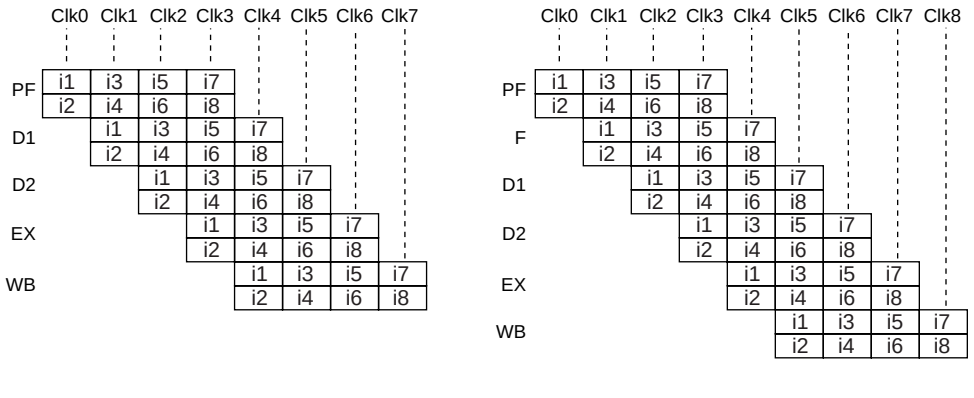
The integer instructions traverse a five stage pipeline in the embedded Pentium processor (the embedded Pentium® processor with MMX™ technology has an additional pipeline stage). The pipeline stages are as follows:

PF	Prefetch
F	Fetch (embedded Pentium processor with MMX technology only)
D1	Instruction Decode
D2	Address Generate
EX	Execute - ALU and Cache Access
WB	Writeback

The embedded Pentium processor is a superscalar machine, built around two general purpose integer pipelines and a pipelined floating-point unit capable of executing two instructions in parallel. Both pipelines operate in parallel, allowing integer instructions to execute in a single clock in each pipeline. Figure 3-1 depicts instruction flow in the embedded Pentium processor.

The pipelines in the embedded Pentium processor are called the “u” and “v” pipes and the process of issuing two instructions in parallel is termed “pairing.” The u-pipe can execute any instruction in the Intel architecture, whereas the v-pipe can execute “simple” instructions as defined in ““Pairing Two MMX™ Instructions” on page 3-16” section of this chapter. When instructions are paired, the instruction issued to the v-pipe is always the next sequential instruction after the one issued to the u-pipe.

Figure 3-1. Embedded Pentium® Processor Pipeline Execution



NOTE: i1 refers to instruction 1

A6103-01

3.1.1 Integer Pipeline Description

The embedded Pentium processor pipeline has been optimized to achieve higher throughput compared to previous generations of Intel architecture processors.

The first stage of the pipeline is the Prefetch (PF) stage in which instructions are prefetched from the on-chip instruction cache or memory. Because the processor has separate caches for instructions and data, prefetches do not conflict with data references for access to the cache. If the requested line is not in the code cache, a memory reference is made. In the PF stage, two independent pairs of line-size (32-byte) prefetch buffers operate in conjunction with the branch target buffer. This allows one prefetch buffer to prefetch instructions sequentially while the other prefetches according to the branch target buffer predictions. The prefetch buffers alternate their prefetch paths. In the embedded Pentium processor with MMX technology, four 16-byte prefetch buffers operate in conjunction with the BTB to prefetch up to four independent instruction streams. See the “Instruction Prefetch” on page 3-3 for further details on prefetch buffers.

In the embedded Pentium processor with MMX technology only, the next pipeline stage is Fetch (F), which is used for instruction length decode. It replaces the D1 instruction-length decoder and eliminates the need for end-bits to determine instruction length. Also, any prefixes are decoded in the F stage. The Fetch stage is not supported by the embedded Pentium processor (at 100, 133, 166 MHz) or the embedded Pentium processor with VRT.

The embedded Pentium processor with MMX technology also features an instruction FIFO between the F and D1 stages. This FIFO is transparent; it does not add additional latency when it is empty. During every clock cycle, two instructions can be pushed into the instruction FIFO (depending on availability of the code bytes, and on other factors such as prefixes). Instruction pairs are pulled out of the FIFO into the D1 stage. Since the average rate of instruction execution is less than two per clock, the FIFO is normally full. As long as the FIFO is full, it can buffer any stalls that may occur during instruction fetch and parsing. If such a stall occurs, the FIFO prevents the stall from causing a stall in the execution stage of the pipe. If the FIFO is empty, an execution

stall may result from the pipeline being “starved” for instructions to execute. Stalls at the FIFO entrance may be caused by long instructions or prefixes, or “extremely misaligned targets” (i.e., Branch targets that reside at the last bytes of 16-aligned bytes).

The pipeline stage after the PF stage in the embedded Pentium processor is Decode1 (D1), in which two parallel decoders work to decode and issue the next two sequential instructions. The decoders determine whether one or two instructions can be issued contingent upon the instruction pairing rules described in “Pairing Two MMX™ Instructions” on page 3-16.” The embedded Pentium processor requires an extra D1 clock to decode instruction prefixes. Prefixes are issued to the u-pipe at the rate of one per clock without pairing. After all prefixes have been issued, the base instruction is issued and paired according to the pairing rules. The one exception to this is that the embedded Pentium processor decodes near conditional jumps (long displacement) in the second opcode map (0FH prefix) in a single clock in either pipeline. The embedded Pentium processor with MMX technology handles 0FH as part of the opcode and not as a prefix. Consequently, 0FH does not take one extra clock to get into the FIFO. Note that in the embedded Pentium processor with MMX technology, MMX instructions can be paired. This is discussed in “Pairing Two MMX™ Instructions” on page 3-16.

The D1 stage is followed by Decode2 (D2) in which addresses of memory resident operands are calculated. In the Intel486™ processor, instructions containing both a displacement and an immediate or instructions containing a base and index addressing mode require an additional D2 clock to decode. The embedded Pentium processor removes both of these restrictions and is able to issue instructions in these categories in a single clock.

The embedded Pentium processor uses the Execute (EX) stage of the pipeline for both ALU operations and for data cache access; therefore, those instructions specifying both an ALU operation and a data cache access require more than one clock in this stage. In EX, all u-pipe instructions and all v-pipe instructions except conditional branches are verified for correct branch prediction. Microcode is designed to utilize both pipelines; therefore, those instructions requiring microcode execute faster.

The final stage is Writeback (WB), in which instructions are enabled to modify the processor state and complete execution. In this stage, v-pipe conditional branches are verified for correct branch prediction.

During their progression through the pipeline, instructions may be stalled due to certain conditions. Both the u-pipe and v-pipe instructions enter and leave the D1 and D2 stages in unison. When an instruction in one pipe is stalled, the instruction in the other pipe is also stalled at the same pipeline stage. Thus both the u-pipe and the v-pipe instructions enter the EX stage in unison. Once in EX, if the u-pipe instruction is stalled, then the v-pipe instruction (if any) is also stalled. If the v-pipe instruction is stalled, then the instruction paired with it in the u-pipe is not allowed to advance. No successive instructions are allowed to enter the EX stage of either pipeline until the instructions in both pipelines have advanced to WB.

3.1.1.1 Instruction Prefetch

In the embedded Pentium processor PF stage, two independent pairs of line-size (32-byte) prefetch buffers operate in conjunction with the branch target buffer. Only one prefetch buffer actively requests prefetches at any given time. Prefetches are requested sequentially until a branch instruction is fetched. When a branch instruction is fetched, the branch target buffer (BTB) predicts whether the branch will be taken or not. If the branch is predicted not taken, prefetch requests continue linearly. On a predicted taken branch the other prefetch buffer is enabled and begins to prefetch as though the branch were taken. If a branch is discovered mispredicted, the instruction pipelines are flushed and prefetching activity starts over.

The embedded Pentium processor with MMX technology's prefetch stage has four 16-byte buffers that can prefetch up to four independent instruction streams, based on predictions made by the BTB. In this case, the Branch Target Buffer predicts whether the branch will be taken or not in the PF stage. The embedded Pentium processor with MMX technology features an enhanced two-stage Branch prediction algorithm, compared to the embedded Pentium processor.

For more information on branch prediction, see "Component Introduction" on page 2-3.

3.1.2 Integer Instruction Pairing Rules

The embedded Pentium processor can issue one or two instructions every clock. In order for the processor to issue two instructions simultaneously, they must satisfy the following conditions:

- Both instructions in the pair must be "simple" as defined below.
- There must be no read-after-write or write-after-write register dependencies between the instructions.
- Neither instruction may contain both a displacement and an immediate.
- Instructions with prefixes can only occur in the u-pipe (except for JCC instructions with a 0FH prefix on the embedded Pentium processor and instructions with a 0FH, 66H or 67H prefix on the embedded Pentium processor with MMX technology).
- Instruction prefixes are treated as separate 1-byte instructions (except for all 0FH prefixed instructions in the embedded Pentium processor with MMX technology).

Simple instructions are entirely hardwired; they do not require any microcode control and, in general, execute in one clock. The exceptions are the ALU mem,reg and ALU reg,mem instructions which are three and two clock operations, respectively. Sequencing hardware is used to allow them to function as simple instructions. The following integer instructions are considered simple and may be paired:

- mov reg, reg/mem/imm
- mov mem, reg/imm
- alu reg, reg/mem/imm
- alu mem, reg/imm
- inc reg/mem
- dec reg/mem
- push reg/mem
- pop reg
- lea reg,mem
- jmp/call/jcc near
- nop
- test reg, reg/mem
- test acc, imm

In addition, conditional and unconditional branches may be paired only if they occur as the second instruction in the pair. They may not be paired with the next sequential instruction. Also, SHIFT/ROT by 1 and SHIFT by IMM may pair as the first instruction in a pair.

The register dependencies that prohibit instruction pairing include implicit dependencies via registers or flags not explicitly encoded in the instruction. For example, an ALU instruction in the u-pipe (which sets the flags) may not be paired with an ADC or an SBB instruction in the v-pipe. There are two exceptions to this rule. The first is the commonly occurring sequence of compare and branch, which may be paired. The second exception is pairs of pushes or pops. Although these instructions have an implicit dependency on the stack pointer, special hardware is included to allow these common operations to proceed in parallel.

Although two paired instructions generally may proceed in parallel independently, there is an exception for paired “read-modify-write” instructions. Read-modify-write instructions are ALU operations with an operand in memory. When two of these instructions are paired, there is a sequencing delay of two clocks in addition to the three clocks required to execute the individual instructions.

Although instructions may execute in parallel, their behavior as seen by the programmer is exactly the same as if they were executed sequentially.

Information regarding pairing of FPU and MMX instructions is discussed in “Floating-Point Unit” on page 3-7 and “Intel MMX™ Technology Unit” on page 3-11 For additional details on code optimization, refer to *Optimizing for Intel’s 32-Bit Processors* (order number 241799).

3.2 Branch Prediction

The embedded Pentium processor uses a Branch Target Buffer (BTB) to predict the outcome of branch instructions, thereby minimizing pipeline stalls due to prefetch delays.

The processor accesses the BTB with the address of the instruction in the D1 stage. It contains a Branch prediction state machine with four states: (1) strongly not taken, (2) weakly not taken, (3) weakly taken, and (4) strongly taken. In the event of a correct prediction, a branch executes without pipeline stalls or flushes. Branches that miss the BTB are assumed to be not taken. Conditional and unconditional near branches and near calls execute in one clock and may be executed in parallel with other integer instructions. A mispredicted branch (whether a BTB hit or miss) or a correctly predicted branch with the wrong target address causes the pipelines to be flushed and the correct target to be fetched. Incorrectly predicted unconditional branches incur an additional three clock delay, incorrectly predicted conditional branches in the u-pipe incur an additional three clock delay, and incorrectly predicted conditional branches in the v-pipe incur an additional four clock delay.

The benefits of branch prediction are illustrated in the following example. Consider the following loop from a benchmark program for computing prime numbers:

```
for(k=i+prime;k<=SIZE;k+=prime)
    flags[k]=FALSE;
```

A popular compiler generates the following assembly code (prime is allocated to ECX, K is allocated to EDX, and AL contains the value FALSE):

```
inner_loop:
    mov byte ptr flags[edx],al
    add edx,ecx
    cmp edx, SIZE
    jle inner_loop
```

Each iteration of this loop executes in six clocks on the Intel486™ processor. On the embedded Pentium processor, the MOV is paired with the ADD; the CMP with the JLE. With branch prediction, each loop iteration executes in two clocks.

Note: The dynamic branch prediction algorithm speculatively runs code fetch cycles to addresses corresponding to instructions executed some time in the past. Such code fetch cycles are run based on past execution history, regardless of whether the instructions retrieved are relevant to the currently executing instruction sequence.

One effect of the branch prediction mechanism is that the processor may run code fetch bus cycles to retrieve instructions that are never executed. Although the opcodes retrieved are discarded, the system must complete the code fetch bus cycle by returning BRDY#. It is particularly important that the system return BRDY# for all code fetch cycles, regardless of the address.

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

Furthermore, the processor may run speculative code fetch cycles to addresses beyond the end of the current code segment (approximately 100 bytes past end of last executed instruction). Although the processor may prefetch beyond the CS limit, it will not attempt to execute beyond the CS limit. Instead, it will raise a GP fault. Thus, segmentation cannot be used to prevent speculative code fetches to inaccessible areas of memory. On the other hand, the processor never runs code fetch cycles to inaccessible pages (i.e., not present pages or pages with incorrect access rights), so the paging mechanism guards against both the fetch and execution of instructions in inaccessible pages.

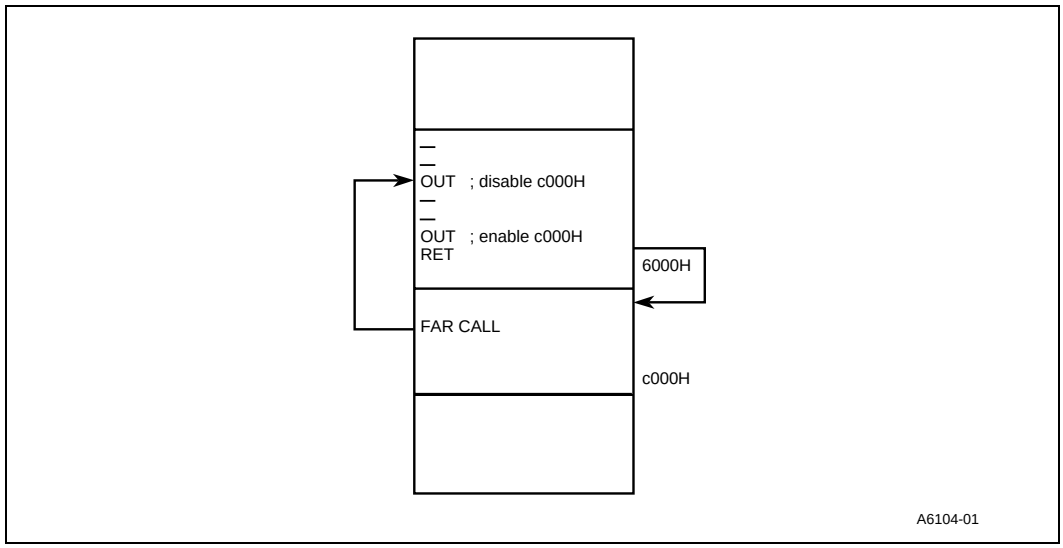
For memory reads and writes, both segmentation and paging prevent the generation of bus cycles to inaccessible regions of memory. If paging is not used, branch prediction can be disabled by setting TR12.NBP (bit 0)[1] and flushing the BTB by loading CR3 before disabling any areas of memory. Branch prediction can be re-enabled after re-enabling memory.

The following is an example of a situation that may occur:

1. Code passes control to segment at address C000H.
2. Code transfers control to code at different address (6000H) by using the FAR CALL instruction.
3. This portion of the code does an I/O write to a port that disables memory at address C000H.
4. At the end of this segment, an I/O write is performed to re-enable memory at address C000H.
5. Following the OUT instruction, there is a RET instruction to C000H segment.

1. Please refer to Chapter 14 of this volume.

Figure 3-2. Branch Prediction Example



The branch prediction mechanism of the embedded Pentium processor, however, predicts that the RET instruction is going to transfer control to the segment at address C000H and performs a prefetch from that address prior to the OUT instruction that re-enables that memory address. The result is that no BRDY is returned for that prefetch cycle and the system hangs.

In this case, branch prediction should be disabled (by setting TR12.NBP and flushing the BTB by loading CR3) prior to disabling memory at address C000H, and re-enabled after the RET instruction by clearing TR12.NBP as indicated above. (See Chapter 14, “Model Specific Registers and Functions” for more information on register operation.)

In the embedded Pentium processor with MMX technology, the branch prediction algorithm changes from the embedded Pentium processor in the following ways:

- BTB Lookup is done when the branch is in the PF stage.
- The BTB Lookup tag is the Prefetch address.
- A Lookup in the BTB performs a search spanning sixteen consecutive bytes.
- BTB can contain four branch instructions for each line of 16 bytes.
- BTB is constructed from four independent Banks. Each Bank contains 64 entries and is 4-way associative.
- Enhanced two-stage branch prediction algorithm.

3.3 Floating-Point Unit

The floating-point unit (FPU) of the embedded Pentium processor is integrated with the integer unit on the same chip. It is heavily pipelined. The FPU is designed to be able to accept one floating-point operation every clock. It can receive up to two floating-point instructions every clock, one of which must be an exchange instruction.

For information on code optimization, please refer to *Optimizing for Intel’s 32-Bit Processors* (order number 241799).

3.3.1 Floating-Point Pipeline Stages

The embedded Pentium processor FPU has eight pipeline stages, the first five of which it shares with the integer unit. Integer instructions pass through only the first five stages. Integer instructions use the fifth (X1) stage as a WB (write-back) stage. The eight FP pipeline stages, and the activities that are performed in them are summarized below:

PF	Prefetch
F	Fetch (applicable to the embedded Pentium processor with MMX technology only)
D1	Instruction decode
D2	Address generation
EX	Memory and register read; conversion of FP data to external memory format and memory write
X1	Floating-Point Execute stage one; conversion of external memory format to internal FP data format and write operand to FP register file; bypass 1 (bypass 1 is described in “FPU Bypasses” on page 3-10)
X2	Floating-Point Execute stage two
WF	Perform rounding and write floating-point result to register file; bypass 2 (bypass 2 is described in “FPU Bypasses” on page 3-10)
ER	Error Reporting/Update Status Word

3.3.2 Instruction Issue

The rules of how floating-point (FP) instructions get issued on the embedded Pentium processor are described as follows:

1. FP instructions do not get paired with integer instructions. However, a limited pairing of two FP instructions can be performed.
2. When a pair of FP instructions is issued to the FPU, only the FXCH instruction can be the second instruction of the pair. The first instruction of the pair must be one of a set F where F = [FLD single/double, FLD ST(i), all forms of FADD, FSUB, FMUL, FDIV, FCOM, FUCOM, FTST, FABS, FCHS].
3. FP instructions other than the FXCH instruction and other than instructions belonging to set F (defined in rule 2) always get issued singly to the FPU.
4. FP instructions that are not directly followed by an FP exchange instruction are issued singly to the FPU.

The embedded Pentium processor stack architecture instruction set requires that all instructions have one source operand on the top of the stack. Since most instructions also have their destination as the top of the stack, most instructions see a “top of stack bottleneck.” New source operands must be brought to the top of the stack before we can issue an arithmetic instruction on them. This calls for extra usage of the exchange instruction, which allows the programmer to bring an available operand to the top of the stack. The processor FPU uses pointers to access its registers to allow fast execution of exchanges and the execution of exchanges in parallel with other floating-point instructions. An FP exchange that is paired with other FP instructions takes zero clocks for its execution. Because such exchanges can be executed in parallel, it is recommended that one use them when necessary to overcome the stack bottleneck.

Note that when exchanges are paired with other floating-point instructions, they should not be followed immediately by integer instructions. The processor stalls such integer instructions for a clock if the FP pair is declared safe, or for four clocks if the FP pair is unsafe.

Also note that the FP exchange must always follow another FP instruction to get paired. The pairing mechanism does not allow the FP exchange to be the first instruction of a pair that is issued in parallel. If an FP exchange is not paired, it takes one clock for its execution.

3.3.3 Safe Instruction Recognition

The embedded Pentium processor FPU performs Safe Instruction Recognition or SIR in the X1 stage of the pipeline. SIR is an early inspection of operands and opcodes to determine whether the instruction is guaranteed not to generate an arithmetic overflow, underflow, or unmasked inexact exception. An instruction is declared safe if it cannot raise any other floating-point exception, and if it does not need microcode assist for delivery of special results. If an instruction is declared safe, the next FP instruction is allowed to complete its E stage operation. If an instruction is declared unsafe, the next FP instruction stalls in the E stage until the current one completes (ER stage) with no exception. This means a four clock stall, which is incurred even if the numeric instruction that was declared unsafe does not eventually raise a floating-point exception.

For normal data, the rules used on the embedded Pentium processor for declaring an instruction safe are as follows.

On the embedded Pentium processor, if FOP = FADD/FSUB/FMUL/FDIV, the instruction is safe from arithmetic overflow, underflow, and unmasked inexact exceptions if:

1. Both operands have unbiased exponent ≤1FFEH
   and
2. Both operands have unbiased exponent ≥–1FFEH
   and
3. The inexact exception is masked.

Similarly, on the embedded Pentium processor with MMX technology, if FOP = FADD/FSUB/FMUL/FDIV, the instruction is safe from arithmetic overflow, underflow, and unmasked inexact exceptions if:

1. Both operands have unbiased exponent ≤1000H
   and
2. Both operands have unbiased exponent ≥–0FFFH
   and
3. The inexact exception is masked.

Note that arithmetic overflow of the double precision format occurs when the unbiased exponent of the result is ≥400H, and underflow occurs when the exponent is ≤–3FFH. Hence, the SIR algorithm on the embedded Pentium processor allows improved throughput on a much greater range of numbers than that spanned by the double precision format.

3.3.4 FPU Bypasses

The following section describes the floating-point register file bypasses that exist on the embedded Pentium processor. The register file has two write ports and two read ports. The read ports are used to read data out of the register file in the E stage. One write port is used to write data into the register file in the X1 stage, and the other in the WF stage. A bypass allows data that is about to be written into the register file to be available as an operand that is to be read from the register file by any succeeding floating-point instruction. A bypass is specified by a pair of ports (a write port and a read port) that get circumvented. Using the bypass, data is made available even before actually writing it to the register file.

The following procedures are implemented:

1. Bypass the X1 stage register file write port and the E stage register file read port.
2. Bypass the WF stage register file write port and the E stage register file read port.

With bypass 1, the result of a floating-point load (that writes to the register file in the X1 stage) can bypass the X1 stage write and be sent directly to the operand fetch stage or E stage of the next instruction.

With bypass 2, the result of any arithmetic operation can bypass the WF stage write to the register file, and be sent directly to the desired execution unit as an operand for the next instruction.

Note that the FST instruction reads the register file with a different timing requirement, so that for the FST instruction, which attempts to read an operand in the E stage:

1. There is no bypassing the X1 stage write port and the E stage read port, i.e., no added bypass for FLD followed by FST. Thus FLD (double) followed by FST (double) takes four clocks (two for FLD, and two for FST).
2. There is no bypassing the WF stage write port and the E stage read port. The E stage read for the FST happens only in the clock following the WF write for any preceding arithmetic operation.

Furthermore, there is no memory bypass for an FST followed by an FLD from the same memory location.

3.3.5 Branching Upon Numeric Condition Codes

Branching upon numeric condition codes is accomplished by transferring the floating-point SW to the integer FLAGS register and branching on it. The “test numeric condition codes and branch” construct looks like:

```
FP instruction1; instruction whose effects on the status word are to be examined;
“numeric_test_and_branch_construct”:
FSTSW AX; move the status word to the ax register.
SAHF; transfer the value in ah to the lower half of the eflags register.
JC xyz; jump upon the condition codes in the eflags register.
```

Note that all FP instructions update the status word only in the ER stage. Hence there is a built-in status word interlock between FP instruction1 and the FSTSW AX instruction. The above piece of code takes nine clocks before execution of code begins at the target of the jump. These nine clocks are counted as:

FP instruction1:	X1, X2, WF, ER (4 E stage stalls for the FSTSWAX);
FSTSW AX:	Two E clocks;
SAHF:	Two E clocks;
JC xyz:	One clock if no mispredict on branch.

Note that if there is a branch mispredict, there is a minimum of three clocks added to the clock count of nine.

It is recommended that such attempts to branch upon numeric condition codes be preceded by integer instructions; i.e., you should insert integer instructions in between FP instruction1 and the FSTSW AX instruction that is the first instruction of the “numeric test and branch” construct. This allows the elimination of up to four clocks (the 4 E-stage stalls on FSTSW AX) from the cost attributed to this construct, so that numeric branching can be accomplished in five clocks.

3.4 Intel MMX™ Technology Unit

Intel’s MMX technology, supported on the embedded Pentium processor with MMX technology, is a set of extensions to the Intel architecture that are designed to greatly enhance the performance of advanced media and communications applications. These extensions (which include new registers, data types, and instructions) are combined with a single-instruction, multiple-data (SIMD) execution model to accelerate the performance of applications such as motion video, combined graphics with video, image processing, audio synthesis, speech synthesis and compression, telephony, video conferencing, and 2D and 3D graphics, which typically use compute-intensive algorithms to perform repetitive operations on large arrays of simple, native data elements.

MMX technology defines a simple and flexible software model, with no new mode or operating-system visible state. All existing software runs correctly, without modification, on Intel architecture processors that incorporate MMX technology, even in the presence of existing and new applications that incorporate this technology.

The following sections of this chapter describe the basic programming environment for the technology, the MMX technology register set, data types and instruction set. Detailed descriptions of the MMX instructions are provided in Chapter 3 of the *Intel Architecture Software Developer’s Manual*, Volume 2. The manner in which the MMX technology extensions fit into the Intel architecture system programming model is described in Chapter 10 of the *Intel Architecture Software Developer’s Manual*, Volume 3.

3.4.1 MMX™ Technology Programming Environment

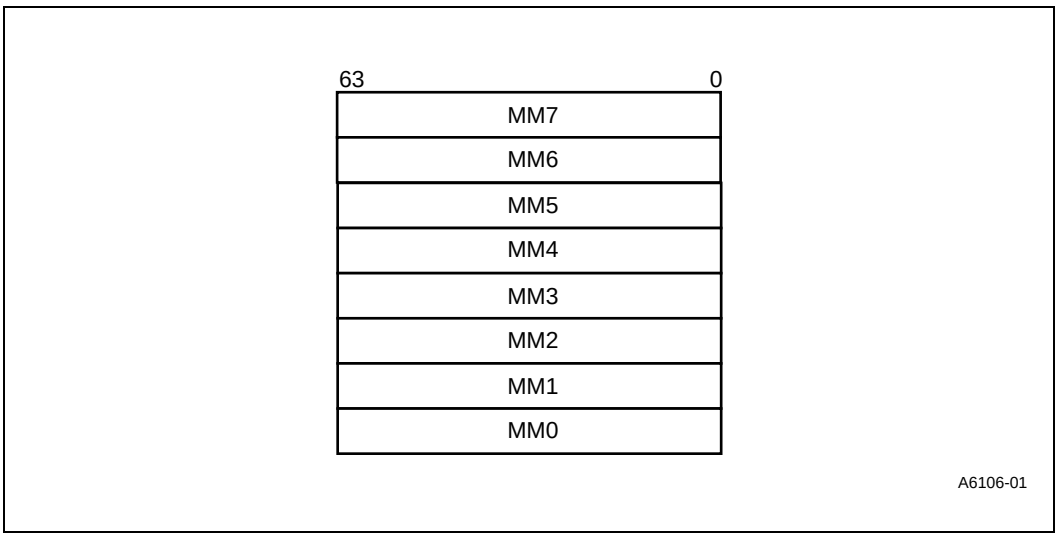
MMX technology provides the following new extensions to the Intel architecture programming environment:

- Eight MMX technology registers (MM0 through MM7)
- Four MMX technology data types (packed bytes, packed words, packed doublewords and quadword)
- The MMX technology instruction set

3.4.1.1 MMX™ Technology Registers

The MMX technology register set consists of eight 64-bit registers (Figure 3-3). The MMX instructions access the registers directly using the register names MM0 through MM7. These registers can only be used to perform calculations on MMX technology data types; they cannot be used to address memory. Addressing of MMX instruction operands in memory is handled by using the standard Intel architecture addressing modes and general-purpose registers (EAX, EBX, ECX, EDX, EBP, ESI, EDI and ESP).

Figure 3-3. MMX™ Technology Register Set



Although the MMX registers are defined in the Intel architecture as separate registers, they are aliased to the registers in the FPU data register stack (R0 through R7). (See Chapter 10 in the *Intel Architecture Software Developer's Manual*, Volume 3, for a more detailed discussion of MMX technology register aliasing.)

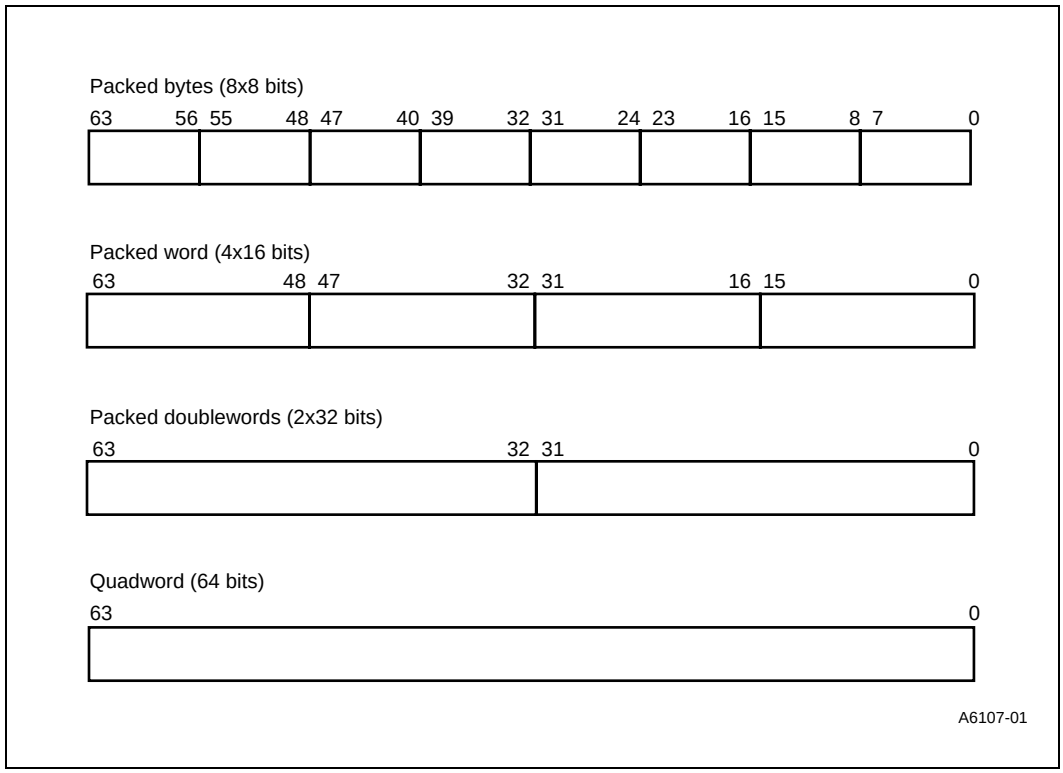
3.4.1.2 MMX™ Technology Data Types

The MMX technology defines the following new 64-bit data types (Figure 3-4):

Packed bytes	Eight bytes packed into one 64-bit quantity.
Packed words	Four (16-bit) words packed into one 64-bit quantity.
Packed doublewords	Two (32-bit) doublewords packed into one 64-bit quantity.
Quadword	One 64-bit quantity.

The bytes in the packed bytes data type are numbered 0 through 7. Byte 0 is contained in the least significant bits of the data type (bits 0 through 7) and byte 7 is contained in the most significant bits (bits 56 through 63). The words in the packed words data type are numbered 0 through 4. Word 0 is contained in the bits 0 through 15 of the data type and word 4 is contained in bits 48 through 63. The doublewords in a packed doublewords data type are numbered 0 through 1. Doubleword 0 is contained in bits 0 through 31 and doubleword 1 is contained in bits 32 through 63.

Figure 3-4. Packed Data Types



The MMX instructions move the packed data types (packed bytes, packed words or packed doublewords) and the quadword data type to-and-from memory or to-and-from the Intel architecture general-purpose registers in 64-bit blocks. However, when performing arithmetic or logical operations on the packed data types, the MMX instructions operate in parallel on the individual bytes, words or doublewords contained in a 64-bit MMX register.

When operating on the bytes, words and doublewords within packed data types, the MMX instructions recognize and operate on both signed and unsigned byte integers, word integers and doubleword integers.

3.4.1.3 Single Instruction, Multiple Data (SIMD) Execution Model

The MMX technology uses the single instruction, multiple data (SIMD) technique for performing arithmetic and logical operations on the bytes, words or doublewords packed in an MMX packed data type. For example, the PADDSB instruction adds eight signed bytes from the source operand to eight signed bytes in the destination operand and stores eight byte-results in the destination operand. This SIMD technique speeds up software performance by allowing the same operation to be carried out on multiple data elements in parallel. The MMX technology supports parallel operations on byte, word and doubleword data elements when contained in MMX packed data types.

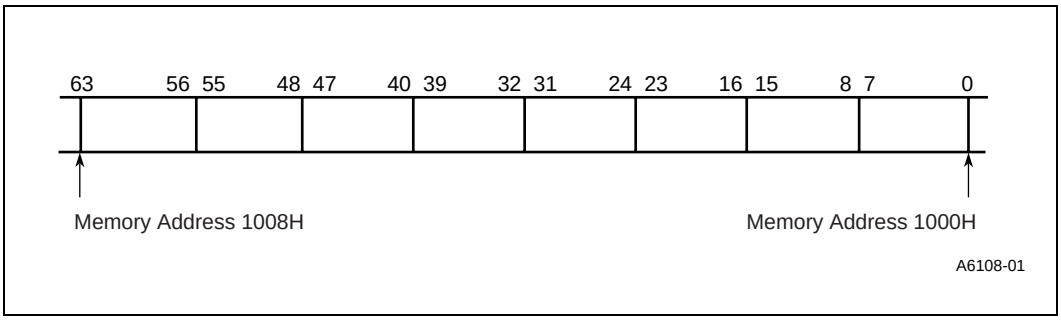
The SIMD execution model supported in the MMX technology directly addresses the needs of modern media, communications and graphics applications, which often use sophisticated algorithms that perform the same operations on a large number of small data types (bytes, words and doublewords). For example, most audio data is represented in 16-bit (word) quantities. The

MMX instructions can operate on four of these words simultaneously with one instruction. Video and graphics information is commonly represented as palletized 8-bit (byte) quantities. Here, one MMX instruction can operate on eight of these bytes simultaneously.

3.4.1.4 Memory Data Formats

When stored in memory the bytes, words and doublewords in the packed data types are stored in consecutive addresses, with the least significant byte, word or doubleword being stored at the lowest address and the more significant bytes, words or doublewords being stored at consecutively higher addresses (see Figure 3-5). The ordering of bytes, words or doublewords in memory is always little endian. That is, the bytes with the lower addresses are less significant than the bytes with the higher addresses.

Figure 3-5. Eight Packed Bytes in Memory (at Address 1000H)



3.4.1.5 MMX™ Technology Register Data Formats

Values in MMX registers have the same format as a 64-bit quantity in memory. MMX registers have two data access modes: 64-bit access mode and 32-bit access mode.

The 64-bit access mode is used for 64-bit memory access, 64-bit transfer between registers, all pack, logical and arithmetic instructions, and some unpack instructions.

The 32-bit access mode is used for 32-bit memory access, 32-bit transfer between integer registers and MMX technology registers, and some unpack instructions.

3.4.2 MMX™ Instruction Set

The MMX instruction set consists of 57 instructions, grouped into the following categories:

- Data Transfer Instructions
- Arithmetic Instructions
- Comparison Instructions
- Conversion Instructions
- Logical Instructions
- Shift Instructions
- Empty MMX State (EMMS) Instruction

These instructions provide a rich set of operations that can be performed in parallel on the bytes, words or doublewords of an MMX packed data type.

When operating on the MMX packed data types, the data within a data type is cast by the type specified by the instruction. For example, the PADDB (add packed bytes) instruction adds two groups of eight packed bytes. The PADDW (add packed words) instruction, which adds packed words, can operate on the same 64 bits as the PADDB instruction treating 64 bits as four 16-bit words.

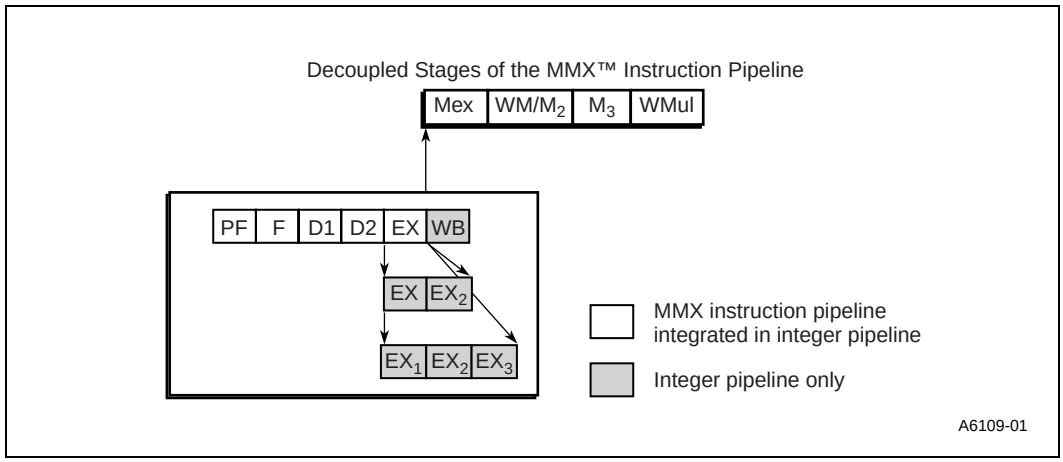
3.4.3 Intel MMX™ Technology Pipeline Stages

The MMX technology unit of the embedded Pentium processor with MMX technology has six pipeline stages. The integration of the MMX technology pipeline with the integer pipeline is very similar to that of the floating-point pipe.

The embedded Pentium processor with MMX technology adds an additional fetch stage to the pipeline. The instruction bytes are prefetched from the code cache in the prefetch (PF) stage, and they are parsed into instructions (and prefixes) in the fetch (F) stage. Additionally, any prefixes are decoded in the F stage.

When instructions execute in the two pipes, their behavior is exactly the same as if they were executed sequentially. When a stall occurs, successive instructions are not allowed to pass the stalled instruction in either pipe. Figure 3-6 shows the pipelining structure for this scheme.

Figure 3-6. MMX™ Technology Pipeline Structure



Instruction parsing is decoupled from the instruction decoding by means of an instruction FIFO, which is situated between the F and D1 (Decode 1) stages. The FIFO has slots for up to four instructions. This FIFO is transparent, it does not add additional latency when it is empty.

Every clock cycle, two instructions can be pushed into the instruction FIFO (depending on the availability of the code bytes, and on other factors such as prefixes). Instruction pairs are pulled out of the FIFO into the D1 stage. Since the average rate of instruction execution is less than two per clock, the FIFO is normally full. If the FIFO is full, then the FIFO can buffer a stall that may have occurred during instruction fetch and parsing. If this occurs, then that stall will not cause a stall in the execution stage of the pipe. If the FIFO is empty, then an execution stall may result from the pipeline being “starved” for instructions to execute. Also, if the FIFO contains only one instruction, then the instruction will not pair. Additionally, if an instruction is longer than 7 bytes, then only one instruction will be pushed into the FIFO. Figure 3-6 details the MMX pipeline on superscalar processors and the conditions where a stall may occur in the pipeline.

Table 3-1. Pipeline Stage Summary

Pipeline Stage	Abbreviation	Description
Prefetch	PF	Prefetches instructions
Fetch	F	The prefetched instruction bytes are passed into instructions. The prefixes are decoded and up to two instructions are pushed into the FIFO. Two MMX instructions can be pushed if each of the instructions are less than seven in bytes length.
Decode1	D1	Integer, floating-point and MMX instructions are decoded in the D1 pipe stage.
Decode2	D2	Source values are read.
Execution	E	The instruction is committed for execution.
MMX Execution	Mex	Execution clock for MMX instructions. ALU, shift, pack, and unpack instructions are executed and completed in this clock. First clock of multiply instructions. No stall conditions.
Write/Multiply2	WM/M_2	Single clock operations are written. Second stage of multiplier pipe. No stall conditions.
Multiply3	M_3	Third stage of multiplier pipe. No stall conditions.
Write of multiply	Wmul	Write of multiplier result. No stall conditions.

3.4.4 Instruction Issue

The rules of how MMX instructions get issued on the embedded Pentium processor with MMX technology are summarized as follows:

- Pairing of two MMX instructions can be performed.
- Pairing of one MMX instruction with an integer instruction can be performed.
- MMX instructions do not get paired with floating-point instructions.

3.4.4.1 Pairing Two MMX™ Instructions

The rules of how two MMX instructions can be paired are listed below:

- Two MMX instructions that both use the MMX shifter unit (pack, unpack and shift instructions) cannot pair since there is only one MMX shifter unit. Shift operations may be issued in either the u-pipe or the v-pipe but not in both in the same clock cycle.
- Two MMX instructions that both use the MMX multiplier unit (PMULL, PMULH, PMADD type instructions) cannot pair since there is only one MMX multiplier unit. Multiply operations may be issued in either the u-pipe or the v-pipe but not in both in the same clock cycle.
- MMX instructions that access either memory or the integer register file can be issued in the u-pipe only. Do not schedule these instructions to the v-pipe as they will wait and be issued in the next pair of instructions (and to the u-pipe).
- The MMX destination register of the u-pipe instruction should not match the source or destination register of the v-pipe instruction (dependency check).
- The EMMS instruction is not pairable.
- If either the CR0.TS or the CR0.EM bits are set, MMX instructions cannot go into the v-pipe.

3.4.4.2 Pairing an Integer Instruction in the U-pipe with an MMX Instruction in the V-pipe

The rules of how an integer instruction in the u-pipe is paired with an MMX instruction in the v-pipe are listed below:

- The MMX instruction cannot be the first MMX instruction following a floating-point instruction.
- The v-pipe MMX instruction does not access either memory or the integer register file.
- The u-pipe integer instruction is a pairable u-pipe integer instruction.

3.4.4.3 Pairing an MMX Instruction in the U-pipe with an Integer Instruction in the V-pipe

The rules of how an MMX instruction in the u-pipe is paired with an integer instruction in the v-pipe are listed below:

- The v-pipe instruction is a pairable integer v-pipe instruction.
- The u-pipe MMX instruction does not access either memory or the integer register file.

3.5 On-Chip Caches

The embedded Pentium processor (at 100/133/166 MHz) implements two internal caches for a total integrated cache size of 16 Kbytes: an 8-Kbyte data cache and a separate 8-Kbyte code cache. These caches are transparent to application software to maintain compatibility with previous Intel architecture generations. The embedded Pentium processor with MMX technology doubles the internal cache size to 32 Kbytes: a 16-Kbyte data cache and a separate 16-Kbyte code cache.

The data cache fully supports the MESI (modified/exclusive/shared/invalid) cache consistency protocol. The code cache is inherently write protected to prevent code from being inadvertently corrupted, and as a consequence supports a subset of the MESI protocol, the S (shared) and I (invalid) states.

The caches have been designed for maximum flexibility and performance. The data cache is configurable as writeback or writethrough on a line-by-line basis. Memory areas can be defined as non-cacheable by software and external hardware. Cache writeback and invalidations can be initiated by hardware or software. Protocols for cache consistency and line replacement are implemented in hardware, easing system design.

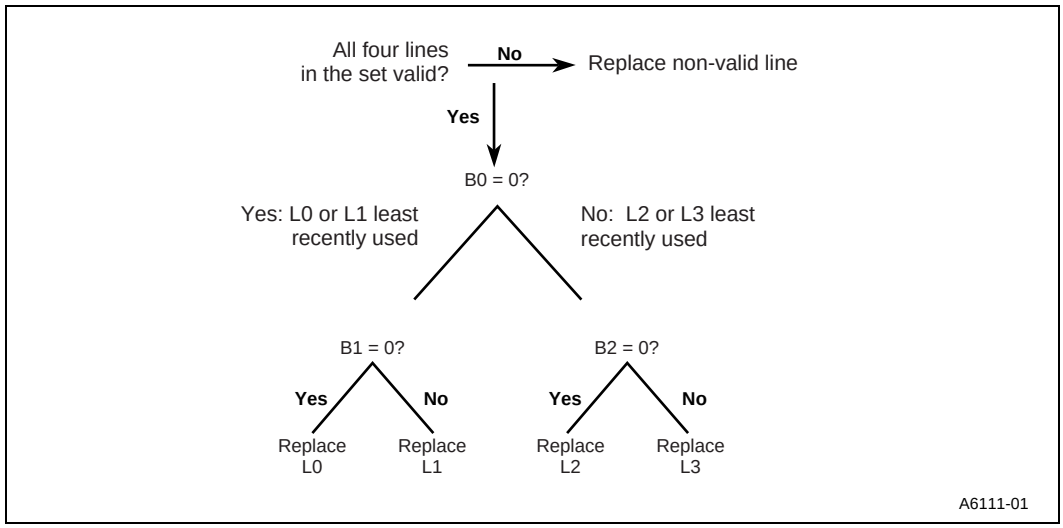
3.5.1 Cache Organization

On the embedded Pentium processor, each cache is 8 Kbytes and is organized as a 2-way set associative cache. There are 128 sets in each cache; each set contains 2 lines (each line has its own tag address). Each cache line is 32 bytes wide. The embedded Pentium processor with MMX technology has two 16-Kbyte, 4-way set-associative caches the with a line length of 32 bytes.

On the embedded Pentium processor, replacement in both the data and instruction caches is handled by the LRU mechanism, which requires one bit per set in each of the caches. The embedded Pentium processor with MMX technology uses a pseudo-LRU replacement algorithm that requires three bits per set in each of the caches. When a line must be replaced, the cache selects

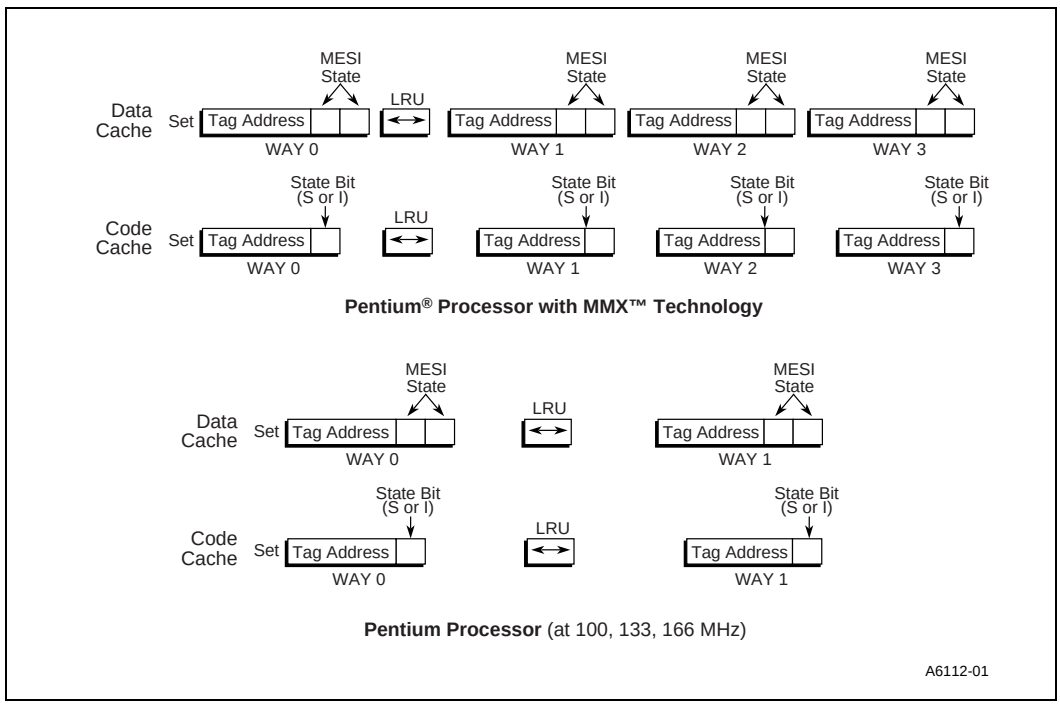
which of L0:L1 and L2:L3 was least recently used. Then the cache determines which of the two lines was least recently used and marks it for replacement. This decision tree is shown in Figure 3-7.

Figure 3-7. Pseudo-LRU Cache Replacement Strategy



The data cache consists of eight banks interleaved on 4-byte boundaries. The data cache can be accessed simultaneously from both pipes, as long as the references are to different cache banks. A conceptual diagram of the organization of the data and code caches is shown in Figure 3-8. The data cache supports the MESI writeback cache consistency protocol, which requires two state bits, while the code cache supports the S and I state only and therefore requires only one state bit.

Figure 3-8. Conceptual Organization of Code and Data Caches



3.5.2 Cache Structure

The instruction and data caches can be accessed simultaneously. The instruction cache can provide up to 32 bytes of raw opcodes and the data cache can provide data for two data references all in the same clock. This capability is implemented partially through the tag structure. The tags in the data cache are triple-ported. One of the ports is dedicated to snooping while the other two are used to lookup two independent addresses corresponding to data references from each of the pipelines. The instruction cache tags of the embedded Pentium processor (at 100/133/166 MHz) are also triple-ported. Again, one port is dedicated to support snooping and the other two ports facilitate split line accesses (simultaneously accessing upper half of one line and lower half of the next line). Note that the embedded Pentium processor with MMX technology does not support split line accesses to the code cache; its code cache tags are dual ported.

The storage array in the data cache is single ported but interleaved on 4-byte boundaries to be able to provide data for two simultaneous accesses to the same cache line.

Each of the caches are parity protected. In the instruction cache, there are parity bits on a quarter line basis and there is one parity bit for each tag. The data cache contains one parity bit for each tag and a parity bit per byte of data.

Each of the caches are accessed with physical addresses and each cache has its own TLB (translation lookaside buffer) to translate linear addresses to physical addresses. The TLBs associated with the instruction cache are single-ported whereas the data cache TLBs are fully dual-ported to be able to translate two independent linear addresses for two data references simultaneously. The tag and data arrays of the TLBs are parity protected with a parity bit associated with each of the tag and data entries in the TLBs.

The data cache of the embedded Pentium processor has a 4-way set associative, 64-entry TLB for 4-Kbyte pages and a separate 4-way set associative, 8-entry TLB to support 4-Mbyte pages. The code cache has one 4-way set associative, 32-entry TLB for 4-Kbyte pages and 4-Mbyte pages, which are cached in 4-Kbyte increments. Replacement in the TLBs is handled by a pseudo-LRU mechanism (similar to the Intel486 processor) that requires 3 bits per set. The embedded Pentium processor with MMX technology has a 64-entry fully associative data TLB and a 32-entry fully associative code TLB. Both TLBs can support 4-Kbyte pages as well as 4-Mbyte pages.

3.5.3 Cache Operating Modes

The operating modes of the caches are controlled by the CD (cache disable) and NW (not writethrough) bits in CR0. See Table 3-2 for a description of the modes. For normal operation and highest performance, these bits should both be cleared to “0.” The bits come out of RESET as CD = NW = 1.

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

Table 3-2. Cache Operating Modes

CD	NW	Description
1	1	Read hits access the cache.
		Read misses do not cause linefills.
		Write hits update the cache, but do not access memory.
		Write hits cause Exclusive State lines to change to Modified State.
		Shared lines remain in the Shared state after write hits.
		Write misses access memory.
		Inquire and invalidation cycles do not affect the cache state or contents.
		This is the state after reset.
1	0	Read hits access the cache.
		Read misses do not cause linefills.
		Write hits update the cache.
		Writes to Shared lines and write misses update external memory.
		Writes to Shared lines can be changed to the Exclusive State under the control of the WB/WT# pin.
		Inquire cycles (and invalidations) are allowed.
0	1	GP(0)
0	0	Read hits access the cache.
		Read misses may cause linefills.
		These lines will enter the Exclusive or Shared state under the control of the WB/WT# pin.
		Write hits update the cache.
		Only writes to shared lines and write misses appear externally.
		Writes to Shared lines can be changed to the Exclusive State under the control of the WB/WT# pin.
		Inquire cycles (and invalidations) are allowed.

To completely disable the cache, the following two steps must be performed:

1. CD and NW must be set to 1.
2. The caches must be flushed.

If the cache is not flushed, cache hits on reads will still occur and data will be read from the cache. In addition, the cache must be flushed after being disabled to prevent any inconsistencies with memory.

3.5.4 Page Cacheability

Two bits for cache control, PWT and PCD are defined in the page table and page directory entries. The state of these bits are driven out on the PWT and PCD pins during memory access cycles. The PWT bit controls write policy for the second-level caches used with the embedded Pentium processor. Setting PWT to 1 defines a writethrough policy for the current page, while clearing PWT to 0 defines a writeback policy for the current page.

The PCD bit controls cacheability on a page-by-page basis. The PCD bit is internally ANDed with the KEN# signal to control cacheability on a cycle-by-cycle basis. PCD = 0 enables cacheing, while PCD = 1 disables it. Cache linefills are enabled when PCD = 0 and KEN# = 0.

3.5.4.1 PCD and PWT Generation

The value driven on PCD is a function of the PWT bits in CR3, the page directory pointer, the page directory entry and the page table entry, and the CD and PG bits in CR0.

The value driven on PWT is a function of the PCD bits in CR3, the page directory pointer, the page directory entry and the page table entry, and the PG bit in CR0 (CR0.CD does not affect PWT).

CR0.CD = 1

If cacheing is disabled, the PCD pin is always driven high. CR0.CD does not affect the PWT pin.

CR0.PG = 0

If paging is disabled, the PWT pin is forced low and the PCD pin reflects the CR0.CD. The PCD and PWT bits in CR3 are assumed 0 during the caching process.

CR0.CD = 0, PG = 1, normal operation

The PCD and PWT bits from the last entry (can be either PDE or PTE, depends on 4 Mbyte or 4 Kbyte mode) are cached in the TLB and are driven anytime the page mapped by the TLB entry is referenced.

CR0.CD = 0, PG = 1, during TLB Refresh

During TLB refresh cycles when the PDE and PTE entries are read, the PWT and PCD bits are obtained as shown in Table 3-3 and Table 3-4.

Table 3-3. 32-Bits/4-Kbyte Pages

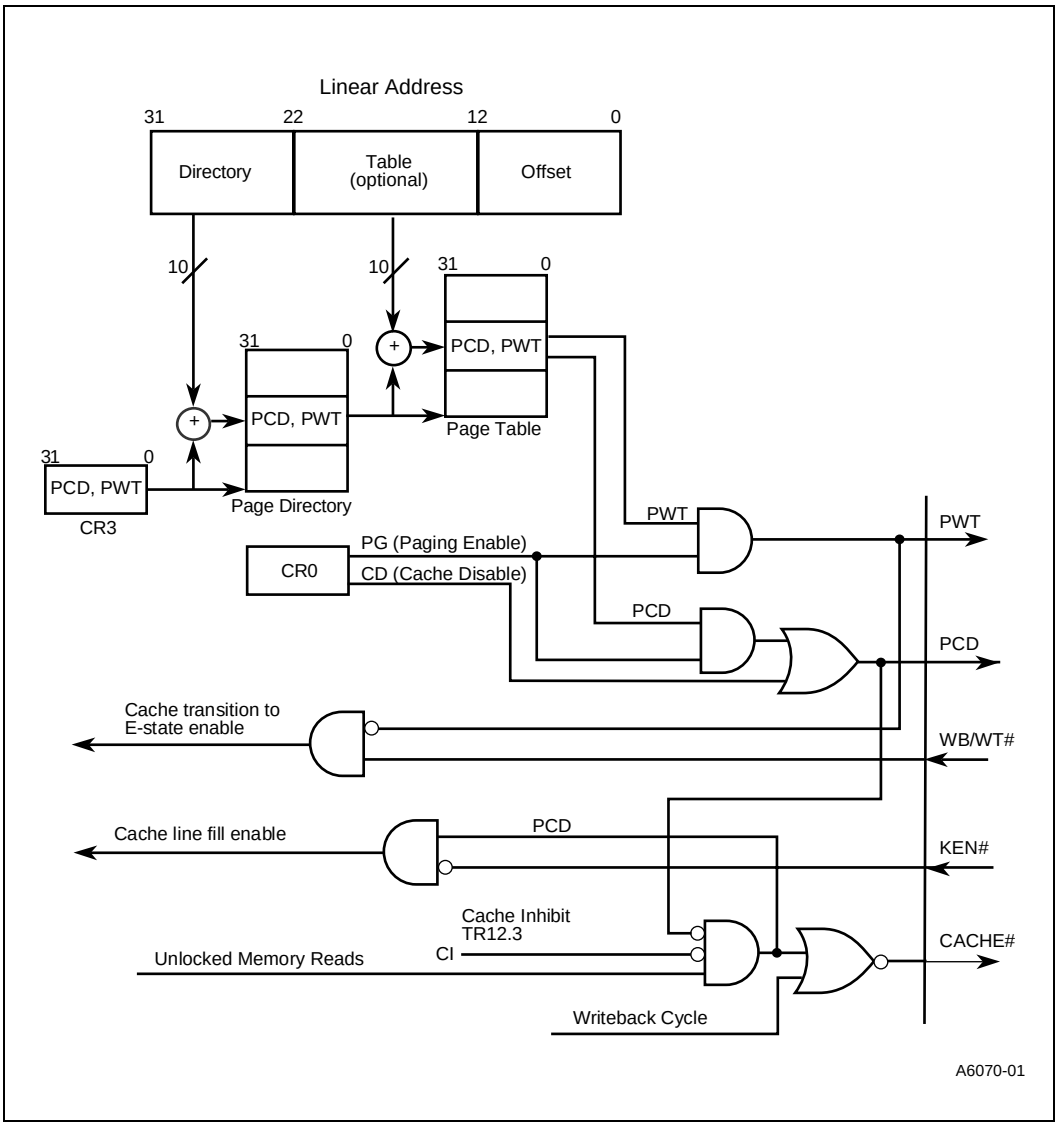
PCD/PWT Taken From	During Accesses To
CR3	PDE
PDE	PTE
PTE	All other paged mem references

Table 3-4. 32-Bits/4-Mbyte Pages

PCD/PWT Taken From	During Accesses To
CR3	PDE
PDE	All other paged mem references

Figure 3-9 shows how PCD and PWT are generated.

Figure 3-9. PCD and PWT Generation



3.5.5 Inquire Cycles

Inquire cycles are initiated by the system to determine if a line is present in the code or data cache, and what state the line is in. This manual refers to inquire cycles and snoop cycles interchangeably.

Inquire cycles are driven to the processor when a bus master other than the processor initiates a read or write bus cycle. Inquire cycles are driven to the processor when the bus master initiates a read to determine if the processor data cache contains the latest information. If the snooped line is in the processor data cache in the modified state, the processor has the most recent information and must schedule a writeback of the data. Inquire cycles are driven to the processor when the other bus master initiates a write to determine if the processor code or data cache contains the snooped line and to invalidate the line if it is present. Inquire cycles are described in detail in Chapter 6, “Bus Functional Description.”

3.5.6 Cache Flushing

The on-chip cache can be flushed by external hardware or by software instructions.

Flushing the cache through hardware is accomplished by driving the FLUSH# pin low. This causes the cache to write back all modified lines in the data cache and mark the state bits for both caches invalid. The Flush Acknowledge special cycle is driven by the processor when all writebacks and invalidations are complete.

The INVD and WBINVD instructions cause the on-chip caches to be invalidated also. WBINVD causes the modified lines in the internal data cache to be written back, and all lines in both caches to be marked invalid. After execution of the WBINVD instruction, the Writeback and Flush special cycles are driven to indicate to any external cache that it should write back and invalidate its contents.

INVD causes all lines in both caches to be invalidated. Modified lines in the data cache are not written back. The Flush special cycle is driven after the INVD instruction is executed to indicate to any external cache that it should invalidate its contents. Care should be taken when using the INVD instruction that cache consistency problems are not created.

Note that the implementation of the INVD and WBINVD instructions are processor dependent. Future processor generations may implement these instructions differently.

3.5.7 Data Cache Consistency Protocol (MESI Protocol)

The embedded Pentium processor Cache Consistency Protocol is a set of rules by which states are assigned to cached entries (lines). The rules apply for memory read/write cycles only. I/O and special cycles are not run through the data cache.

Every line in the data cache is assigned a state dependent on both processor generated activities and activities generated by other bus masters (snooping). The embedded Pentium processor Data Cache Protocol consists of four states that define whether a line is valid (HIT/MISS), if it is available in other caches, and if it has been MODIFIED. The four states are the M (Modified), E (Exclusive), S (Shared) and the I (Invalid) states and the protocol is referred to as the MESI protocol. A definition of the states is given below:

M - Modified: An M-state line is available in only one cache and it is also MODIFIED (different from main memory). An M-state line can be accessed (read/written to) without sending a cycle out on the bus.

E - Exclusive: An E-state line is also available in only one cache in the system, but the line is not MODIFIED (i.e., it is the same as main memory). An E-state line can be accessed (read/written to) without generating a bus cycle. A write to an E-state line causes the line to become MODIFIED.

S - Shared: This state indicates that the line is potentially shared with other caches (i.e., the same line may exist in more than one cache). A read to an S-state line does not generate bus activity, but a write to a SHARED line generates a write-through cycle on the bus. The write-through cycle may invalidate this line in other caches. A write to an S-state line updates the cache.

I - Invalid: This state indicates that the line is not available in the cache. A read to this line will be a MISS and may cause the processor to execute a LINE FILL (fetch the whole line into the cache from main memory). A write to an INVALID line causes the processor to execute a write-through cycle on the bus.

3.5.7.1 State Transition Tables

Lines cached in the processor can change state because of processor-generated activity or as a result of activity on the processor bus generated by other bus masters (snooping). State transitions happen because of processor-generated transactions (memory reads/writes) and by a set of external input signals and internally generated variables. The processor also drives certain pins as a consequence of the Cache Consistency Protocol.

3.5.7.2 Read Cycle

Table 3-5 shows the state transitions for lines in the data cache during unlocked read cycles.

Table 3-5. Data Cache State Transitions for UNLOCKED Processor Initiated Read Cycles†

Present State	Pin Activity	Next State	Description
M	n/a	M	Read hit; data is provided to processor core by cache. No bus cycle is generated.
E	n/a	E	Read hit; data is provided to processor core by cache. No bus cycle is generated.
S	n/a	S	Read hit; data is provided to the processor by the cache. No bus cycle is generated.
I	CACHE# low AND KEN# low AND WB/WT# high AND PWT low	E	Data item does not exist in cache (MISS). A bus cycle (read) will be generated. This state transition will happen if WB/WT# is sampled high with first BRDY# or NA#.
I	CACHE# low AND KEN# low AND (WB/WT# low OR PWT high)	S	Same as previous read miss case except that WB/WT# is sampled low with first BRDY# or NA#.
I	CACHE# high OR KEN# high	I	KEN# pin inactive; the line is not intended to be cached in the embedded Pentium processor.

† Locked accesses to the data cache cause the accessed line to transition to the Invalid state.

Note the transition from I to E or S states (based on WB/WT#) happens only if KEN# is sampled low with the first of BRDY# or NA#, and the cycle is transformed into a LINE FILL cycle. If KEN# is sampled high, the line is not cached and remains in the I state.

3.5.7.3 Write Cycle

The state transitions of data cache lines during processor-generated write cycles are illustrated in Table 3-6. Writes to SHARED lines in the data cache are always sent out on the bus along with updating the cache with the write item. The status of the PWT and WB/WT# pins during these write cycles on the bus determines the state transitions in the data cache during writes to S-state lines.

A write to a SHARED line in the data cache generates a write cycle on the processor bus to update memory and/or invalidate the contents of other caches. If the PWT pin is driven high when the write cycle is run on the bus, the line is be updated and will stay in the S-state regardless of the status of the WB/WT# pin that is sampled with the first BRDY# or NA#. If PWT is driven low, the status of the WB/WT# pin sampled along with the first BRDY# or NA# for the write cycle determines which state (E:S) the line transitions to.

The state transition from S to E is the only transition in which the data and the status bits are not updated at the same time. The data is updated when the write is written to the processor write buffers. The state transition does not occur until the write has completed on the bus (BRDY# has been returned). Writes to the line after the transition to the E-state do not generate bus cycles. However, it is possible that writes to the same line that were buffered or in the pipeline before the transition to the E-state generate bus cycles after the transition to E-state.

An inactive EWBE# input stalls subsequent writes to an E- or an M-state line. All subsequent writes to E- or M-state lines are held off until EWBE# is returned active.

Table 3-6. Data Cache State Transitions for Processor Initiated Write Cycles

Present State	Pin Activity	Next State	Description
M	n/a	M	Write hit; update data cache. No bus cycle generated to update memory.
E	n/a	M	Write hit; update cache only. No bus cycle generated; line is now MODIFIED.
S	PWT low AND WB/WT# high	E	Write hit; data cache updated with write data item. A write-through cycle is generated on bus to update memory and/or invalidate contents of other caches. The state transition occurs after the writethrough cycle completes on the bus (with the last BRDY#).
S	PWT low AND WB/WT# low	S	Same as above case of write to S-state line except that WB/WT# is sampled low.
S	PWT high	S	Same as above cases of writes to S state lines except that this is a write hit to a line in a writethrough page; status of WB/WT# pin is ignored.
I	n/a	I	Write MISS; a writethrough cycle is generated on the bus to update external memory. No allocation done.

NOTE: Memory writes are buffered while I/O writes are not. There is no guarantee of synchronization between completion of memory writes on the bus and instruction execution after the write. A serializing instruction needs to be executed to synchronize writes with the next instruction if necessary.

3.5.7.4 Inquire Cycles (Snooping)

The purpose of inquire cycles is to check whether the address being presented is contained within the caches in the embedded Pentium processor. Inquire cycles may be initiated with or without an INVALIDATION request (INV = 1 or 0). An inquire cycle is run through the data and code caches through a dedicated snoop port to determine if the address is in one of the processor caches. If the address is in a processor cache, the HIT# pin is asserted. If the address hits a modified line in the data cache, the HITM# pin is also asserted and the modified line is then written back onto the bus.

The state transition tables for inquire cycles are given below:

Table 3-7. Cache State Transitions During Inquiry Cycles

Present State	Next State INV=1	Next State INV=0	Description
M	I	S	Snoop hit to a MODIFIED line indicated by HIT# and HITM# pins low. embedded Pentium® processor schedules the writing back of the modified line to memory.
E	I	S	Snoop hit indicated by HIT# pin low; no bus cycle generated.
S	I	S	Snoop hit indicated by HIT# pin low; no bus cycle generated.
I	I	I	Address not in cache; HIT# pin high.

3.5.7.5 Code Cache Consistency Protocol

The processor code cache follows a subset of the MESI protocol. Accesses to the code cache are either a Hit (Shared) or a Miss (Invalid).

In the case of a read hit, the cycle is serviced internally to the processor and no bus activity is generated. In the case of a read miss, the read is sent to the external bus and may be converted to a linefill.

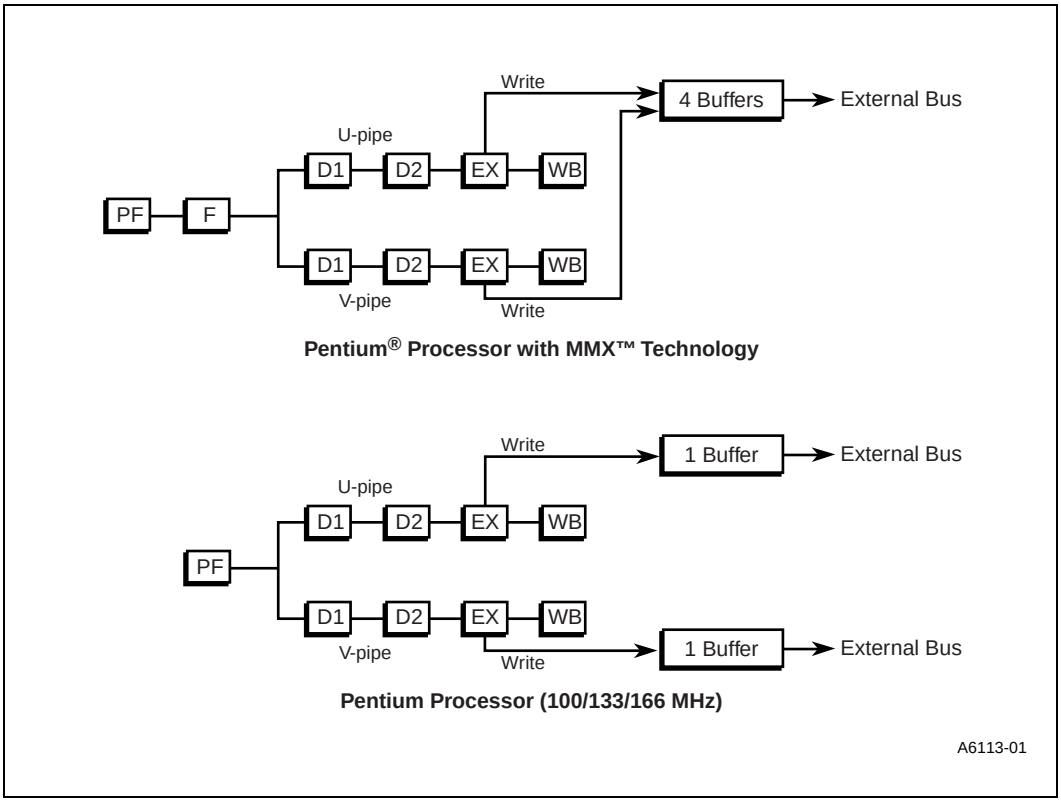
Lines are never overwritten in the code cache. Writes generated by the processor are snooped by the code cache. If the snoop is a hit in the code cache, the line is invalidated. If there is a miss, the code cache is not affected.

3.6 Write Buffers and Memory Ordering

The embedded Pentium processor has two write buffers, one corresponding to each of the pipelines, to enhance the performance of consecutive writes to memory. These write buffers are one quadword wide (64-bits) and can be filled simultaneously in one clock e.g., by two simultaneous write misses in the two instruction pipelines. Writes in these buffers are driven out on the external bus in the order they were generated by the processor core. No reads (as a result of cache miss) are reordered around previously generated writes sitting in the write buffers. The implication of this is that the write buffers will be flushed or emptied before a subsequent bus cycle is run on the external bus (unless BOFF# is asserted and a writeback cycle becomes pending, see “Linefill and Writeback Buffers” on page 3-29).

The embedded Pentium processor with MMX technology has four write buffers that can be used by either the u-pipe or v-pipe. Posting writes to these buffers enables the pipe to continue advancing when consecutive writes to memory occur. The writes will be executed on the bus as soon as it is free, in FIFO order. Reads cannot bypass writes posted in these buffers.

Figure 3-10. Embedded Pentium® Processor Write Buffer Implementation



The embedded Pentium processor supports strong write ordering only. That is, writes generated by the embedded Pentium processor are driven to the bus or updated in the cache in the order in which they occur. The embedded Pentium processor does not write to E or M-state lines in the data cache if there is a write in either write buffer, if a write cycle is running on the bus, or if EWBE# is inactive.

Note that only memory writes are buffered and I/O writes are not. There is no guarantee of synchronization between completion of memory writes on the bus and instruction execution after the write. The OUT instruction or a serializing instruction needs to be executed to synchronize writes with the next instruction. Refer to “Serializing Operations” on page 3-28 for more information.

No re-ordering of read cycles occurs on the embedded Pentium processor. Specifically, the write buffers are flushed before the IN instruction is executed.

3.6.1 External Event Synchronization

When the system changes the value of NMI, INTR, FLUSH#, SMI# or INIT as the result of executing an OUT instruction, these inputs must be at a valid state three clocks before BRDY# is returned to ensure that the new value will be recognized before the next instruction is executed.

Note that if an OUT instruction is used to modify A20M#, this will not affect previously prefetched instructions. A serializing instruction must be executed to guarantee recognition of A20M# before a specific instruction.

3.6.2 Serializing Operations

After executing certain instructions, the embedded Pentium processor serializes instruction execution. This means that any modifications to flags, registers, and memory for previous instructions are completed before the next instruction is fetched and executed. The prefetch queue is flushed as a result of serializing operations.

The embedded Pentium processor serializes instruction execution after executing one of the following instructions: MOV to Debug Register, MOV to Control Register, INVD, INVLPG, IRET, IRETD, LGDT, LLDT, LIDT, LTR, WBINVD, CPUID, RSM and WRMSR.

The CPUID instruction can be executed at any privilege level to serialize instruction execution.

When the processor serializes instruction execution, it ensures that it has completed any modifications to memory, including flushing any internally buffered stores; it then waits for the EWBE# pin to go active before fetching and executing the next instruction. Systems may use the EWBE# pin to indicate that a store is pending externally. In this manner, a system designer may ensure that all externally pending stores complete before the processor begins to fetch and execute the next instruction.

The processor does not generally writeback the contents of modified data in its data cache to external memory when it serializes instruction execution. Software can force modified data to be written back by executing the WBINVD instruction.

Whenever an instruction is executed to enable/disable paging (that is, change the PG bit of CR0), this instruction must be followed with a jump. The instruction at the target of the branch is fetched with the new value of PG (i.e., paging enabled/disabled); however, the jump instruction itself is fetched with the previous value of PG. Intel386™, Intel486 and embedded Pentium processors have slightly different requirements to enable and disable paging. In all other respects, an MOV to CR0 that changes PG is serializing. Any MOV to CR0 that does not change PG is completely serializing.

Whenever an instruction is executed to change the contents of CR3 while paging is enabled, the next instruction is fetched using the translation tables that correspond to the new value of CR3. Therefore the next instruction and the sequentially following instructions should have a mapping based upon the new value of CR3.

The embedded Pentium processor implements branch-prediction techniques to improve performance by prefetching the destination of a branch instruction before the branch instruction is executed. Consequently, instruction execution is not generally serialized when a branch instruction is executed.

Although the I/O instructions are not “serializing” because the processor does not wait for these instructions to complete before it prefetches the next instruction, they do have the following properties that cause them to function in a manner that is identical to previous generations. I/O reads are not re-ordered within the processor; they wait for all internally pending stores to complete. Note that the embedded Pentium processor does not sample the EWBE# pin during reads. If necessary, external hardware must ensure that externally pending stores are complete before returning BRDY#. This is the same requirement that exists on Intel386 and Intel486 processor systems. The OUT and OUTS instructions are also not “serializing,” as they do not stop the prefetcher. They do, however, ensure that all internally buffered stores have completed, that EWBE# has been sampled active indicating that all externally pending stores have completed and that the I/O write has completed before they begin to execute the next instruction. Note that unlike the Intel486 processor, it is not necessary for external hardware to ensure that externally pending stores are complete before returning BRDY#.

On the embedded Pentium processor with MMX technology, serializing instructions require an additional clock to complete compared to the embedded Pentium processor due to the additional pipeline stage.

3.6.3 Linefill and Writeback Buffers

In addition to the write buffers corresponding to each of the internal pipelines, the embedded Pentium processor has three writeback buffers. Each of the writeback buffers are 1 deep and 32-bytes (1 line) wide.

A dedicated replacement writeback buffer stores writebacks caused by linefills that replace modified lines in the data cache. In addition, an external snoop writeback buffer stores writebacks caused by a inquire cycles that hit modified lines in the data cache. Finally, an internal snoop writeback buffer stores writebacks caused by internal snoop cycles that hit modified lines in the data cache. Internal and external snoops are discussed in detail in “Cache Consistency Cycles (Inquire Cycles)” on page 6-33. Write cycles are driven to the bus with the following priority:

1. Contents of external snoop writeback buffer
2. Contents of internal snoop writeback buffer
3. Contents of replacement writeback buffer
4. Contents of write buffers.

Note that the contents of the write buffer that was written into first are driven to the bus first. If both write buffers were written to in the same clock, the contents of the u-pipe buffer is written out first. In the embedded Pentium processor with MMX technology, the write buffers are written in order, even though there is no u-pipe buffer and no v-pipe buffer.

The embedded Pentium processor implements two linefill buffers, one for the data cache and one for the code cache. As information (data or code) is returned to the processor for a cache linefill, it is written into the linefill buffer. After the entire line has been returned to the processor it is transferred to the cache. Note that the processor requests the needed information first and uses that information as soon as it is returned. The processor does not wait for the linefill to complete before using the requested information.

If a line fill causes a modified line in the data cache to be replaced, the replaced line remains in the cache until the linefill is complete. After the linefill is complete, the line being replaced is moved into the replacement writeback buffer and the new linefill is moved into the cache.

3.7 External Interrupt Considerations

The embedded Pentium processor recognizes the following external interrupts: BUSCHK#, R/S#, FLUSH#, SMI#, INIT, NMI, INTR and STPCLK#. These interrupts are recognized at instruction boundaries. The instruction boundary is the first clock in the execution stage of the instruction pipeline. This means that before an instruction is executed, the processor checks to see if any interrupts are pending. If an interrupt is pending, the processor flushes the instruction pipeline and then services the interrupt.

The embedded Pentium processor interrupt priority scheme is shown in Table 3-8.

Table 3-8. Embedded Pentium® Processor Interrupt Priority Scheme

Priority Level	ITR = 0 (default)	ITR = 1
1	Breakpoint (INT 3)	Breakpoint (INT 3)
2	BUSCHK#	BUSCHK#
3	Debug Traps (INT 1)	FLUSH#
4	R/S#	SMI#
5	FLUSH#	Debug Traps (INT 1)
6	SMI#	R/S#
7	INIT	INIT
8	NMI	NMI
9	INTR	INTR
10	Floating-Point Error	Floating-Point Error
11	STPCLK#	STPCLK#
12	Faults on Next Instruction	Faults on Next Instruction

NOTE: ITR is bit 9 of the TR12 register.

3.8 Introduction to Dual Processor Mode

Symmetric dual processing in a system is supported with two embedded Pentium processors sharing a single second-level cache. The processors must be of the same type, either two embedded Pentium processors or two embedded Pentium processor with MMX technology. The two processors appear to the system as a single processor. Multiprocessor operating systems properly schedule computing tasks between the two processors. This scheduling of tasks is transparent to software applications and the end-user. Logic built into the processors support a “glueless” interface for easy system design. Through a private bus, the two processors arbitrate for the external bus and maintain cache coherency.

In this manual, in order to distinguish between two processors in dual processing mode, one processor is designated as the Primary processor and the other as the Dual processor. Note that this is a different concept than that of “master” and “checker” processors.

The Dual processor is a configuration option of the embedded Pentium processor. The Dual processor must operate at the same bus and core frequency and bus/core ratio as the Primary processor.

The Primary and Dual processors include logic to maintain cache consistency between the processors and to arbitrate for the common bus. The cache consistency and bus arbitration activity causes the dual processor pair to issue extra bus cycles that does not appear in a embedded Pentium processor uniprocessor system.

Chapter 4, “Microprocessor Initialization and Configuration,” describes in detail how the DP bootup, cache consistency, and bus arbitration mechanisms operate. In order to operate properly in dual processing mode, the Primary and Dual processors require a private APIC, cache consistency, and bus arbitration interfaces, as well as a multiprocessing-ready operating system.

The dual processor interface allows the Dual processor to be added for a substantial increase in system performance. The interface allows the Primary and Dual processor to operate in a coherent manner that is transparent to the system.

The memory subsystem transparency was the primary goal of the cache coherency and bus arbitration mechanisms.

3.8.1 Dual Processing Terminology

This section defines some terms used in the following discussions.

Symmetric Multi-Processing:	Two or more processors operating with equal priorities in a system. No individual processor is a master, and none is a slave.
DP or Dual Processing:	The Primary and Dual processor operating symmetrically in a system sharing a second-level cache.
MRM or Most Recent Master:	The processor (either the Primary or Dual) that currently owns the processor address bus. When interprocessor pipelining, this is the processor which last issued an ADS#.
LRM or Least Recent Master:	The processor (either the Primary or Dual) that does not own the address bus. The LRM automatically snoops every ADS# from the MRM processor in order to maintain level-one cache coherency.
Primary Processor:	The embedded Pentium processor when CPUTYP = V_{SS} (or left floating).
Dual Processor:	The embedded Pentium processor when CPUTYP = V_{CC}.

3.8.2 Dual Processing Overview

The Primary and Dual processor both have logic built-in to support “glueless” dual-processing behind a shared L2 cache. Through a set of private handshake signals, the Primary and Dual processors arbitrate for the external bus and maintain cache coherency between themselves. The bus arbitration and cache coherency mechanisms allow the Primary and Dual processors to look like a single embedded Pentium processor to the external bus.

The Primary and Dual processors implement a fair arbitration scheme. If the Least Recent Master (LRM) requests the bus from the Most Recent Master (MRM), the bus is granted. The embedded Pentium processor arbitration scheme provides no penalty to switch from one master to the next. If pipelining is used, the two processors pipeline into and out of each other’s cycles according to the embedded Pentium processor specification.

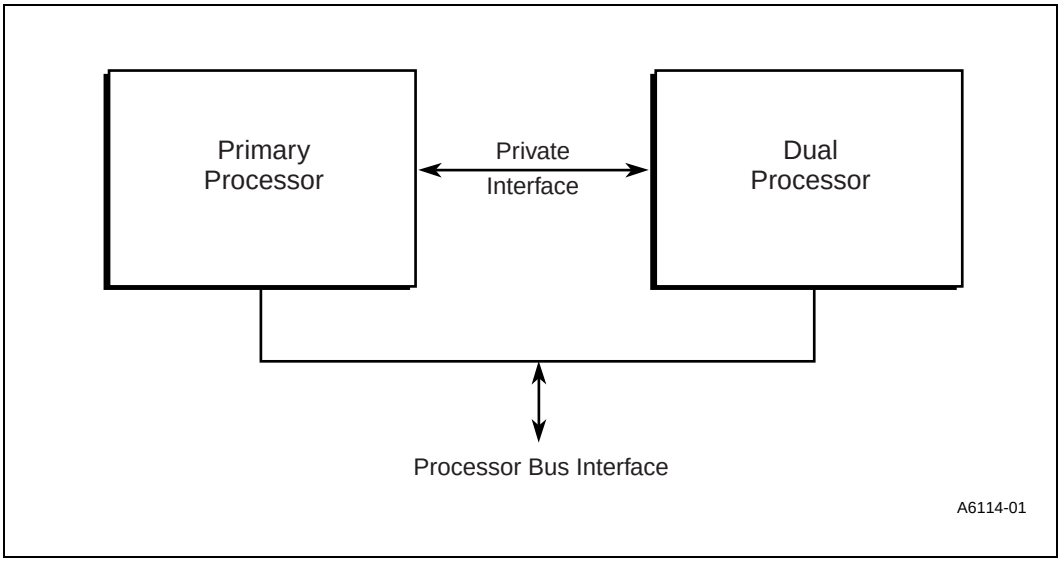
Cache coherency is maintained between the two processors by snooping on every bus access. The LRM must snoop with every ADS# assertion of the MRM. Internal cache states are maintained accordingly. If an access hits a modified line, a writeback is scheduled as the next cycle, in accordance with the embedded Pentium processor specification.

Using the Dual processor may require special design considerations. Refer to Chapter 5, “Hardware Interface” for more details.

3.8.2.1 Conceptual Overview

Figure 3-11 is a block diagram of a two processor system.

Figure 3-11. Dual Processors



The dual processor pair appears to the system bus as a single, unified processor. The operation is identical to a uni-processor embedded Pentium processor, except as noted in "Summary of Dual Processing Bus Cycles" on page 6-43. The interface shields the system designer from the cache consistency and arbitration mechanisms that are necessary for dual processor operation.

Both the Primary and Dual processors contain local APIC modules. The system designer is recommended to supply an I/O APIC or other multiprocessing interrupt controller in the chip set that interfaces to the local APIC blocks over a three-wire bus. The APIC allows directed interrupts as well as inter-processor interrupts.

The Primary and Dual processors, when operating in dual processing mode, require the local APIC modules to be hardware enabled in order to complete the bootup handshake protocol. This method is used to "wake up" the Dual processor at an address other than the normal Intel architecture high memory execution address. On bootup, if the Primary processor detects that a Dual processor is present, the dual processor cache consistency and arbitration mechanisms are automatically enabled. The bootup handshake process is supported in a protocol that is included in the embedded Pentium processor. See Chapter 4, "Microprocessor Initialization and Configuration," for more details on the APIC.

3.8.2.2 Arbitration Overview

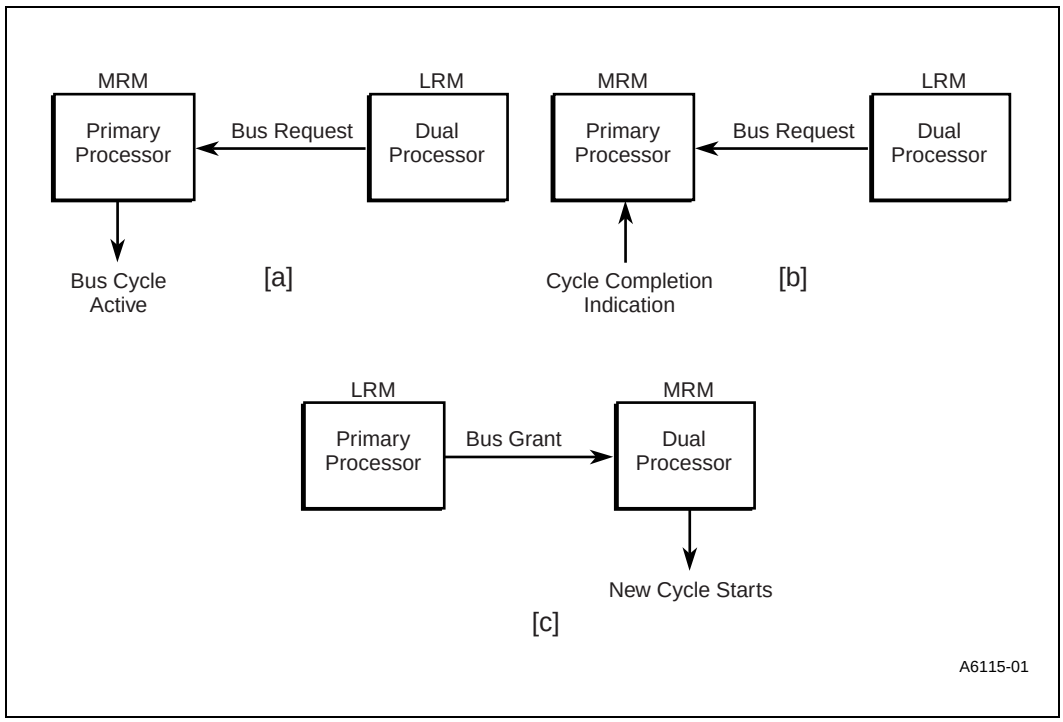
In the dual processor configuration, a single-system bus provides the processors access to the external system. This bus is a single, shared resource.

The dual processor pair must arbitrate for use of the system bus as requests are generated. The processors implement a fair arbitration mechanism.

When the LRM processor needs to run a cycle on the bus it submits a request for bus ownership to the MRM. The MRM processor grants the LRM processor bus ownership as soon as all outstanding bus requests have finished on the processor bus. The LRM processor assumes the MRM state, and the processor that was just the MRM, becomes the LRM. Figure 3-12 further illustrates this point:

Diagram (a) of Figure 3-12 shows a configuration where the Primary processor is in the MRM state and the Dual processor is in the LRM state. The Primary processor is running a cycle on the system bus when it receives a bus request from the Dual processor. In diagram (b) of Figure 3-12 the MRM (still the Primary processor) has received an indication that the bus request has finished. The bus ownership has transferred in diagram (c) of Figure 3-12, where the Dual processor is now the MRM. At this point, the Dual processor starts a bus transaction and continues to own the bus until the LRM requests the bus.

Figure 3-12. Dual Processor Arbitration Mechanism



3.8.2.3 Cache Coherency Overview

The Primary and Dual processors both contain separate code and data caches. The data cache uses the MESI protocol to enforce cache consistency. A line in the data cache can be in the Modified, Exclusive, Shared or Invalid state, whereas a line in the instruction cache can be either in the valid or invalid state.

A situation can arise where the Primary and Dual processors are operating in dual processor mode with shared code or data. The first-level caches attempt to cache this code and data whenever possible (as indicated by the page cacheability bits and the cacheability pins). The private cache coherency mechanism guarantees data consistency across the processors. If any data is cached in one of the processors, and the other processor attempts to access the data, the processor containing

the data notifies the requesting processor that it has cached the data. The state of the cache line in the processor containing the data changes depending on the current state and the type of request the other processor has made.

In some cases, the data returned by the system is ignored. This constraint is placed on the dual processor cache consistency mechanism so that the dual processor pair looks like a single processor to the system bus. However, in general, bus accesses are minimized to efficiently use the available bus bandwidth.

The basic coherency mechanism requires the processor that is in the LRM state to snoop all MRM bus activity. The MRM processor running a bus cycle watches the LRM processor for an indication that the data is contained in the LRM cache. The following diagrams illustrate the basic coherency mechanism. These figures show an example in which the Primary processor (the MRM) is performing a cache line fill of data. The data requested by the Primary processor is cached by the Dual processor (the LRM), and is in the modified state.

In diagram (a) of Figure 3-13, the Primary processor has already negotiated with the Dual processor for use of the system bus and has started a cycle. As the Primary processor starts running the cycle on the system bus, the Dual processor snoops the transaction. The key for the start of the snoop sequence for the LRM processor is an assertion of ADS# by the MRM processor.

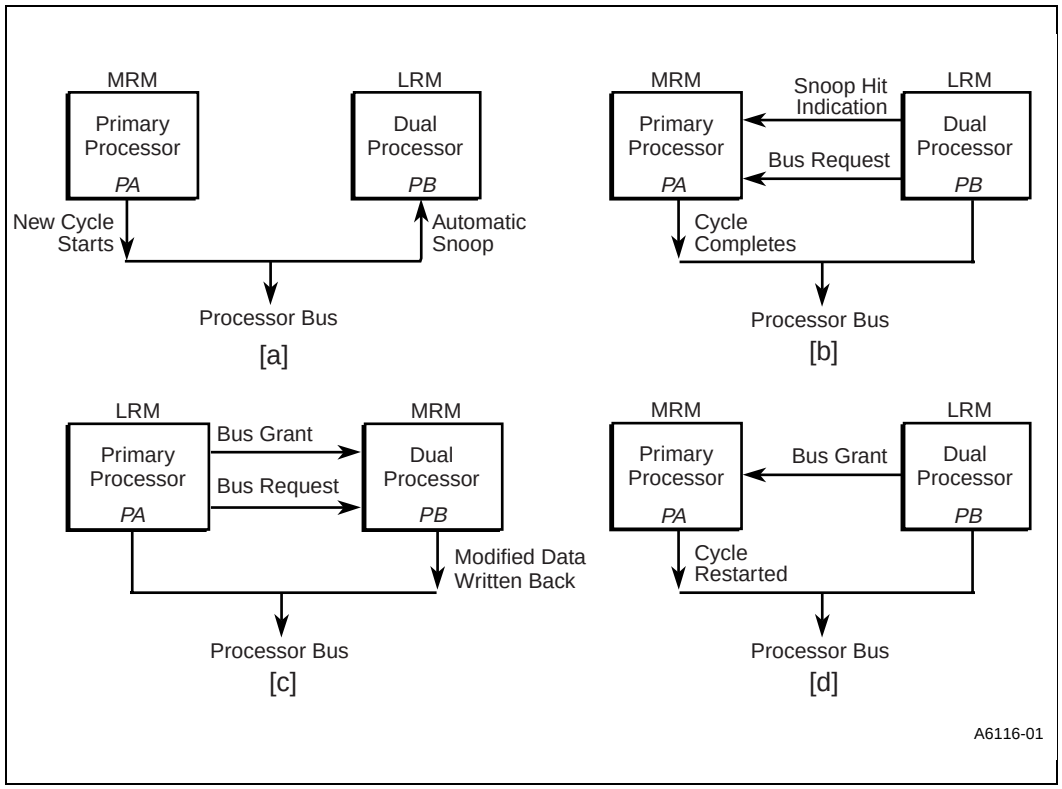
Diagram (b) of Figure 3-13 shows the Dual processor indicating to the Primary processor that the requested data is cached and modified in the Dual processor cache. The snoop notification mechanism uses a dedicated, two-signal interface that is private to the dual processor pair. At the same time that the Dual processor indicates that the transaction is contained as Modified in the its cache, the Dual processor requests the bus from the Primary processor (still the MRM). The MRM processor continues with the transaction that is outstanding on the bus, but ignores the data returned by the system bus.

After the Dual processor notifies the Primary processor that the requested data is modified in the Dual processor cache, the Dual processor waits for the bus transaction to complete. At this point, the LRM/MRM state will toggle, with the Primary processor becoming the LRM processor and the Dual processor becoming the MRM processor. This sequence of events is shown in diagram (c) of Figure 3-13.

Diagram (c) of Figure 3-13 also shows the Dual processor writing the data back on the system bus. The write back cycle looks like a normal cache line replacement to the system bus. The final state of the line in the Dual processor is determined by the value of the W/R# pin as sampled during the ADS# assertion by the Primary processor.

Finally, diagram (d) of Figure 3-13 shows the Primary processor re-running the bus transaction that started the entire sequence. The requested data is returned by the system as a normal line fill request without intervention from the LRM processor.

Figure 3-13. Dual Processor L1 Cache Consistency



3.9 APIC Interrupt Controller

The embedded Pentium processor contains implementations of the Advanced Programmable Interrupt Controller architecture. These implementations are capable of supporting a multiprocessing interrupt scheme with an external APIC-compatible controller.

The Advanced Programmable Interrupt Controller (APIC) is an on-chip interrupt controller that supports multiprocessing. In a uniprocessor system, the APIC may be used as the sole system interrupt controller, or may be disabled and bypassed completely.

In a multiprocessor system, the APIC operates with an additional and external I/O APIC system interrupt controller. The dual-processor configuration requires that the APIC be hardware enabled. The APICs of the Primary and Dual processors are used in the bootup procedure to communicate start-up information.

Note: The APIC is not hardware compatible with the 82489DX.

On the embedded Pentium processor, the APIC uses 3 pins: PICCLK, PICD0, and PICD1. PICCLK is the APIC bus clock while PICD0-PICD1 form the two-wire communication bus.

To use the 8259A interrupt controller, or to completely bypass it, the APIC may be disabled using the APICEN pin. You must use the local APICs when using the dual-processor component.

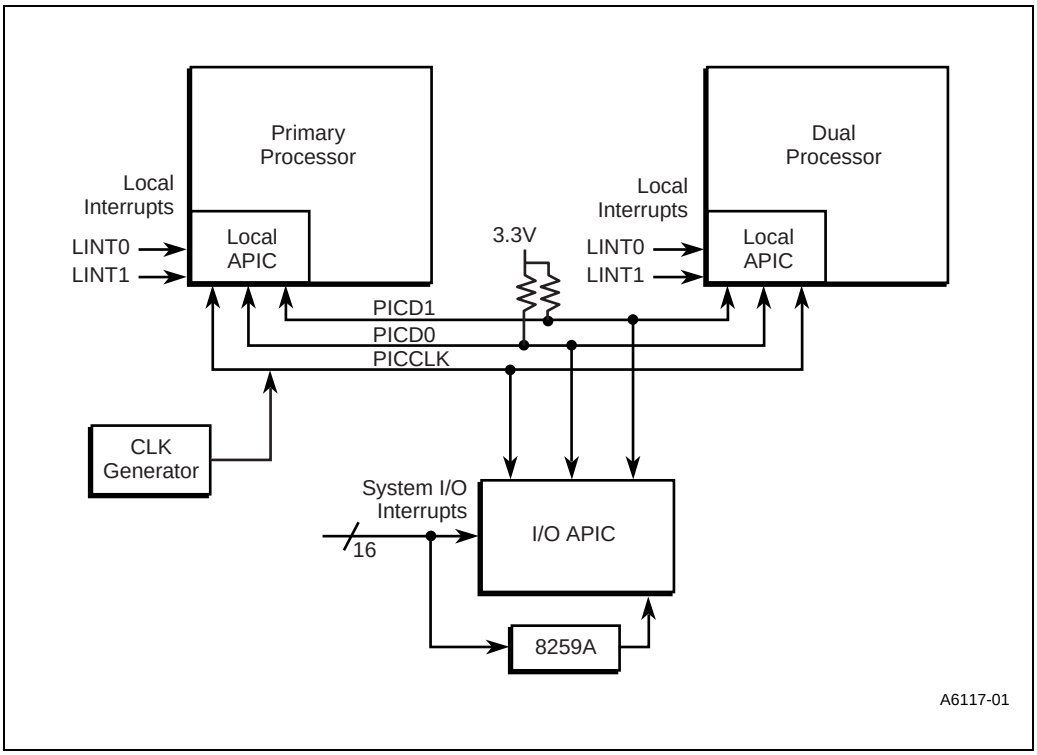
The main features of the APIC architecture include:

- Multiprocessor interrupt management (static and dynamic symmetric interrupt distribution across all processors)
- Dynamic interrupt distribution that includes routing interrupts to the lowest-priority processor
- Inter-processor interrupt support
- Edge or level triggered interrupt programmability
- Various naming/addressing schemes
- System-wide processor control functions related to NMI, INIT, and SMI (see Chapter 12 for APIC handling of SMI)
- 8259A compatibility by becoming virtually transparent with regard to an externally connected 8259A style controller, making the 8259A visible to software
- A 32-bit wide counter used as a timer to generate time slice interrupts local to that processor.

The AC timings of the embedded Pentium processor APIC are described in Chapter 7. Note that although there are minor software differences from the 82489DX, programming to the integrated APIC model ensures compatibility with the external 82489DX. For additional APIC programming information, refer to the *MultiProcessor Specification* (order number 242016).

In a dual-processor configuration, the local APIC may be used with an additional device similar to the I/O APIC. The I/O APIC is a device that captures all system interrupts and directs them to the appropriate processors via various programmable distribution schemes. An external device provides the APIC system clock. Interrupts that are local to each processor go through the APIC on each chip. A system example is shown in Figure 3-14.

Figure 3-14. APIC System Configuration

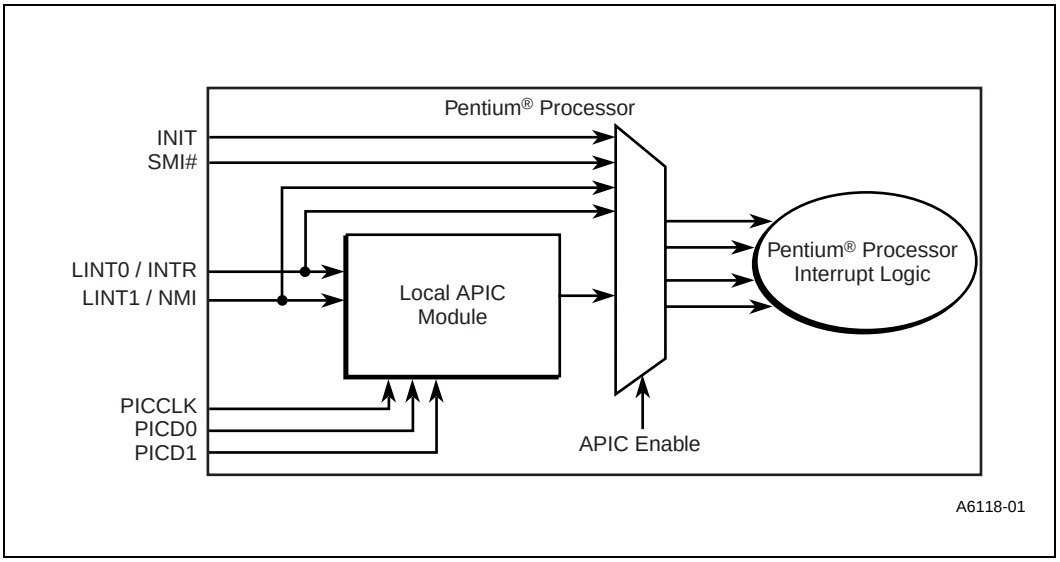


The APIC devices in the Primary and Dual processors may receive interrupts from the I/O APIC via the three-wire APIC bus, locally via the local interrupt pins (LINT0, LINT1), or from the other processor via the APIC bus. The local interrupt pins, LINT0 and LINT1, are shared with the INTR and NMI pins, respectively. When the APIC is bypassed (hardware disabled) or programmed in "through local" mode, the 8259A interrupt (INTR) and NMI are connected to the INTR/LINT0 and NMI/LINT1 pins of the processor. Figure 3-15 shows the APIC implementation in the embedded Pentium processor. Note that the PICCLK has a maximum frequency of 16.67 MHz.

When the local APIC is hardware enabled, *data* memory accesses to its 4 Kbyte address space are executed internally and do not generate an ADS# on the processor bus. However, a *code* memory access in the 4 KByte APIC address space will not be recognized by the APIC and will generate a cycle on the processor bus.

Note: Internally executed data memory accesses may cause the address bus to toggle even though no ADS# is issued on the processor bus.

Figure 3-15. Local APIC Interface



3.9.1 APIC Configuration Modes

There are four possible APIC Modes:

- Normal mode
- Bypass mode (hardware disable)
- Through local mode
- Masked mode (software disable)

3.9.1.1 Normal Mode

This is the normal operating mode of the local APIC. When in this mode, the local APIC is both hardware and software enabled.

3.9.1.2 Bypass Mode

Bypass mode effectively removes (bypasses) the APIC from the embedded Pentium processor, causing it to operate as if there were no APIC present. Any accesses to the APIC address space go to memory. APICEN is sampled at the falling edge of RESET, and later becomes the PICD1 (part of the APIC 3-wire bus) signal. Bypass mode is entered by driving APICEN low at the falling edge of RESET. Since the APIC must be used to enable the Dual processor after RESET, PICD1 must be driven high at reset to ensure that APIC is hardware enabled if a second processor is present.

For hardware disabling operations, the following implications must be considered:

- The INTR and NMI pins become functionally equivalent to the corresponding interrupt pins in the embedded Pentium processor, and the APIC is bypassed.
- The APIC PICCLK must be tied high.
- The system will not operate with the Dual Processor if the local APIC is hardware disabled.

3.9.1.3 Through Local Mode

Configuring in Through Local Mode allows the APICs to be used for the dual-processor bootup handshake protocol and then pass interrupts through the local APIC to the core to support an external interrupt controller.

To use the Through Local Mode of the local APIC, the APIC must be enabled in both hardware and software. This is done by programming two local vector table entries, LVT1 and LVT2, at addresses 0FEE00350H and 0FEE00360H, as external interrupts (ExtInt) and NMI, respectively. The 8259A responds to the INTA cycles and returns the interrupt vector to the processor.

The local APIC should not be sent any interrupts prior to it’s being programmed. Once the APIC is programmed it can receive interrupts.

Note that although external interrupts and NMI are passed through the local APIC to the core, the APIC can still receive messages on the APIC bus.

3.9.1.4 Masked Mode

The local APIC is initialized to masked mode once hardware enabled via the APICEN pin. In order to be programmed in normal or Through Local Modes, the APIC must be “software enabled.” Once operating in normal or Through Local Modes, the APIC may be disabled by software by clearing bit 8 of the APIC’s spurious vector interrupt register (Note: this register is normally cleared at RESET and INIT). This register is at address 0FEE000F0H. Disabling APIC in software returns it to Masked mode. With the exception of NMI, SMI, INIT, remote reads, and the startup IPI, all interrupts are masked on the APIC bus. The local APIC does not accept interrupts on LINT0 or LINT1.

3.9.1.5 Software Disabling Implications

For the software disabling operations, the following implications must be considered:

- The 4-Kbyte address space for the APIC is always blocked for data accesses (i.e., external memory in this region must not be accessed).
- The interrupt control register (ICR) can be read and written (e.g., interprocessor interrupts are sent by writing to this register).

- The APIC can continue to receive SMI, NMI, INIT, “startup,” and remote read messages.
- Local interrupts are masked.
- Software can enable/disable the APIC at any time. After software disabling the local APICs, pending interrupts must be handled or masked by software.
- The APIC PICCLK must be driven at all times.

3.9.1.6 Dual Processing with the Local APIC

The Dual processor bootup protocol may be used in the normal, through local, or masked modes.

3.9.2 Loading the APIC ID

Loading the APIC ID may be done with external logic that would drive the proper address at reset. If the BE3#–BE0# signals are not driven and do not have external resistors to V_{CC} or V_{SS}, the APIC ID value defaults to 0000 for the Primary processor and 0001 for the Dual processor.

Table 3-9. APIC ID

APIC ID Register Bit	Pin Latched at RESET
bit 24	BE0#
bit 25	BE1#
bit 26	BE2#
bit 27	BE3#

Warning: An APIC ID of all 1s is an APIC special case (i.e., a broadcast) and must not be used. Since the Dual processor inverts the lowest order bit of the APIC ID placed on the BE pins, the value “1110” should also be avoided when operating in Dual Processing mode.

In a dual processor configuration, the OEM and Socket 5 should have the four BE pairs tied together. The OEM processor loads the value seen on these four pins at RESET. The dual processor loads the value seen on these pins and automatically inverts bit 24 of the APIC ID Register. Thus, the two processors have unique APIC ID values.

In a general multi-processing system consisting of multiple embedded Pentium processors, these pins must not be tied together, so each local APIC can have unique ID values.

These four pins must be valid and stable two clocks before and after the falling edge of RESET.

3.9.3 Response to HOLD

While the embedded Pentium processor is accessing the APIC, the processor will respond to a HOLD request with a maximum delay of six clocks. To external agents that are not aware of the APIC bus, this looks like the processor is not responding to HOLD even though ADS# has not been driven and the processor bus seems idle.

3.10 Fractional Speed Bus

The embedded Pentium processor is offered in various bus-to-core frequency ratios. The BF2–BF0 configuration pins determine the bus-to-core frequency ratio. The processor multiplies the input CLK by the bus-to-core ratio to achieve higher internal core frequencies.

Note: Only the Low-power Embedded Pentium Processor with MMX technology has a BF2 pin.

The external bus frequency is set on power-up RESET through the CLK pin. The processor samples the BF*n* pins on the falling edge of RESET to determine which bus-to-core ratio to use. When the BF*n* pins are left unconnected, the embedded Pentium processor defaults to the 2/3 ratio and the embedded Pentium processor with MMX technology defaults to the 1/2 ratio. BF*n* settings must not change its value while RESET is active. Changing the external bus speed or bus-to-core ratio requires a "power-on" RESET pulse initialization. Once a frequency is selected, it may not be changed with a warm-reset (15 clocks). The BF pin must meet a 1 ms setup time to the falling edge of RESET.

Each embedded Pentium processor is specified to operate within a single bus-to-core ratio and a specific minimum to maximum bus frequency range (corresponding to a minimum to maximum core frequency range).

Caution: Operation in other bus-to-core ratios or outside the specified operating frequency range is not supported.

Tables 3-10 through 3-12 summarize these specifications.

Table 3-10. Bus-to-Core Frequency Ratios for the Embedded Pentium® Processor (at 100/133/166 MHz)

BF1	BF0	Embedded Pentium® Processor Bus/Core Ratio	Max Bus/Core Frequency (MHz)	Min Bus/Core Frequency (MHz)
0	0	2/5	66/166	33/83
1	0	1/2	66/133	33/66
1	1	2/3†	66/100	33/50

† This is the default bus fraction for the embedded Pentium processor (at 100/133/166 MHz). If the BF pins are left floating, the processor will be configured for the 2/3 bus to core frequency ratio.

Table 3-11. Bus-to-Core Frequency Ratios for the Embedded Pentium® Processor with MMX™ Technology

BF1	BF0	Embedded Pentium Processor with MMX™ Technology Bus/Core Ratio	Max Bus/Core Frequency (MHz)	Min Bus/Core Frequency (MHz)
1	1	2/7	66/233	33/117
0	1	1/3	66/200	33/100
1	0	1/2†	N/A	N/A

† This is the default bus-to-core ratio for the Pentium processor with MMX technology. If the BF pins are left floating, the processor will be configured for the 1/2 bus-to-core frequency ratio, which is unsupported. *Do not float the BF pins at RESET.*

Table 3-12. Bus-to-Core Frequency Ratios for the Low-Power Embedded Pentium® Processor with MMX™ Technology

BF2	BF1	BF0	Low-Power Embedded Pentium Processor with MMX™ Technology Bus/Core Ratio[4]	Max Bus/Core Frequency (MHz)	Min Bus/Core Frequency (MHz)
0	0	0	2/5	66/166	
1	0	0	1/4	66/266	

3.10.1 Fractional Bus Operation Examples

The following examples illustrate the embedded Pentium processor synchronization mechanism.

Figure 3-16. Processor 1/2 Bus Internal/External Data Movement

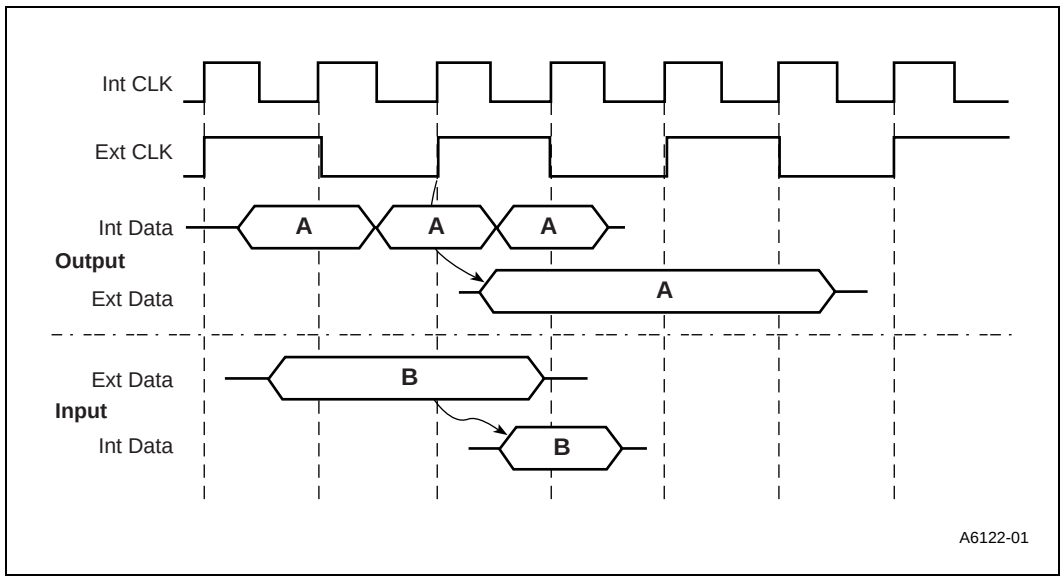


Figure 3-17. Processor 2/3 Bus Internal/External Data Movement

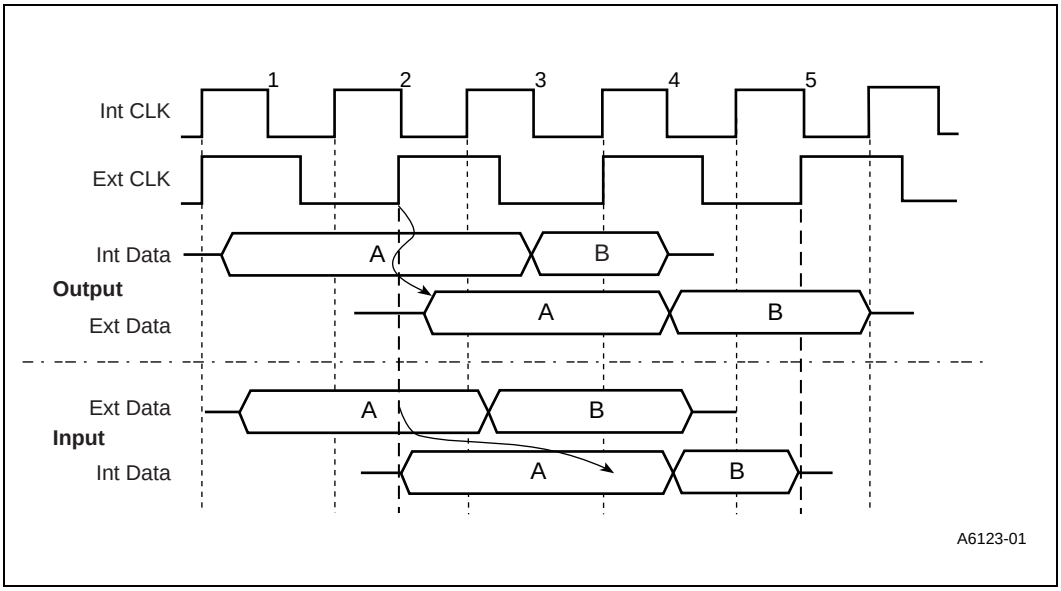


Figure 3-18 shows how the embedded Pentium processor prevents data from changing in clock 2, where the 2/3 external clock rising edge occurs in the middle of the internal clock phase, so it can be properly synchronized and driven.

Figure 3-18. Processor 2/5 Bus Internal/External Data Movement

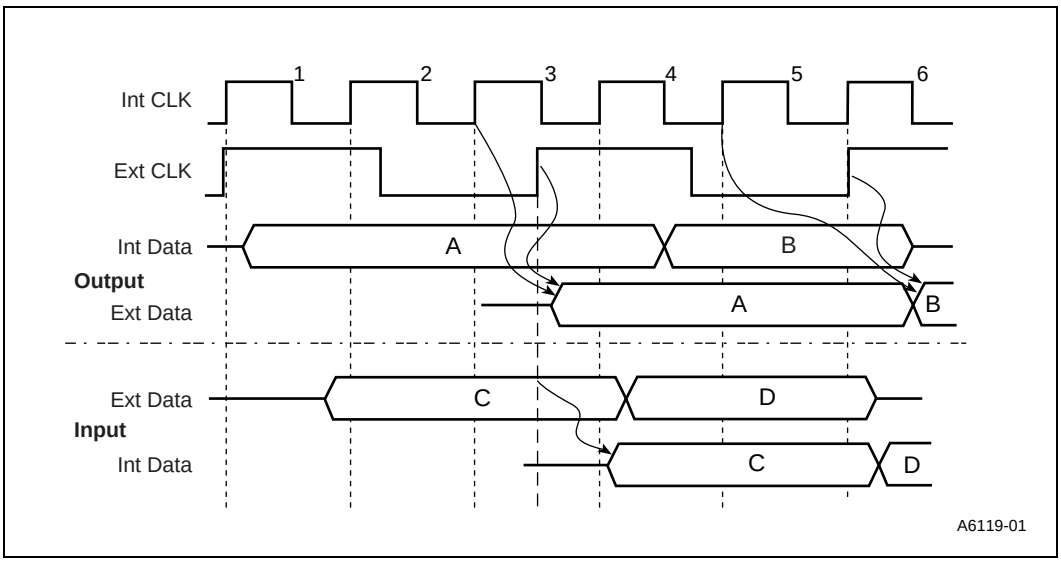
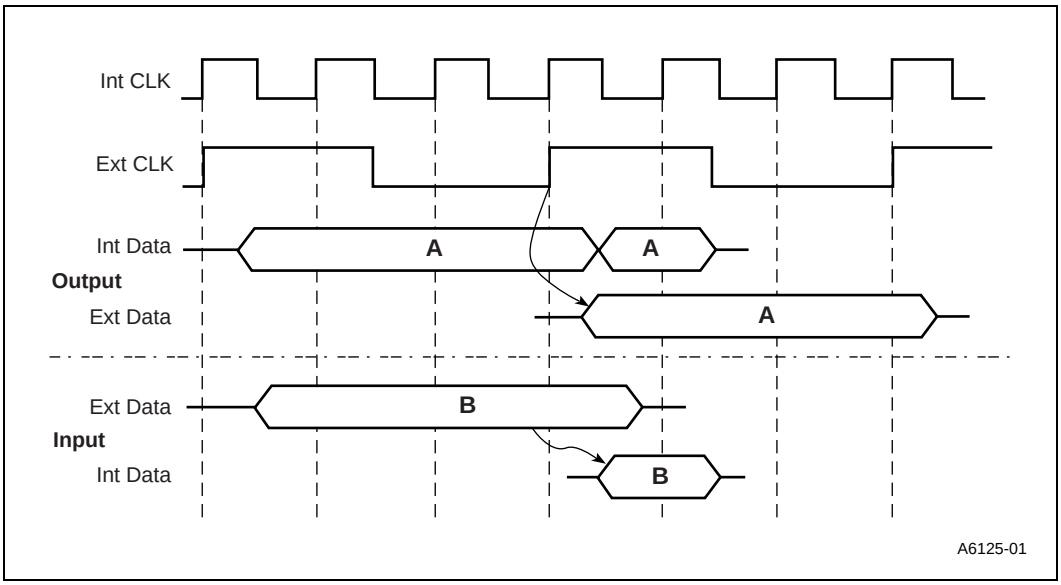


Figure 3-19. Processor 1/3 Bus Internal/External Data Movement



3.11 Power Management

3.11.1 I/O Instruction Restart

I/O Instruction restart is a power management feature of the embedded Pentium processor that allows the processor to re-execute an I/O instruction. In this way, an I/O instruction can alert a sleeping device in a system and SMI# can be recognized before the I/O instruction is re-executed. SMI# assertion causes a wake-up routine to be executed, so the restarted I/O instruction can be executed by the system.

3.11.2 Stop Clock and Auto Halt Powerdown

The embedded Pentium processor uses Stop Clock and Auto Halt Powerdown to immediately reduce the power of each device. These features cause the clock to be stopped to most of the processor's internal units and thus significantly reduce power consumption by the processor as a whole.

Stop clock is enabled by asserting the STPCLK# pin of the embedded Pentium processor. While asserted, the embedded Pentium processor stops execution and does service interrupts, but allows external and interprocessor (Primary and Dual processor) snooping.

AutoHalt Powerdown is entered once the embedded Pentium processor executes a HLT instruction. In this state, most internal units are powered-down, but the embedded Pentium processor recognizes all interrupts and snoops.

Embedded Pentium processor pin functions (D/P#, etc.) are not affected by STPCLK# or AutoHalt.

For additional details on power management, refer to Chapter 12, "Power Management."

3.12 CPUID Instruction

The CPUID instruction provides information to software about the vendor, family, model and stepping of the microprocessor on which it is executing. In addition, it indicates the features supported by the processor.

When executing CPUID:

- If the value in EAX is “0,” then the 12-byte ASCII string “GenuineIntel” (little endian) is returned in EBX, EDX, and ECX. Also, EAX contains a value of “1” to indicate the largest value of EAX which should be used when executing CPUID.
- If the value in EAX is “1,” then the processor version is returned in EAX and the processor capabilities (feature flags) are returned in EDX.
- If the value in EAX is neither “0” nor “1”, the embedded Pentium processor writes “0” to EAX, EBX, ECX, and EDX.

The following EAX value is defined for the CPUID instruction executed with EAX = 1. The processor version EAX bit assignments are given in Figure 3-20. Table 3-13 lists the feature flag bits assignment definitions.

Figure 3-20. EAX Bit Assignments for CPUID

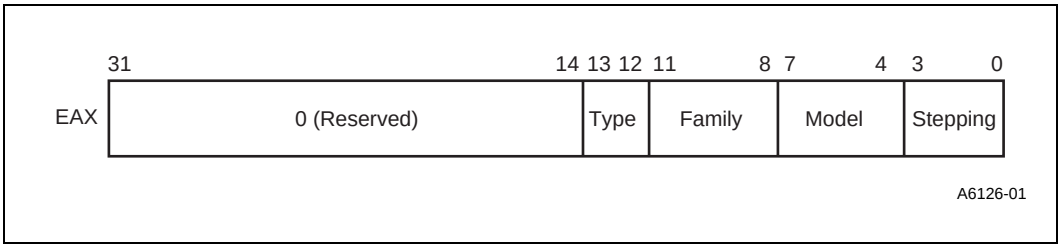


Table 3-13. EDX Bit Assignment Definitions (Feature Flags)

Bit	Name	Value	Description When Flag=1	Comments
0	FPU	1	Floating-point unit on-chip	The processor contains an FPU that supports the Intel 387 floating-point instruction set.
1	VME	1	Virtual Mode Enhancements	The processor supports extensions to virtual-8086 mode.
2	DE	1	Debugging Extension	The processor supports I/O breakpoints, including the CR4.DE bit for enabling debug extensions and optional trapping of access to the DR4 and DR5 registers.
3	PSE	1	Page Size Extension	The processor supports 4-Mbyte pages.
4	TSC	1	Time Stamp Counter	The RDTSC instruction is supported including the CR4.TSD bit for access/privilege control.
5	MSR	1	Embedded Pentium® Processor MSR	Model SpecificRegisters are implemented with the RDMSR, WRMSR instructions.
6	PAE	0	Physical Address Extension	Physical addresses greater than 32 bits are supported.
7	MCE	1	Machine Check Exception	Machine Check Exception, Exception 18, and the CR4.MCE enable bit are supported.
8	CX8	1	CMPXCHG8B Instruction Supported	The compare and exchange 8 bytes instruction is supported.
9	APIC	1	On-chip PIC Hardware Enabled†	The processor contains a local APIC.
10-11		R	Reserved	Do not rely on its value.
12	MTRR	0	Memory Type Range Registers	The processor supports the Memory Type Range Registers specifically the MTRR_CAP register.
13	PGE	0	Page Global Enable	The global bit in the PDE's and PTE's and the CR4.PGE enable bit are supported.
14	MCA	0	Machine Check Architecture	The Machine Check Architecture is supported, specifically the MCG_CAP register.
15-22		R	Reserved	Do not rely on its value.
23	MMX technology	1	Intel Architecture MMX™ technology supported	The processor supports the MMX technology instruction set extensions to the Intel Architecture.
24-31		R	Reserved	Do not rely on its value.

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

The family field for the embedded Pentium processor family is 0101B (5H). The model value for the embedded Pentium processor is 0010B (2H) or 0111B (7H), and the model value for the embedded Pentium processor with MMX technology is 0100B (4H). The model value for the low-power embedded Pentium processor with MMX technology is 1000B (8H)

Note: Use the MMX technology feature bit (bit23) in the EFLAGS register, not the model value, to detect the presence of the MMX technology feature set.

For specific information on the stepping field, consult the embedded Pentium processor family Specification Update. The type field is defined in Table 3-14.

Table 3-14. EAX Type Field Values

Bit 13	Bit 12	Processor Type
0	0	Embedded Pentium® processor, embedded Pentium processor with MMX™ technology or low-power embedded Pentium processor with MMX technology
0	1	Reserved
1	0	Dual embedded Pentium processor
1	1	Reserved

3.13 Model Specific Registers

Each embedded Pentium processor contains certain Model Specific Registers that are used in execution tracing, performance monitoring, testing, and machine check errors. They are unique to that embedded Pentium processor and may not be implemented in the same way in future processors.

Two instructions, RDMSR and WRMSR (read/write model specific registers) are used to access these registers. When these instructions are executed, the value in ECX specifies which model specific register is being accessed.

Software must not depend on the value of reserved bits in the model specific registers. Any writes to the model specific registers should write “0” into any reserved bits.

For more information, refer to Chapter 14, “Model Specific Registers and Functions.”

Microprocessor Initialization and Configuration 4

This chapter covers microprocessor initialization and configuration information for both uni-processor and dual-processor implementations of the embedded Pentium® processor family. For configuration information on symmetric dual-processing mode, refer to “Managing and Designing with the Symmetrical Dual Processing Configuration” on page 4-7.

Before normal operation of the processor can begin, the processor must be initialized by driving the RESET pin active. The RESET pin forces the processor to begin execution in a known state. Several features are optionally invoked at the falling edge of RESET: Built-in-Self-Test (BIST), Functional Redundancy Checking and Three-state Test Mode.

In addition to the standard RESET pin, the processor has implemented an initialization pin (INIT) that allows the processor to begin execution in a known state without disrupting the contents of the internal caches or the floating-point state.

This chapter describes the embedded Pentium processor power up and initialization procedures, and the test and configuration features enabled at the falling edge of RESET.

4.1 Power Up Specifications

During power up, RESET must be asserted while V_{CC} is approaching nominal operating voltage to prevent internal bus contention, which could negatively affect the reliability of the processor.

It is recommended that CLK begin toggling within 150 ms after V_{CC} reaches its proper operating level. For the embedded Pentium® processor with MMX™ technology, it is recommended that the CLK signal begin toggling within 150 ms after the last V_{CC} plane stabilizes. This recommendation is only to ensure long term reliability of the device.

In order for RESET to be recognized, the CLK input needs to be toggling. RESET must remain asserted for 1 millisecond after V_{CC} and CLK have reached their AC/DC specifications.

4.2 Test and Configuration Features

The INIT, FLUSH#, and FRCMC# inputs are sampled when RESET transitions from high to low to determine if BIST will be run, or if three-state test mode, or checker mode will be entered (respectively). If RESET is driven synchronously, these signals must be at their valid level and meet setup and hold times on the clock before the falling edge of RESET. If RESET is asserted asynchronously, these signals must be at their valid level two clocks before and after RESET transitions from high to low.

4.2.1 Built-in Self-Test

Self-test is initiated by driving the INIT pin high when RESET transitions from high to low. No bus cycles are run by the processor during self test. The duration of self test is approximately 2^{19} core clocks. Approximately 70% of the devices in the processor are tested by BIST. The embedded Pentium processor BIST consists of two parts: hardware self-test and microcode self-test. During the hardware portion of BIST, the microcode ROM and all large PLAs are tested. All possible input combinations of the microcode ROM and PLAs are tested.

The constant ROMs, BTB, TLBs, and all caches are tested by the microcode portion of BIST. The array tests (caches, TLBs and BTB) have two passes. On the first pass, data patterns are written to arrays, read back, and checked for mismatches. The second pass writes the complement of the initial data pattern, reads it back, and checks for mismatches. The constant ROMs are tested by using the microcode to add various constants and check the result against a stored value.

Upon successful completion of BIST, the cumulative result of all tests are stored in the EAX register. If EAX contains 0H, then all checks passed; any non-zero result indicates a faulty unit. Note that when an internal parity error is detected during BIST, the processor asserts the IERR# pin and attempts to shutdown.

4.2.2 Three-state Test Mode

When the FLUSH# pin is sampled low when RESET transitions from high to low, the processor enters three-state test mode. The processor floats all of its output pins and bidirectional pins, including pins that are never floated during normal operation (except TDO). Three-state test mode can be initiated to facilitate testing board interconnects. The processor remains in three-state test mode until the RESET pin is asserted again.

4.2.3 Functional Redundancy Checking

The functional redundancy checking (FRC) master/checker configuration input is sampled when RESET is high to determine whether the processor is configured in master mode (FRCMC# high) or checker mode (FRCMC# low). Note, the embedded Pentium processor with MMX technology does not support FRC mode.

The final master/checker configuration of the processor is determined the clock before the falling edge of RESET. When configured as a master, the processor drives its output pins as required by the bus protocol. When configured as a checker, the processor three-states all outputs (except IERR#, PICD0, PICD1 and TDO) and samples the output pins (that would normally be driven in master mode). If the sampled value differs from the value computed internally, the processor asserts IERR# to indicate an error. Note that IERR# is not asserted due to an FRC mismatch until two clocks after the ADS# of the first bus cycle (or in the third clock of the bus cycle).

To avoid an FRC error caused by differences in the unitialized FPU state, FINIT/FNINIT must be used to initialize the FPU state prior to using FSAVE/FNSAVE in FRC mode. The initialization should be done before other FPU activity so that it does not corrupt the previous state.

4.2.4 Lock Step APIC Operation

Lock Step operation is entered by holding BE4# high during the falling edge of RESET. Lock Step operation is not supported by the embedded Pentium processor with MMX technology.

Lock Step operation guarantees recognition of an interrupt on a specific clock by two processors operating together that are using the APIC as the interrupt controller. This functionality is related to FRC operation, but FRC on the APIC pins is not fully supported in this way. There is no FRC comparator on the APIC pins, but mismatches on these pins result in a mismatch on other pins of the processor.

Fault tolerant systems implemented with multiple processors that run identical code sequences and generate identical bus cycles on all clocks may utilize Lock Step operation.

Setup and Hold time specifications PICCLK (in relation to CLK) are added for this functionality. Additionally, there is a requirement to sustain specific integer ratios between the frequencies of PICCLK and CLK. This ratio should support both the maximum bus frequency of the device and the maximum frequency of PICCLK. Details of these specifications can be found in Chapter 7, "Electrical Differences Between Family Members."

4.3 Initialization with RESET, INIT and BIST

Two pins, RESET and INIT, are used to reset the processor in different manners. A "cold" or "power on" RESET refers to the assertion of RESET while power is initially being applied to the processor. A "warm" RESET refers to the assertion of RESET or INIT while V_{CC} and CLK remain within specified operating limits.

Table 4-1 shows the effect of asserting RESET and/or INIT.

Table 4-1. Pentium® Processor Reset Modes

RESET	INIT	BIST Run?	Effect on Code and Data Caches	Effect on FP Registers	Effect on BTB and TLBs
0	0	No	n/a	n/a	n/a
0	1	No	None	None	Invalidated
1	0	No	Invalidated	Initialized	Invalidated
1	1	Yes	Invalidated	Initialized	Invalidated

Toggling either the RESET pin or the INIT pin individually forces the processor to begin execution at address FFFFFFF0H. The internal instruction cache and data cache are invalidated when RESET is asserted (modified lines in the data cache are NOT written back). The instruction cache and data cache are not altered when the INIT pin is asserted without RESET. In both cases, the branch target buffer (BTB) and translation lookaside buffers (TLBs) are invalidated.

After RESET (with or without BIST) or INIT, the processor starts executing instructions at location FFFFFFF0H. When the first Intersegment Jump or Call instruction is executed, address lines A20-A31 are driven low for CS-relative memory cycles and the processor only executes instructions in the lower 1 Mbyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system.

RESET is internally hardwired and forces the processor to terminate all execution and bus cycle activity within two clocks. No instruction or bus activity occurs as long as RESET is active. INIT is implemented as an edge triggered interrupt and is recognized when an instruction boundary is reached. As soon as the processor completes the INIT sequence, instruction execution and bus cycle activity continues at address FFFFFFF0H even if the INIT pin is not deasserted.

At the conclusion of RESET (with or without self-test) or INIT, the DX register contains a component identifier. The upper byte contain 05H and the lower byte contains a stepping identifier.

Table 4-2 defines the processor state after RESET, INIT, and RESET with BIST (built in self-test).

Table 4-2. Register State after RESET, INIT and BIST

Storage Element	RESET (No BIST) (Note 1)	RESET (BIST) (Note 1)	INIT
EAX	0	0 if pass	0
EDX	0500+stepping	0500+stepping	0500+stepping
ECX, EBX, ESP, EBP, ESI, EDI	0	0	0
EFLAGS	2	2	2
EIP	0FFF0	0FFF0	0FFF0
CS	selector = F000	selector = F000	selector = F000
	AR = P, R/W, A	AR = P, R/W, A	AR = P, R/W, A
	base = FFFF0000	base = FFFF0000	base = FFFF0000
	limit = FFFF	limit = FFFF	limit = FFFF
DS, ES, FE, GS, SS	selector = 0	selector = 0	selector = 0
	AR = P, R/W, A	AR = P, R/W, A	AR = P, R/W, A
	base = 0	base = 0	base = 0
	limit = FFFF	limit = FFFF	limit = FFFF
(I/G/L)DTR, TSS	selector = 0	selector = 0	selector = 0
	base = 0	base = 0	base = 0
	AR = P, R/W	AR = P, R/W	AR = P, R/W
	limit = FFFF	limit = FFFF	limit = FFFF
CR0	60000010	60000010	Note 2
CR2, 3, 4	0	0	0
DR3–DR0	0	0	0
DR6	FFFF0FF0	FFFF0FF0	FFFF0FF0
DR7	00000400	00000400	00000400
Time Stamp Counter	0	0	Unchanged
Control and Event Select	0	0	Unchanged
TR12	0	0	Unchanged
All other MSR's	Undefined	Undefined	Unchanged
CW	0040	0040	Unchanged
SW	0	0	Unchanged
TW	5555	5555	Unchanged
FIP, FEA, FCS, FDS, FOP	0	0	Unchanged
FSTACK	0	0	Unchanged
SMBASE	30000	30000	Unchanged
Data and Code Cache	Invalid	Invalid	Unchanged
Code Cache TLB, Data Cache TLB, BTB, SDC	Invalid	Invalid	Invalid

NOTES:
1. Register States are given in hexadecimal format.
2. CD and NW are unchanged, bit 4 is set to 1, all other bits are cleared.

4.3.1 Recognition of Interrupts after RESET

To guarantee recognition of the edge sensitive interrupts (FLUSH#, NMI, R/S#, SMI#) after RESET or after RESET with BIST, the interrupt input must not be asserted until four clocks after RESET is deasserted, regardless of whether or not BIST is run.

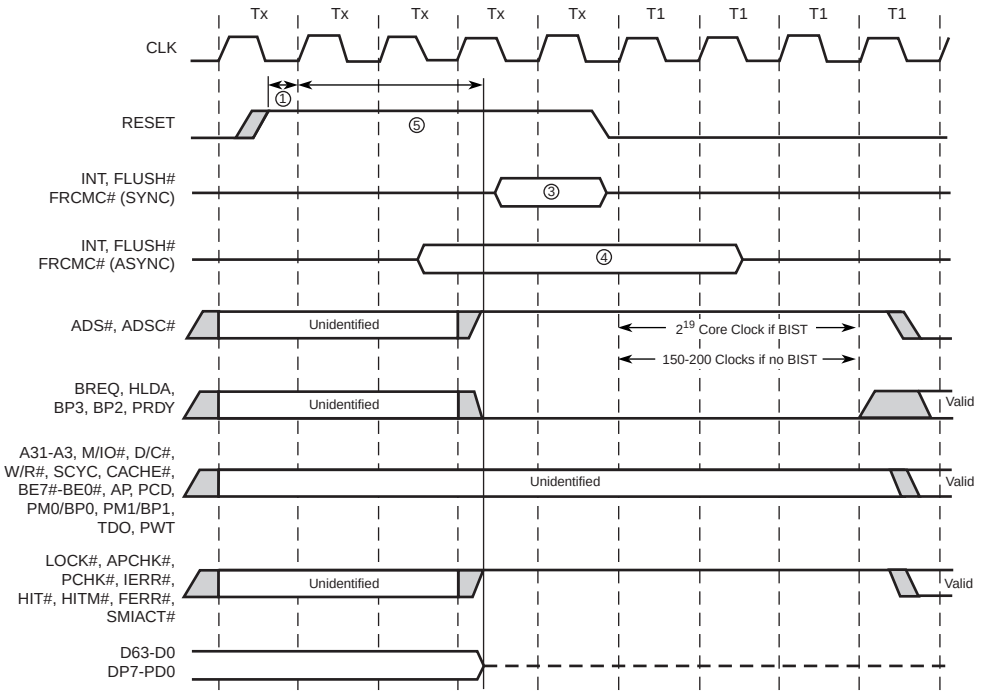
4.3.2 Pin State During/After RESET

The processor recognizes and responds to HOLD, AHOLD, and BOFF# during RESET. Figure 4-1 shows the processor state during and after a power on RESET if HOLD, AHOLD, and BOFF# are inactive. Note that the address bus pins (A31–A3, AP, BE7#–BE0#) and cycle definition pins (M/IO#, D/C#, W/R#, CACHE#, SCYC, PCD, PWT, PM0/BP0, PM1/BP1 and LOCK#) are undefined from the time RESET is asserted until the start of the first bus cycle.

The following lists the state of the output pins after RESET assuming HOLD, AHOLD, and BOFF# are inactive, boundary scan is not invoked, and no internal parity error is detected.

- High: LOCK#, ADS#, ADSC#, APCHK#, PCHK#, IERR#, HIT#, HITM#, FERR#, SMIACT#
- Low: HLDA, BREQ, BP3, BP2, PRDY
- High Independence: D63–D0, DP7–DP0
- Undefined: A31–A3, AP, BE7#–BE0#, W/R#, M/IO#, D/C#, PCD, PWT, CACHE#, TDO, SCYC, PM0/BP0, PM1/BP1

Figure 4-1. Pin States during Reset



NOTES:

1. RESET must meet setup and hold times to guarantee recognition on a specific clock edge. If RESET does not need to be recognized on a specific clock edge, it may be asserted asynchronously.

2. At power up, RESET needs to be asserted for 1 ms after V_{CC} and CLK have reached their AC/DC specifications. For warm reset, RESET needs to be asserted for at least 15 clocks while V_{CC} and CLK remain within specified operating limits.

3. If RESET is driven synchronously, FLUSH#, FRCMC# and INIT must be at their valid level and meet setup and hold times to the clock before the falling edge of RESET.

4. If RESET is driven asynchronously, FLUSH#, FRCMC# and INIT must be at their valid level two clocks before and after the falling edge of RESET.

5. An assertion of RESET takes at least two clocks to affect the pins.

A6127-01

4.4 Managing and Designing with the Symmetrical Dual Processing Configuration

4.4.1 Dual Processor Bootup Protocol

4.4.1.1 Bootup Overview

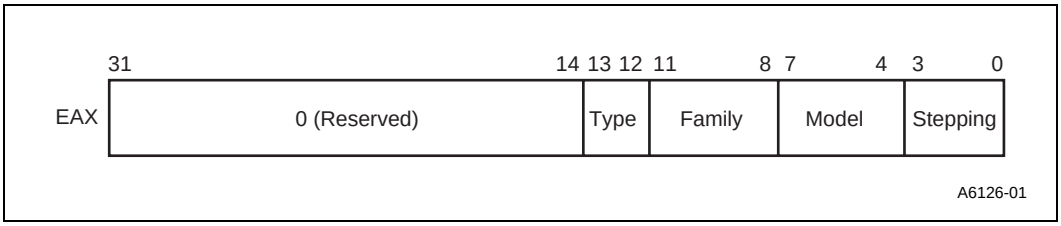
Systems using the embedded Pentium processor may be equipped with a second processor socket. For correct system operation, the processor must be able to identify the presence and type of the second processor (such as a Dual processor). Furthermore, since upgrade processors typically are installed in the field by end users, system configuration may change between any two consecutive power-down/up sequences. The system must therefore have a mechanism to ascertain the system configuration during boot time. The boot up handshake protocol provides this mechanism.

4.4.1.2 BIOS/Operating System Requirements

The BIOS or HAL (hardware abstraction layer) of the operating system software should be generic, independent of the kind of OEM or upgrade processor present in the system. BIOS/HAL are specific to the system hardware, and should not need any change when an upgrade processor is installed. For dual processors, if the BIOS is not DP-ready, it will be up to the operating system to initialize and configure the dual processor appropriately.

The CPUID instruction is used to deliver processor-specific information. The embedded Pentium processor CPUID status has been extended to supply the processor type information which includes “turbo-upgrade” classification (“type” field: bits 13-12 = 0-1). For upgradability with a future Pentium Overdrive processor, system software must allow the type field of the EAX register following the CPUID instruction to contain the values for both the embedded Pentium processor and the Pentium Overdrive processor. Note also that the model field of the CPUID is different for a Pentium OverDrive processor.

Figure 4-2. EAX Bit Assignments for CPUID



4.4.1.3 System Requirements

The number of Dual processors per Primary processor is limited to 1.

This bootup handshake protocol requires enabling the local APIC module using the APICEN pin. The startup IPI must be sent via the local APICs. Once the Dual processor has been initialized, software can later disable the local APIC module using several methods. These methods and their considerations are discussed in “APIC Interrupt Controller” on page 3-35.

The protocol does not preclude more generic multiprocessing systems where multiple pairs of Primary and Dual processors may exist on the system bus.

4.4.1.4 Start-up Behavior

On RESET and INIT (message or pin), the processor begins execution at the reset vector (0FFFFFFF0H). The Dual processor waits for a startup IPI from the BIOS or operating system via the local APIC of the processor. The INIT IPI can be used to put the embedded Pentium processor or Dual processor to sleep (once the INIT IPI is received, the processor must wait for the startup IPI).

The startup IPI is specifically provided to start the Dual processor's execution from a location other than the reset vector, although it also can be used for the processor. The startup IPI is sent by the system software via the local APIC by using a delivery mode of 110B. The startup IPI must include an 8-bit vector that defines the starting address. The starting address = *000 VV 000H*, where *VV* indicates the vector field (in hex) passed through the IPI.

The 8-bit vector defines the address of a 4 Kbyte page in the Intel architecture Real Mode Space (1 Mbyte space). For example, a vector of 0CDH specifies a startup memory address of 000CD000H. This value is used by the processor to initialize the segment descriptor for the upgrade's CS register as follows:

- The CS selector is set to the startup memory address/16 (real mode addressing)
- The CS base is set to the startup memory address
- The CS limit is set to 64 Kbytes
- The current privilege level (CPL) and instruction pointer (IP) are set to 0

Note: Vectors of 0A0H to 0BFH are reserved by Intel. Do *not* use them.

The benefit of the startup IPI is that it does not require the APIC to be software enabled (the APIC must be hardware enabled via the APICEN pin) and does not require the interrupt table to be programmed. Startup IPIs are non-maskable and can be issued at any time to the embedded Pentium processor or Dual processor. If the startup IPI message is not preceded by a RESET or INIT (message or pin), it is ignored.

It is the responsibility of the system software to resend the startup IPI message if there is an error in the IPI message delivery. Although the APIC need not be enabled in order to send the startup IPI, the advantage to enabling the APIC prior to sending the startup IPI is to allow APIC error handling to occur via the APIC error handling entry of the local vector table (ERROR INT or LVT3 at APIC address 0FEE00370H). Otherwise, the system software would have to poll the delivery status bit of the interrupt command register to determine if the IPI is pending (Bit 12 of the ICR=1) and resend the startup IPI if the IPI remains pending after an appropriate amount of time.

4.4.1.5 Dual-Processor Presence Indication

The bootup handshake protocol becomes aware that an additional processor is present through the DPEN# pin. The second processor is guaranteed to drive this signal low during RESETs falling edge. If the system needs to remember the presence of a second processor for future use, it must latch the state of the DPEN# pin during the falling edge of RESET.

4.4.2 Dual-Processor Arbitration

The embedded Pentium processor incorporates a private arbitration mechanism that allows the Primary and Dual processors to arbitrate for the shared processor bus without assistance from a bus controller. The arbitration scheme is architected in such a way that the dual processor pair appears as a single processor to the system.

The processor arbitration logic uses a fair arbitration scheme. The arbitration state machine is designed to efficiently use the processor bus bandwidth. The dual processor pair supports inter-processor pipelining of most bus transactions. Furthermore, the arbitration mechanism does not introduce any dead clocks on bus transactions.

4.4.2.1 Basic Dual-Processor Arbitration Mechanism

The basic set of arbitration premises requires that the processor check the second socket (Socket 7) for a processor every time the processor enters reset. To perform the checking of the Socket 7 and to perform the actual boot sequence, the processor in the 296-pin socket always comes out of reset as the most-recent master (MRM). This requires the part in the Socket 7 to always come out of reset as the least-recent master (LRM).

The LRM processor requests ownership of the processor bus by asserting the private arbitration request pin, PBREQ#. The processor that is currently the MRM and owns the bus grants the bus to the LRM as soon as any pending bus transactions have completed. The MRM grants the bus to the LRM immediately if that processor has a pipelined cycle to issue. The MRM notifies that the LRM can assume ownership by asserting the private arbitration grant pin, PBGNT#. The PBREQ# pin is always the output of the LRM and the PBGNT# is always an input to the LRM.

A processor can park on the processor bus if there are no requests from the LRM. A parked processor can be running cycles or just sitting idle on the bus. If a processor just ran a cycle on the bus and has another cycle pending without an LRM request, the processor runs the second cycle on the bus.

Locked cycles present an exception to the simple arbitration rules. All locked cycles are performed as atomic operations without interrupt from the LRM. An exception to this rule is when a locked access causes an assertion of PHITM# by the LRM. In this case, the MRM grants the bus to the LRM and allows the writeback to complete.

The normal system arbitration pins (HOLD, HLDA, BOFF#) functions the same as in uni-processor mode. Thus, the dual-processor pair always factors the state of the processor bus as well as the state of the local arbitration before actually running a cycle on the processor bus.

4.4.2.2 Dual-Processor Arbitration Interface

Figure 4-3 details the hardware arbitration interface**.**

Note: For proper operation, PBREQ# and PBGNT# must not be loaded by the system.

Figure 4-3. Dual-Processor Arbitration Interface

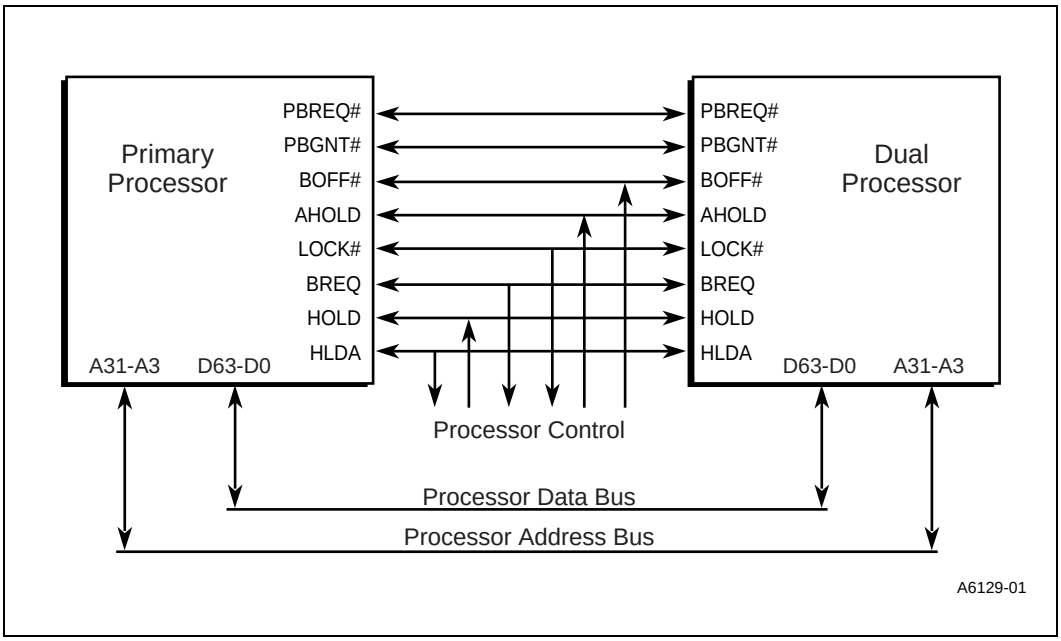


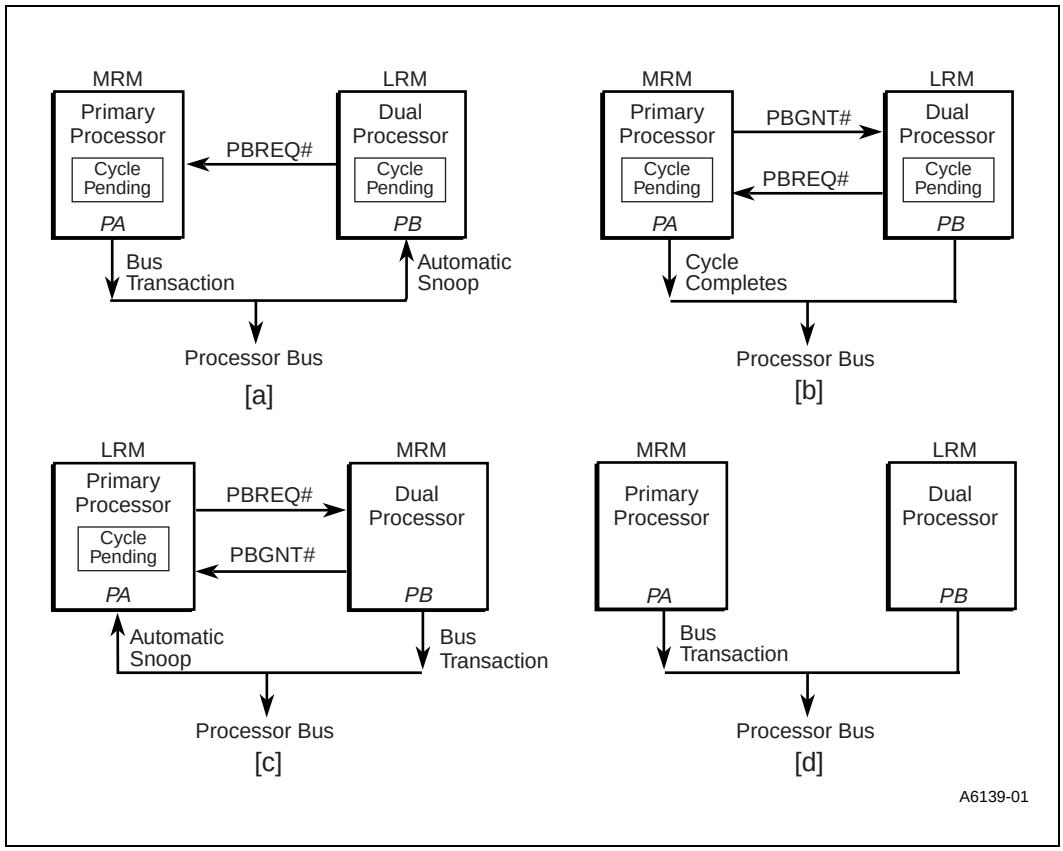
Figure 4-4 shows a typical arbitration exchange.

Diagram (a) of Figure 4-4 shows *PA* running a cycle on the processor bus with a transaction pending. At the same time, *PB* has a cycle pending and has asserted the PBREQ**#** pin to notify *PA* that *PB* needs the bus.

Diagram (b) of Figure 4-4 shows *PA's* cycle completing with an NA# or the last BRDY#. Note here that *PA* does not run the pending cycle, instead, *PA* grants the bus to *PB* to allow *PB* to run its pending cycle.

In Diagram (c) of Figure 4-4, *PB* is running the pending transaction on the processor bus, and *PA* asserts a request for the bus to *PB*. The bus is granted to *PA*, and Diagram (d) of Figure 4-4 shows *PA* running the last pending cycle on the bus.

Figure 4-4. Typical Dual-Processor Arbitration Example



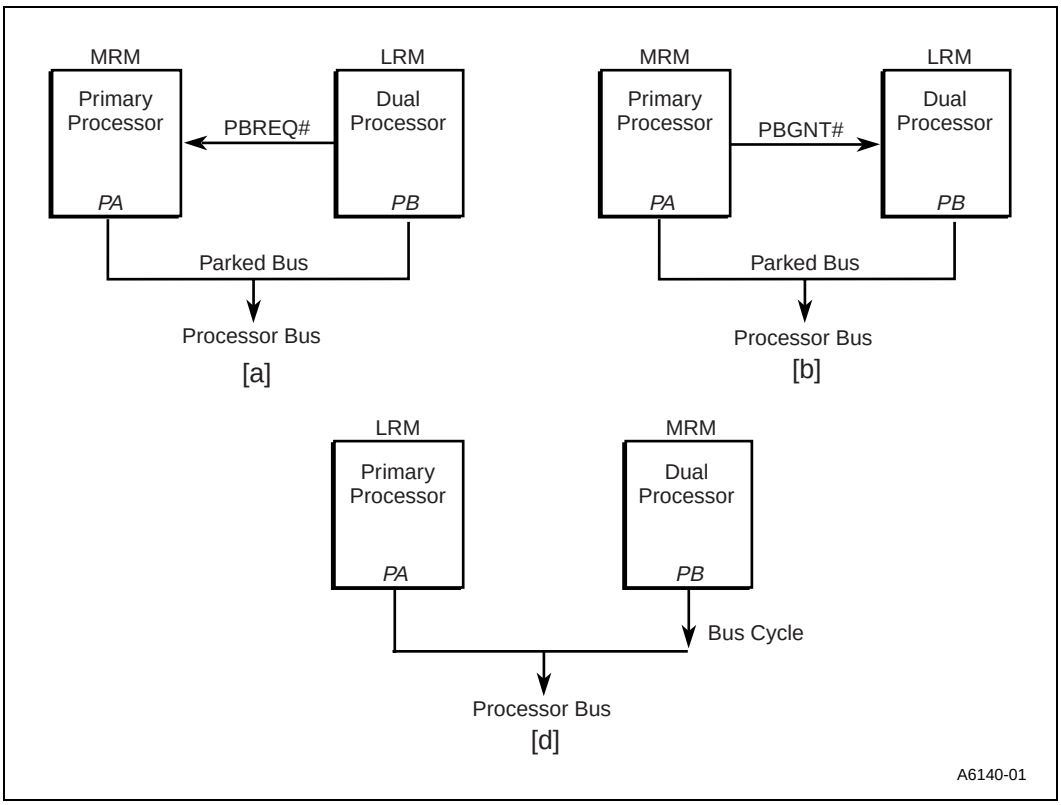
4.4.2.3 Dual-Processor Arbitration from a Parked Bus

When both processors are idle on the processor bus, and the LRM wants to issue an ADS#, there is an arbitration delay in order that it may become the MRM. Figure 4-5 shows how the embedded Pentium processor dual-processor arbitration mechanism handles this case.

This example shows the arbitration necessary for the LRM to gain control of the idle processor bus in order to drive a cycle. In this example, *PA* is the Primary processor, and *PB* is the Dual processor.

Diagram (a) of Figure 4-5 shows *PB* requesting the bus from the MRM (*PA*). Diagram (b) of Figure 4-5 shows *PA* granting control of the bus to *PB*. Diagram (c) of Figure 4-5 shows *PB*, now the MRM, issuing a cycle.

Figure 4-5. Arbitration from LRM to MRM when Bus is Parked



4.4.3 Dual-Processor Cache Consistency

The embedded Pentium processor incorporates a mechanism to maintain cache coherency with the Dual processor. The mechanism allows a dual processor to be inserted into the upgrade socket without special considerations for the system hardware or software. The presence or absence of the dual processor is totally transparent to the system.

4.4.3.1 Basic Cache Consistency Mechanism

A private snoop interface has been added to the embedded Pentium processor. The interface consists of two pins (PHIT#, PHITM#) that only connect between the two sockets. The dual processors arbitrate for the system bus via two private arbitration pins (PBREQ#, PBGNT#).

The LRM processor initiates a snoop sequence for all ADS# cycles to memory that are initiated by the MRM. The LRM processor asserts the private hit indication (PHIT#) if the data accessed (read or written) by the MRM matches a valid cache line in the LRM. In addition, if the data requested by the MRM matches a valid cache line in the LRM that is in the modified state, the LRM asserts the PHITM# signal. The system snooping indication signals (HIT#, HITM#) do not change state as a result of a private snoop.

The processor supports system snooping via the EADS# pin in the same manner in which the processor supports system snooping.

The private snoop interface is bidirectional. The processor that is currently the MRM samples the private snoop interface, while the processor that is the LRM drives the private snoop signals.

The MRM initiates a self backoff sequence if the MRM detects an assertion of the PHITM# signal while running a bus cycle. The self backoff sequence involves the following steps:

1. The MRM allows the cycle that was requested on the bus to finish. However, the MRM ignores the data returned by the system.
2. The MRM-LRM exchanges ownership of the bus (as well as MRM-LRM state) to allow the LRM to write the modified data back to the system.
3. The bus ownership will exchange one more time to allow the original bus master ownership of the bus. At this point the MRM retries the cycle, receiving the fresh data from the system or writing the data again.

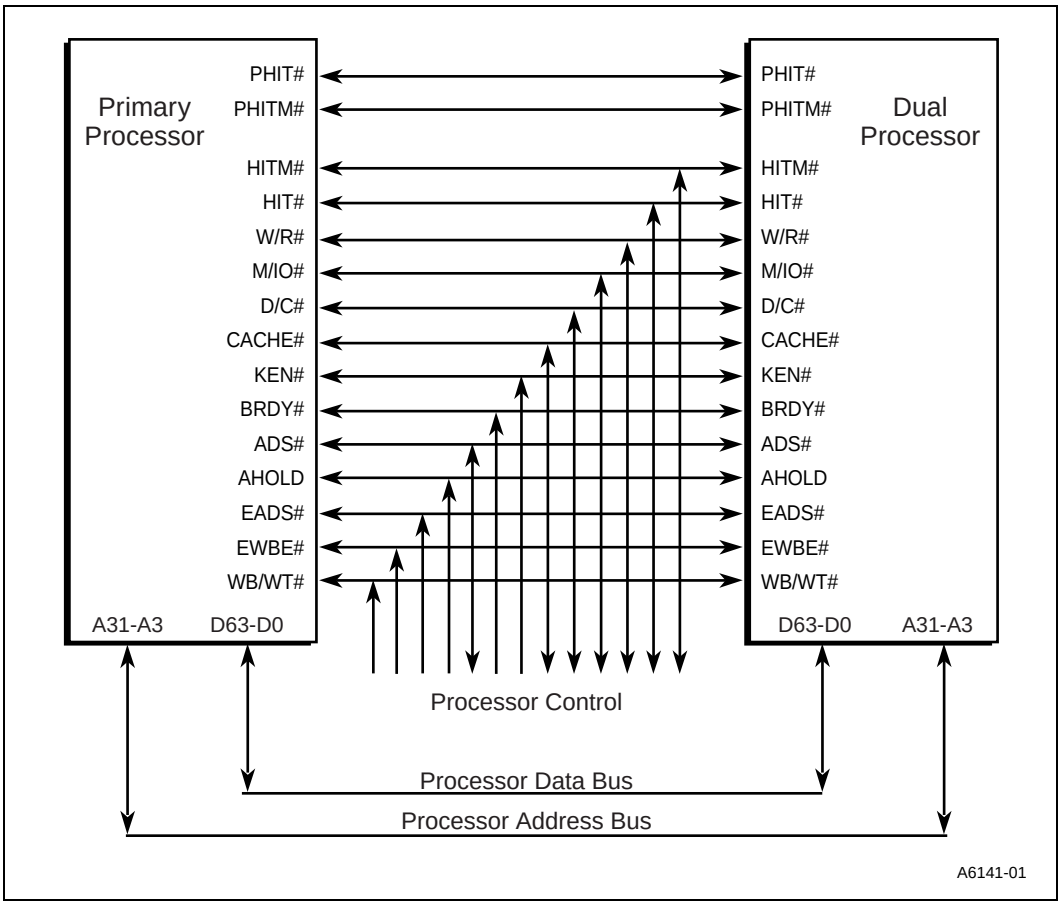
The MRM uses an assertion of the PHIT# signal as an indication that the requested data is being shared with the LRM. Independent of the WB/WT# pin, a cache line is placed in the cache in the shared state if PHIT# is asserted. This makes all subsequent writes to that line externally visible until the state of the line becomes exclusive (E or M states). In a uniprocessor system, the line may have been placed in the cache in the E state. In this situation, all subsequent writes to that line are not visible on the bus until the state is changed to I.

4.4.3.2 Cache Consistency Interface

Figure 4-6 details the hardware cache consistency interface.

Note: For proper operation, PHIT# and PHITM# must not be loaded by the system.

Figure 4-6. Cache Consistency Interface



4.4.3.3 Pin Modifications Due to the Dual-Processor

The processor, when operating in dual processing mode, modifies the functionality of the following signals:

- A20M#, ADS#, BE4#–BE0#, CACHE#, D/C#, FERR#, FLUSH#, HIT#, HITM#, HLDA, IGNNE#, LOCK#, M/IO#, PCHK#, RESET, SCYC, SMIACT#, W/R#

Table 4-10 on page 4-27 summarizes the functional changes of all the pins in dual processor mode.

4.4.3.4 Locked Cycles

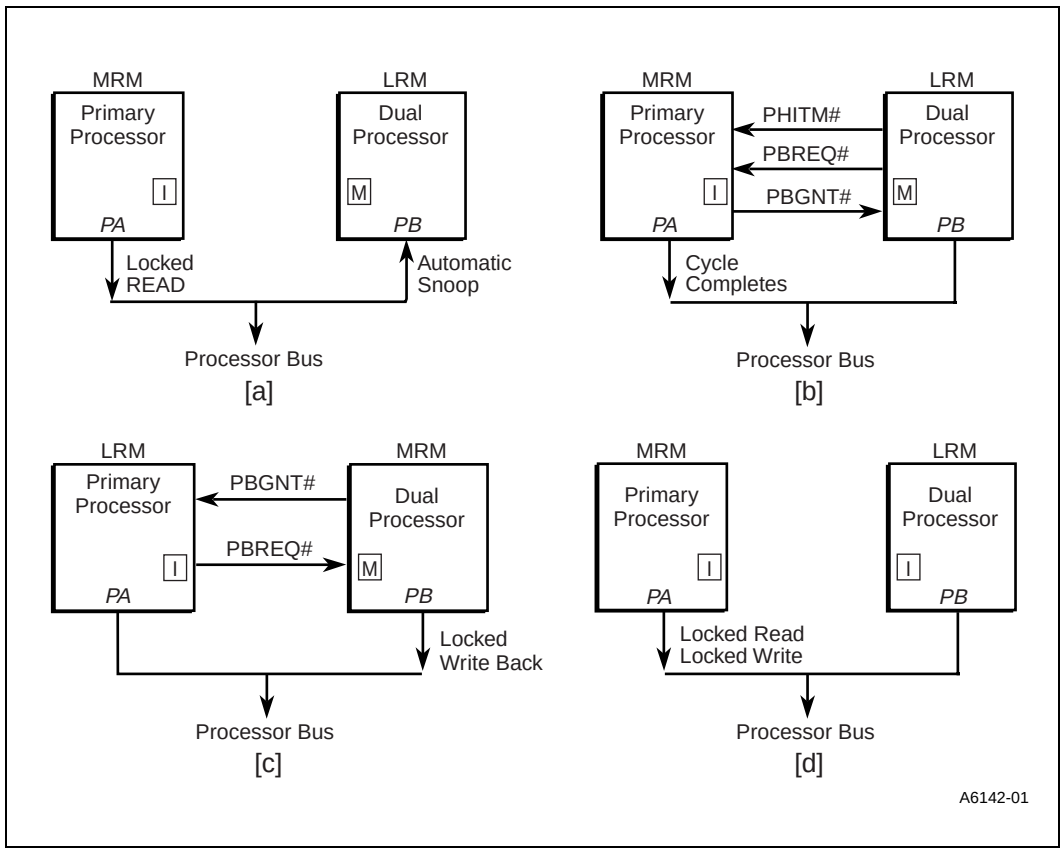
The processor implements atomic bus transactions by asserting the LOCK# pin. Atomic transactions can be initiated explicitly in software by using a LOCK prefix on specific instructions. In addition, atomic cycles may be initiated implicitly for instructions or transactions that perform locked read-modify-write cycles. By asserting the LOCK# pin, the processor indicates to the system that the bus transaction in progress cannot be interrupted.

Lock cycles adhere to the following sequence:

1. An unlocked writeback occurs when a cache line is in the modified state in the MRM processor. Two unlocked write back cycles may be required if the locked item spans two cache lines that are both in the modified state.
2. A locked read to a cache line that is in the shared, exclusive or invalid state is always run on the system bus. The cache line always is moved to the invalid state at the completion of the cycle. A locked read cycle that is run by the MRM could hit a line that is in the modified state in the LRM. In this case, the LRM asserts the PHITM# signal, indicating that the requested data is modified in the LRM data cache. The MRM completes the locked read, but ignores the data returned by the system. The components exchange ownership of the bus, allowing the Modified cache line to be written back with LOCK# still active. The sequence completes with the original bus owner re-running the locked read followed by a locked write. The sequence is as shown in Figure 4-7.

In Figure 4-7, the small box inside each processor indicates the state of an individual cache line in the sequence shown above. Diagram (c) of Figure 4-7 shows the locked writeback occurring as a result of the inter-processor snoop hit to the M-state line.

Figure 4-7. Dual-Processor Cache Consistency for Locked Accesses



4.4.3.5 External Snoop Examples

Example 4-1. During a Write to an M-State Line

The following set of diagrams illustrates the actions performed when one processor attempts a write to a line that is contained in the cache of the other processor. In this situation, the cached line is in the M state in the LRM processor. The external snoop and the write are to the same address in this example. In this example, *PA* is the Primary processor, and *PB* is the Dual processor.

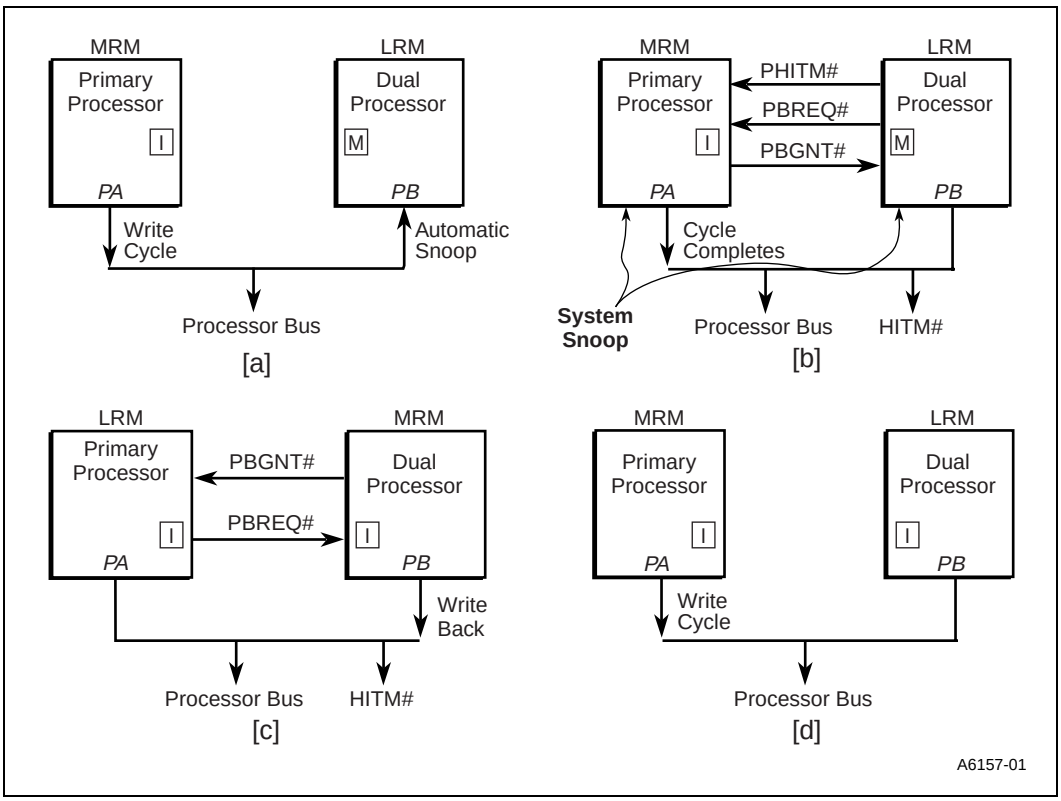
In diagram (a) of Figure 4-8, processor *PA* starts a write cycle on the bus to a line that is in the M state in processor *PB*. Processor *PB* notifies *PA* that the write transaction has hit an M-state line in diagram (b) of Figure 4-8 by asserting the PHITM# signal. The MRM (*PA*) completes the write cycle on the bus as if the LRM processor did not exist.

In this example, an external snoop happens just as the write cycle completes on the bus, but before *PB* has a chance to write the modified data back to the system memory. Diagram (b) of Figure 4-8 shows *PB* asserting the HITM# signal, informing the system that the snoop address is cached in the dual processing pair and is in the modified state. The external snoop in this example is hitting the same line that caused the PHITM# signal to be asserted.

Diagram (c) of Figure 4-8 shows that an arbitration exchange has occurred on the bus, and *PB* is now the MRM. Processor *PB* writes back the M state line; it appears to the system as if a single processor was completing a snoop transaction.

Finally, diagram (d) of Figure 4-8 shows processor *PA* re-running the original write cycle after *PB* has granted the bus back to *PA*.

Figure 4-8. Dual-Processor Cache Consistency for External Snoops



Example 4-2. During an MRM Self-Backoff

The following diagrams show an example in which an external snoop hits an M-state line during a self backoff sequence.

In this example, *PA* is the Primary processor, and *PB* is the Dual processor.

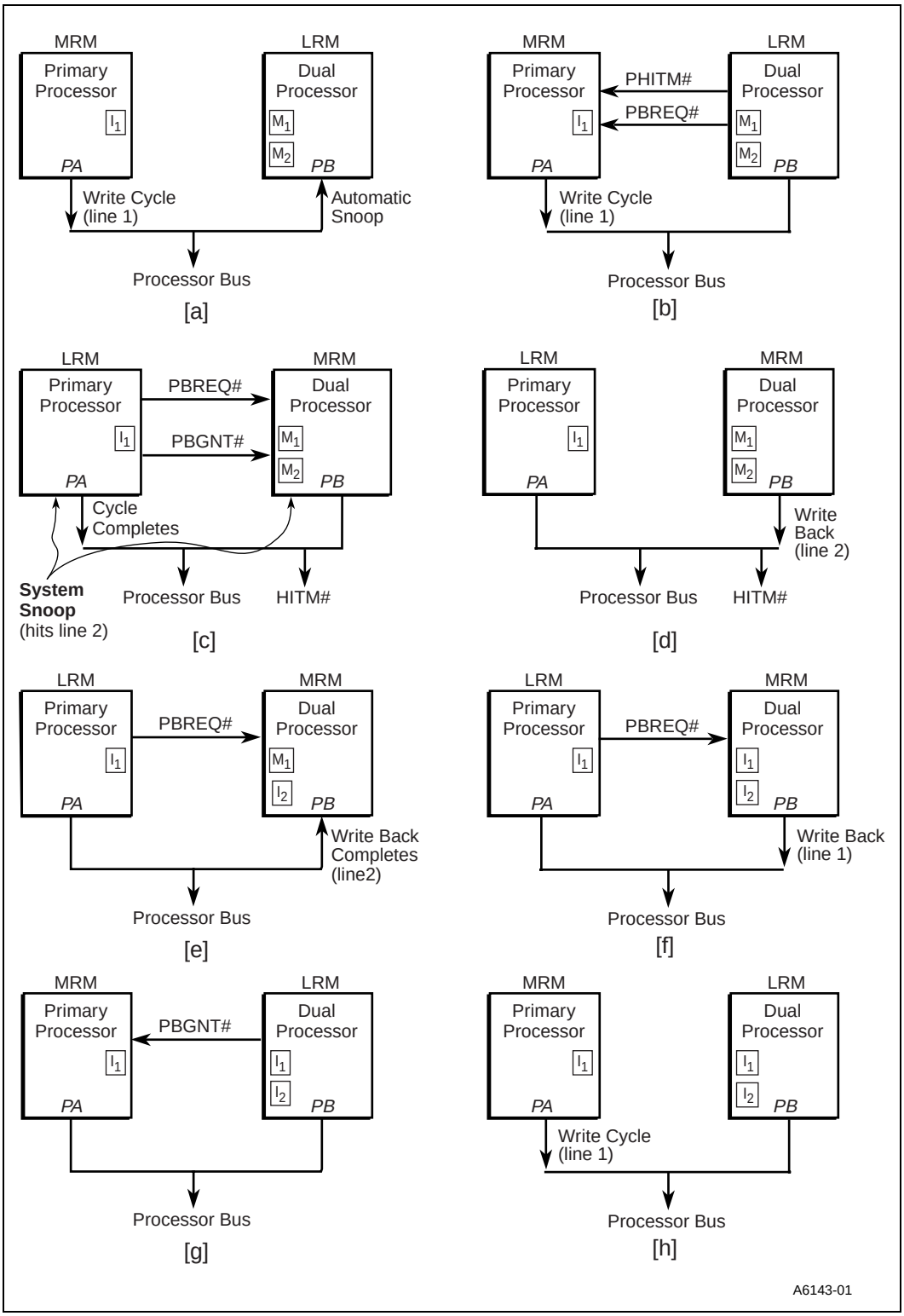
In diagram (a) of Figure 4-9 processor *PA* initiates a write cycle that hits a line that is modified in processor *PB*. In diagram of (b) of Figure 4-9, processor *PB* notifies *PA* that the line is modified in its cache by asserting the PHITM# signal.

Diagram (c) of Figure 4-9 shows an external snoop occurring just as the bus arbitration has exchanged ownership of the bus. Processor *PB* asserts the HITM# signal to notify the system that the external snoop has hit a line in the cache. In this example, the external snoop hits a different line that was just hit on the private snoop.

In diagram (d) of Figure 4-9, processor *PB* takes ownership of the processor bus from *PA*. Processor *PB* initiates a writeback of the data just hit on the external snoop even though a writeback due to the private snoop is pending. The external snoop causes processor *PB* to delay the writeback that was initiated by the private snoop (to line 1).

Diagram (f) of Figure 4-9 shows the writeback of the modified data hit during the initial private snoop. Processor *PA* then restarts the write cycle for the second time, and completes the write cycle in Diagram (h) of Figure 4-9.

Figure 4-9. Dual-Processor Cache Consistency for External Snoops



4.4.3.6 State Transitions Due to Dual-Processor Cache Consistency

The following tables outline the state transitions that a cache line can encounter during various conditions.

Table 4-3. Read Cycle State Transitions Due to Dual-Processor

Present State	Pin Activity	Next State	Description
M	n/a	M	Read hit. Data is provided to the processor core by the cache. No bus activity.
E	n/a	E	Read hit. Data is provided to the processor core by the cache. No bus activity.
S	n/a	S	Read hit. Data is provided to the processor core by the cache. No bus activity.
I	CACHE#(L) & KEN#(L) & WB/WT#(H) & PHIT#(H) & PWT(L)	E	Cache miss. The cacheability information indicates that the data is cacheable. A bus cycle is requested to fill the cache line. PHIT#(H) indicates that the data is not shared by the LRM processor.
I	CACHE#(L) & KEN#(L) & [WB/WT#(L) + PHIT#(L) + PWT(H)]	S	Cache miss. The line is cacheable and a bus cycle is requested to fill the cache line. In this case, either the system or the LRM is sharing the requested data.
I	CACHE#(H) + KEN#(h)	I	Cache miss. The system or the processor indicates that the line is not cacheable.

NOTE: The assertion of PHITM# would cause the requested cycle to complete as normal, with the requesting processor ignoring the data returned by the system. The LRM processor would write the data back and the MRM would retry the cycle. This is called a self backoff cycle.

Table 4-4. Write Cycle State Transitions Due to Dual-Processor

Present State	Pin Activity	Next State	Description
M	n/a	M	Write hit. Data is written directly to the cache. No bus activity.
E	n/a	M	Write hit. Data is written directly to the cache. No bus activity.
S	PWT(L) & WB/WT#(H)	E	Write hit. Data is written directly to the cache. A write-through cycle is generated on the bus to update memory and invalidate the contents of other caches. The LRM invalidates the line if it is sharing the data. The state transition from S to E occurs AFTER the write completes on the processor bus.
S	PWT(H) + WB/WT#(L)	S	Write hit. Data is written directly to the cache. A write-through cycle is generated on the bus to update memory and invalidate the contents of other caches. The LRM invalidates the line if it is sharing the data.
I	n/a	I	Write miss (the Pentium® processor does not support write allocate). The LRM invalidates the line if it is sharing the data.

Table 4-5. Inquire Cycle State Transitions Due to External Snoop

Present State	Next State (INV=1)	Next State (INV=0)	Description
M	I	S	Snoop hit to an M-state line. HIT# and HITM# are asserted, followed by a writeback of the line.
E	I	S	Snoop hit. HIT# will be asserted.
S	I	S	Snoop hit. HIT# will be asserted.
I	I	I	Snoop miss.

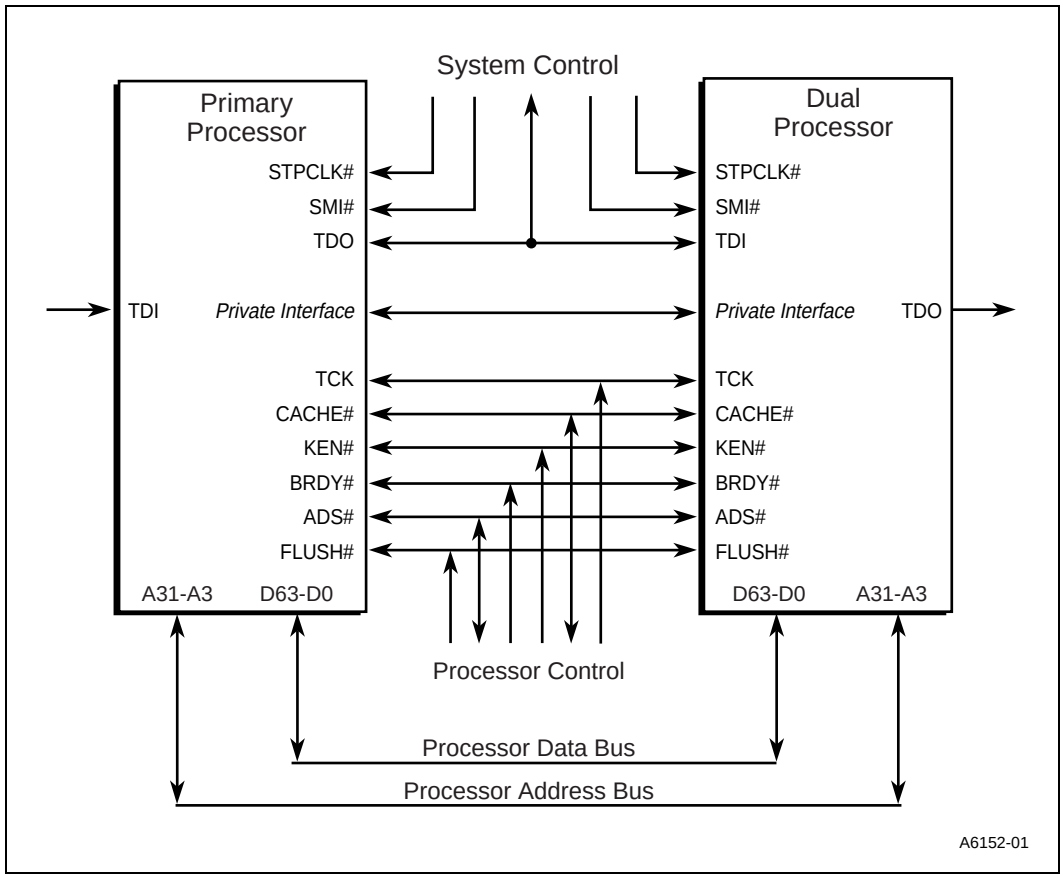
Table 4-6. State Transitions in the LRM Due to Dual-Processor "Private" Snooping

Present State	Next State (MRM Write)	Next State (MRM Read)	Description
M	I	S	Snoop hit to an M state line. PHIT# and PHITM# are asserted, followed by a write-back of the line. Note that HIT# and HITM# are NOT asserted.
E	I	S	Snoop hit. PHIT# is asserted.
S	I	S	Snoop hit. PHIT# is asserted.
I	I	I	Snoop miss.

4.5 Designing with Symmetrical Dual Processors

Figure 4-10 shows how a typical system might be configured to support the Dual processor.

Figure 4-10. Dual-Processor Configuration



Refer to Table 4-10 on page 4-27 for a complete list of dual processor signal connection requirements.

4.5.1 Dual Processor Bus Interface

The processor in the dual-processor configuration is designed to have an identical bus interface to a standard processor system. The processor in dual processor mode has the capability to run the following types bus of cycles:

- Single reads and writes from one processor.
- Burst reads and writes from one processor.
- Address pipelining with up to two outstanding bus cycles from one processor.
- Inter-processor address pipelining with up to two outstanding bus cycles, one from each processor.

All cycles run by the two processors are clock-accurate to corresponding processor bus cycles.

4.5.1.1 Intra- and Inter-Processor Pipelining

In uni-processor mode, the embedded Pentium processor supports bus pipelining with the use of the NA# pin. The bus pipelining concept has been extended to the dual processor pair by allowing inter-processor pipelining. This mechanism allows an exchange between LRM and MRM on assertions of NA#.

When NA# is sampled low, the current MRM processor may drive one more cycle onto the bus or it may grant the address bus and the control bus to the LRM. The MRM gives the bus to the LRM only if its current cycle can have another cycle pipelined into it.

The cacheability (KEN#) and cache policy (WB/WT#) indicators for the current cycle are sampled either in the same clock in which NA# is sampled or with the first BRDY# of the current cycle, whichever comes first.

There are no restrictions on NA# due to dual processing mode.

Inter-processor pipelining is not supported in some situations, as shown in Table 4-7.

Table 4-7. Primary and Dual Processor Pipelining

Cycle Types		Primary and Dual Processor Pipelining		
		Inter-processor	Intra-processor	
First Cycle	**Pipelined Cycle**	**Primary<>Dual**	**Primary<>Primary**	**Dual<>Dual**
Write Back	X	No	No	No
LOCK#	X	No	No	No
X	Write Back	No	No	No
X	LOCK#	No	No	No
Write	**Write**	**No**	**Yes**	**Yes**
Write	Read	Yes	Yes	Yes
Read	Write	Yes	Yes	Yes
Read	Read	Yes	Yes	Yes
I/O	I/O†	**Yes**	**No**	**No**

† I/O write cycles may not be inter-processor pipelined into I/O write cycles

The table indicates that, unlike the uni-processor system, back-to-back write cycles are never pipelined between the two processors.

The processor alone may pipeline I/O cycles into non-I/O cycles, non-I/O cycles into I/O cycles, and I/O cycles into I/O cycles only for OUTS or INS (e.g., string instructions). I/O cycles may be pipelined in any combination (barring writes into writes) between the Primary and Dual processors.

4.5.1.2 FLUSH# Cycles

The on-chip caches can be flushed by asserting the FLUSH# pin. The FLUSH# pin must be connected to both the Primary and Dual processor parts. All cache lines in the instruction cache and all lines in the data cache that are not in the modified state are invalidated when the FLUSH# pin is asserted. All modified lines in the data cache are written back to system memory and then marked as invalid in the data cache. The processor runs a special bus cycle to indicate that the flush process has completed.

The embedded Pentium processor incorporates the following mechanism to present to the system a unified view of the cache flush operation when used with a Dual processor part:

1. FLUSH# is asserted by the system.
2. The Dual processor requests the bus (if it is not already MRM when FLUSH# is recognized). The Dual processor will always perform the cache flush operation first, but will not run a flush special cycle on the system bus.
3. The Dual processor completes writebacks of modified cache lines, and invalidates all others.
4. Once the Dual processor caches are completely invalid, the processor grants the bus to the Primary processor.
5. The Primary processor completes any pending cycles. The Primary processor may have outstanding cycles if the Dual processor initiated its flush operation prior to the Primary processor completing pending operations.
6. Primary processor flushes both of its internal caches and runs the cache flush special cycle. The Primary processor maintains its status of MRM. The Dual processor halts all code execution while the Primary processor is flushing its caches, and does not begin executing code until it recognizes the flush acknowledge special cycle.

The atomic flush operation assumes that the system can tolerate potentially longer interrupt latency during flush operations. The interrupt latency in a dual processor system can be double the interrupt latency in a single processor system during flush operations.

The processor primary cache can be flushed using the WBINVD instruction. In a dual processor system, the WBINVD instruction only flushes the cache in the processor that executed the instruction. The other processor's cache will be intact.

If the FLUSH# signal is deasserted before the corresponding Flush Acknowledge cycle, the FLUSH# signal *must* not be asserted again until the Flush Acknowledge cycle is completed. Similarly, if the FLUSH# signal is asserted in dual processing mode, it must be deasserted at least one clock prior to BRDY# of the Flush Acknowledge cycle to avoid dual-processor arbitration problems. This requirement does not apply to a uni-processor system. In a dual processor system, a single Flush Acknowledge cycle is generated after the caches in both processors have been flushed.

Warning: If FLUSH# is recognized active a second time by the Primary and Dual processors prior to the completion of the Flush Acknowledge special cycle, the private bus arbitration state machines will be corrupted.

4.5.1.3 Arbitration Exchange with Bus Parking

The dual processor pair supports a number of different types of bus cycles. Each processor can run single-transfer cycles or burst-transfer cycles. A processor can only initiate bus cycles if it is the MRM. To gain ownership of the bus, the LRM processor requests the bus from the MRM processor by asserting PBREQ#.

In response to PBREQ# the MRM grants the address and control buses to the LRM by asserting PBGNT#. If NA# is not asserted or if the current cycle on the bus is not capable of being pipelined, the MRM waits until the end of the active cycle before granting the bus to the LRM. Once PBGNT# is asserted, since the bus is idling, the LRM immediately becomes the MRM. While the MRM, the processor owns the address and the control buses and can therefore start a new cycle.

4.5.1.4 BOFF#

If BOFF# is asserted, the dual-processor pair immediately (in the next clock) floats the address, control, and data buses. Any bus cycles in progress are aborted, and any data returned to the processor in the clock in which BOFF# is asserted is ignored. In response to BOFF#, Primary and Dual processors float the same pins as when HOLD is active.

The Primary and Dual processors may reorder cycles after a BOFF#. The reordering occurs if there is inter-processor pipelining at the time of the BOFF#, but the system cannot change the cacheability of the cycles after the BOFF#. Note that there could be a change of bus ownership transparent to the system while the processors are in the backed-off state. Table 4-8 illustrates the flow of events which would result in cycle reordering due to BOFF#:

Table 4-8. Cycle Reordering Due to BOFF#

Time†	Processor A	System	Processor B
0	ADS# driven	--	--
1	--	NA# active	--
2	--	--	ADS# driven
3	Bus float	BOFF# active	Bus float
4	--	EADS# active	--
5	--	--	HITM# driven
6	--	BOFF# inactive	--
7	--	--	Write back ‘M’ data
8	--	BRDY#s	--
9	--	--	Restart ADS#
10	Restart ADS#	--	--

† Time is merely sequential, NOT measured in CLKs.

4.5.1.5 Bus Hold

The processor supports a bus hold/hold acknowledge protocol using the HOLD and HLDA signals. When the processor completes all outstanding bus cycles, it releases the bus by floating the external bus, and driving HLDA active. HLDA normally is driven two clocks after the later of the last BRDY# or HOLD being asserted, but may be up to six clocks due to active internal APIC cycles. Because of this, it is possible that an additional cycle may begin after HOLD is asserted but before HLDA is driven. Therefore, asserting HOLD does not prevent a dual-processor arbitration from occurring before HLDA is driven out. Even if an arbitration switch occurs, no new cycles are started after HOLD has been active for two clocks.

4.5.2 Dual Processing Power Management

4.5.2.1 STPCLK#

The Primary and Dual processor STPCLK# signals may be tied together or left separate. Refer to Chapter 12, “Power Management.” for more information on stop clock and Autohalt.

4.5.2.2 System Management Mode

The embedded Pentium processor supports system management mode (SMM) with a processor inserted in the upgrade socket. SMM provides a means to implement power management functions and operating system independent functions. SMM consists of an interrupt (SMI), an alternate address space and an instruction (RSM). SMM is entered by asserting the SMI# pin or delivering the SMI interrupt via the local APIC.

Although SMM functions the same when a Dual processor is inserted in Socket 5/Socket 7, the dual processor operation of the system must be carefully considered. The SMI# pins may be tied together or not, depending upon the power management features supported.

4.5.3 Other Dual-Processor Considerations

4.5.3.1 Strong Write Ordering

The ordering of write cycles in the processor can be controlled with the EWBE# pin. During uniprocessor operation, the EWBE# pin is sampled by the processor with each BRDY# assertion during a write cycle. The processor stalls all subsequent write operations to E or M state lines if EWBE# is sampled inactive. If the EWBE# pin is sampled inactive, it continues to be sampled on every clock until it is found to be active.

In dual processing mode, each processor tracks EWBE# independently of bus ownership. EWBE# is sampled and handled independently between the two processors. Only the processor that owns the bus (MRM) samples EWBE#. Once sampled inactive, the processor stalls subsequent write operations.

4.5.3.2 Bus Snarfing

The dual processor pair does not support cache-to-cache transfers (bus snarfing). If a processor *PB* requires data that is modified in processor *PA*, processor *PA* writes the data back to memory. After *PA* has completed the data transfer, *PB* runs a read cycle to memory. Where *PA* is either the Primary or the Dual processor, and *PB* is the other processor.

4.5.3.3 Interrupts

A processor may need to arbitrate for the use of the bus as a result of an interrupt. However, from the simple arbitration model used by the embedded Pentium processor, an interrupt is not a special case. There is no interaction between dual-processor support and the interrupt model in the embedded Pentium processor.

4.5.3.4 INIT Sequences

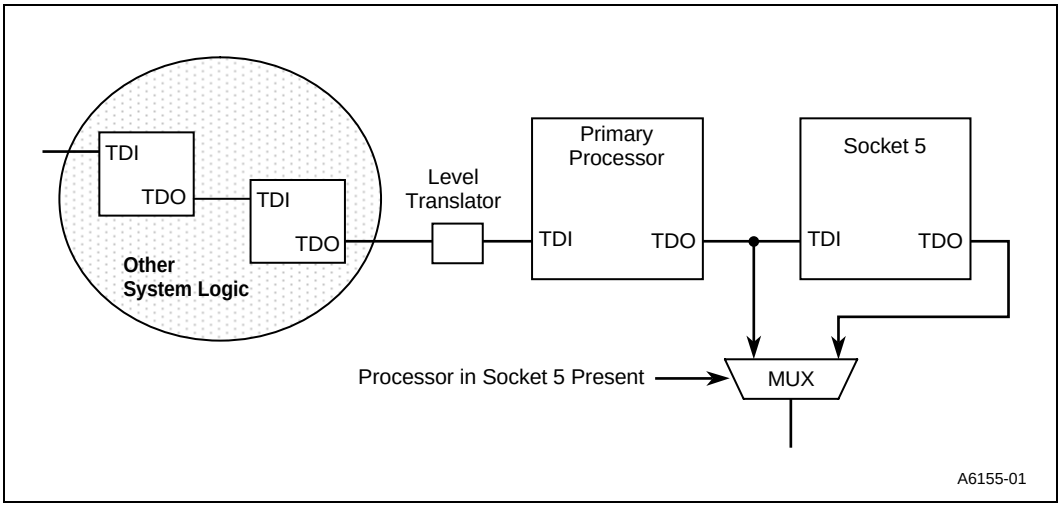
The INIT operation in dual-processor mode is exactly the same as in uni-processor mode. The two INIT pins must be tied together. However, in dual processor mode, the Primary processor must send an IPI and a starting vector to the Dual processor via the local APIC modules.

4.5.3.5 Boundary Scan

The embedded Pentium processor supports the full IEEE JTAG specification. The system designer is responsible to configure an upgrade ready system in such a way that the addition of a Dual processor in Socket 7 allows the boundary scan chain to functional as normal. This could be implemented with a jumper in Socket 7 that connects the TDI and TDO pins. The jumper would then be removed when the dual processor is inserted.

Alternatively, Socket 7 could be placed near the end of the boundary scan chain in the system. A multiplexer in the system boundary scan logic could switch between the TDO of the Primary and the dual processors as a Dual processor part is inserted. An illustration of this approach is shown in Figure 4-11.

Figure 4-11. Dual-Processor Boundary Scan Connections



4.5.3.6 Presence of a Processor in Socket 7

The Dual processor drives the DPEN# signal low during RESET to indicate to the Primary processor that a processor is present in Socket 7. The processor samples this line during RESETs falling edge.

DPEN# shares a pin with the APIC PICD0 signal.

4.5.3.7 MRM Processor Indication

In a dual-processor system, the D/P# (Dual processor/Primary processor Indication) signal indicates which processor is running a cycle on the bus. Table 4-9 shows how the external hardware can determine which processor is the MRM.

Table 4-9. Using D/P# to Determine MRM

D/P#	Bus Owner
0	Primary processor is MRM
1	Dual processor is MRM

D/P# can be sampled by the system with ADS# to determine which processor is driving the cycle on the bus. D/P# is driven only by the processor when operating as the Primary processor. Because of this, this signal is never driven by the Dual processor.

4.5.4 Dual-Processor Pin Functions

All the inputs pins are sampled with bus clock or test clock, and therefore, must meet setup and hold times with respect to the rising edge of the appropriate clock. In the dual-processor configuration, the RESET and FLUSH# pins have been changed to be synchronous (i.e., to meet setup and hold times). There have been no changes to the other existing input pins.

If the FLUSH# signal is deasserted before the corresponding FLUSH ACK cycle, the FLUSH# signal must not be asserted again until the FLUSH ACK cycle is generated. This requirement does not apply to a uni-processor system. In a dual processor system, a single FLUSH ACK cycle is generated after the caches in both processors have been flushed.

All system output pins are driven from the rising edge of the bus clock and meet maximum and minimum valid delays with respect to the bus clock. TDO is driven with respect to the rising edge of TCK and PICD0–PICD1 are driven with respect to the rising edge of PICCLK.

Table 4-10 summarizes the functional changes of all the pins in dual-processor mode.

Table 4-10. Dual-Processor Pin Functions vs. Pentium® Processor (Sheet 1 of 4)

Pin Name	I/O	Load (Note 1)	Same? (Note 2)	Tied Together? (Note 3)	Comments
A31–A3	I/O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states these signals for one CLK.
A20M#	I	Y	Y	Yes	Used in virtual mode and possibly in real mode by DOS and DOS extenders. Internally masked by the Dual processor.
ADS#, ADSC#	I/O O	Y	N	Yes	ADS# and ADSC# are three-stated by the LRM processor in order to allow the MRM processor to begin driving them. There are no system implications.
AHOLD	I	Y	Y	Yes	
AP	I/O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states this signal for one CLK.

NOTES:
1. “Load” indicates whether the pin would introduce a capacitive load to the system board due to the presence of the dual processor.
2. “N” indicates that there is a minor functional change to the pin(s) either as an enhancement to the embedded Pentium processor or due to dual processor operation.
3. “Yes” means that both processors must see the same value on the pin(s) for proper dual-processor operation. “No” means that the system must provide the signal to each processor independently. “May be” means that the system designer can choose to provide the signal to both processors or provide independent signals to each processor.

Table 4-10. Dual-Processor Pin Functions vs. Pentium® Processor (Sheet 2 of 4)

Pin Name	I/O	Load (Note 1)	Same? (Note 2)	Tied Together? (Note 3)	Comments
APCHK#	O	N	Y	No	Requires a system OR function.
BE7–BE5# BE4# –BE0#	O I/O	Y Y	N N	Yes Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states these signals for one CLK. BE3#–BE0# are used by the local APIC modules to load the APIC_ID at RESET. BE3#–BE0# will be three-stated by the Primary and Dual processors during RESET.
BF	I	Y	n/a	Yes	
BOFF#	I	Y	Y	Yes	
BP3–BP0	O	N	N	No	BP3–BP0 now only indicates breakpoint match in the I/O clock. Each processor must have different breakpoints. Note that BP1–BP0 are muxed with PM1–PM0.
BRDY#, BRDYC#	I	Y	Y	Yes	
BREQ	O	Y	N	Yes	The MRM drives this signal as a combined bus cycle request for itself and the LRM.
BUSCHK#	I	Y	Y	Yes	
CACHE#	I/O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states this signal for one CLK.
CLK	I	Y	Y	Yes	Both processors must use the same system clock.
CPUTYP	I	Y	n/a	No	
D/C#	I/O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states this signal for one CLK.
D/P#	O	n/a	n/a	No	The Primary processor always drives this signal. This output is not defined on the Dual processor.
D63–D0	I/O	Y	Y	Yes	
DP7–DP0	I/O	Y	Y	Yes	
EADS#	I	Y	Y	Yes	
EWBE#	I	Y	Y	Yes	This signal is sampled active with BRDY#, but inactive asynchronously. For optimized performance (minimum number of write E/M stalls) the chip set/platform should allow a dead clock between buffer going empty to buffer going full. This allows this signal to be completely independent between the two processors, rather than having one stall internal cache writes due to the other filling the external buffer.
FERR#	O	Y	Y	Yes	Used for DOS floating-point compatibility. The Primary processor drives this signal. The Dual processor never drives this signal.

NOTES:
1. “Load” indicates whether the pin would introduce a capacitive load to the system board due to the presence of the dual processor.
2. “N” indicates that there is a minor functional change to the pin(s) either as an enhancement to the embedded Pentium processor or due to dual processor operation.
3. “Yes” means that both processors must see the same value on the pin(s) for proper dual-processor operation. “No” means that the system must provide the signal to each processor independently. “May be” means that the system designer can choose to provide the signal to both processors or provide independent signals to each processor.

Table 4-10. Dual-Processor Pin Functions vs. Pentium® Processor (Sheet 3 of 4)

Pin Name	I/O	Load (Note 1)	Same? (Note 2)	Tied Together? (Note 3)	Comments
FLUSH#	I	Y	Y	Yes	In a dual-processor system, the flush operation is atomic with a single flush acknowledge bus cycle. Therefore, FLUSH# must not be re-asserted until the corresponding FLUSH ACK cycle is generated.
FRCMC#	I	N	Y	Yes	Both processors must be in Master mode. A processor in the Socket 7 cannot be used as a Checker.
HIT#	I/O	Y	N	Yes	This signal is asserted by the MRM based on the combined outcome of the inquire cycle between the two processors.
HITM#	I/O	Y	N	Yes	See HIT#.
HLDA	I/O	Y	N	Yes	Driven by the MRM.
HOLD	I	Y	Y	Yes	
IERR#	O	N	Y	No	
IGNNE#	I	Y	Y	Yes	The Dual processor ignores this signal.
INIT	I	N	N	Yes	In dual-processor mode, the Dual processor requires an IPI during initialization.
INTR/LINT0	I	N	N	May be	If the APIC is enabled, this pin is a local interrupt. If the APIC is hardware disabled, this pin function is not changed.
INV	I	Y	Y	Yes	
KEN#	I	Y	Y	Yes	
LOCK#	I/O	Y	N	Yes	The LRM samples the value of LOCK#, and drives the sampled value in the clock in which it gets ownership of the dual-processor bus. If sampled active, then the LRM keeps driving the LOCK# signal until ownership changes again.
M/IO#	I/O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states this signal for one CLK.
NA#	I	Y	Y	Yes	
NC	n/a	N	Y	No	
NMI/LINT1	I	N	Y	May be	If the APIC is enabled, then this pin is a local interrupt. If the APIC is hardware disabled, this pin function is not changed.
PBGNT#	I/O	n/a	n/a	Yes	This signal is always driven by one of the processors.
PBREQ#	I/O	n/a	n/a	Yes	This signal is always driven by one of the processors.
PCD	O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states this signal for one CLK.

NOTES:

1. “Load” indicates whether the pin would introduce a capacitive load to the system board due to the presence of the dual processor.
2. “N” indicates that there is a minor functional change to the pin(s) either as an enhancement to the embedded Pentium processor or due to dual processor operation.
3. “Yes” means that both processors must see the same value on the pin(s) for proper dual-processor operation. “No” means that the system must provide the signal to each processor independently. “May be” means that the system designer can choose to provide the signal to both processors or provide independent signals to each processor.

Table 4-10. Dual-Processor Pin Functions vs. Pentium® Processor (Sheet 4 of 4)

Pin Name	I/O	Load (Note 1)	Same? (Note 2)	Tied Together? (Note 3)	Comments
PCHK#	O	N	Y	May be	May be wire-ANDed together in the system, tied together, or the chip set may have two PCHK# inputs for dual-processor data parity.
PEN#	I	Y	Y	Yes	
PHIT#	I/O	n/a	n/a	Yes	This signal is always driven by one of the processors.
PHITM#	I/O	n/a	n/a	Yes	This signal is always driven by one of the processors.
PHITM#	I/O	n/a	n/a	Yes	This signal is always driven by one of the processors.
PICCLK	I	Y	n/a	Yes	
PICD1–PICD0	I/O	Y	n/a	Yes	
PM1–PM0	O	N	N	No	Each processor may track different performance monitoring events. Note that PM1–PM0 are mux'd with BP1–BP0.
PRDY	O	N	Y	No	
PWT	O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states this signal for one CLK.
R/S#	I	N	Y	No	
RESET	I	Y	Y	Yes	In dual-processor mode, RESET must be synchronous to the processor CLK that goes to the Primary and Dual processors.
SCYC	I/O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states this signal for one CLK.
SMI#	I	N	Y	May be	Refer to Chapter 12.
SMIACT#	O	N	Y	Yes	Refer to Chapter 12.
STPCLK#	I	n/a	n/a	May be	Refer to Chapter 12.
TCK	I	n/a	n/a	May be	System dependent
TDI	I	n/a	n/a	No	System dependent
TDO	O	n/a	n/a	No	System dependent
TMS	I	n/a	n/a	May be	System dependent
TRST#	I	n/a	n/a	May be	System dependent
V_{CC}	I	N	N	Yes	V_{CC} on the processor must be connected to 3.3 V.
V_{SS}	I	N	Y	Yes	
W/R#	I/O	Y	N	Yes	When the MRM becomes the LRM (and issues PBGNT#), it three-states this signal for one CLK.
WB/WT#	I	Y	Y	Yes	

NOTES:
1. "Load" indicates whether the pin would introduce a capacitive load to the system board due to the presence of the dual processor.
2. "N" indicates that there is a minor functional change to the pin(s) either as an enhancement to the embedded Pentium processor or due to dual processor operation.
3. "Yes" means that both processors must see the same value on the pin(s) for proper dual-processor operation. "No" means that the system must provide the signal to each processor independently. "May be" means that the system designer can choose to provide the signal to both processors or provide independent signals to each processor.

Hardware Interface 5

5.1 Detailed Pin Descriptions

This chapter describes the embedded Pentium® processor pins that interface to the system. Both the embedded Pentium processor and the embedded Pentium processor with MMX technology have the same logical hardware interface. The embedded Pentium processor with MMX technology has one extra signal, VCC2DET#.

The processor, when operating in dual processing mode, modifies the functionality of the following signals:

- A20M#, ADS#, BE4#–BE0#, CACHE#, D/C#, FERR#, FLUSH#, HIT#, HITM#, HLDA, IGNNE#, LOCK#, M/IO#, PCHK#, RESET, SCYC, SMIACT#, W/R#

5.1.1 A20M#

A20M#	Address 20 Mask
	Used to emulate the 1 Mbyte address wraparound on the 8086
	Asynchronous Input

Signal Description

When the address 20 mask (A20M#) input 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. A20M# is provided to emulate the address wraparound at 1 Mbyte which occurs on the 8086.

Note: A20M# must be asserted only when the processor is in real mode. *The effect of asserting A20M# in protected mode is undefined and may be implemented differently in future processors.*

Inquire cycles and writebacks caused by inquire cycles are not affected by this input. Address bit A20 is not masked when an external address is driven into the processor for an inquire cycle. Note that if an OUT instruction is used to modify A20M#, this does not affect previously prefetched instructions. A serializing instruction must be executed to guarantee recognition of A20M# before a specific instruction.

The processor, when configured as a Dual processor, ignores the A20M# input.

When Sampled/Driven

A20M# is sampled on every rising clock edge. A20M# is level sensitive and active low. This pin is asynchronous, but must meet setup and hold times for recognition in any specific clock. To guarantee that A20M# will be recognized before the first ADS# after RESET, A20M# must be asserted within two clocks after the falling edge of RESET

Note: As the performance of embedded Pentium processors continues to improve, code sequences are executed faster. As a result, some code sequences that rely upon hardware timing may fail. Specifically when a keyboard controller is used to toggle the A20M# pin and the keyboard

controller is slow in response, data or code may be read from a wrong address at some point in a code sequence. Therefore, you should ensure that the keyboard controller switches the A20M# signal fast enough to match the execution speed of the processor. Software should be written to synchronize code execution with the toggling of the A20M# signal.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A20	When asserted, A20M# masks the value of address pin A20.
CPUTYP	When strapped to V_{CC}, the processor ignores the A20M# input.

5.1.2 A31–A3

A31–A3	Address Lines
	Defines the physical area of memory or I/O accessed.
	Input/Output

Signal Description

As outputs, the Address Lines (A31–A3) along with the byte enable signals (BE7#–BE0#) form the address bus and define the physical area of memory or I/O accessed.

The embedded Pentium processor is capable of addressing 4 gigabytes of physical memory space and 64 Kbytes of I/O address space.

As inputs, the address bus lines A31–A5 are used to drive addresses back into the processor to perform inquire cycles. Since inquire cycles affect an entire 32-byte line, the logic values of A4 and A3 are not used for the hit/miss decision, however A4 and A3 must be at valid logic level and meet setup and hold times during inquire cycles.

When Sampled/Driven

When an output, the address is driven in the same clock as ADS#. The address remains valid from the clock in which ADS# is asserted until the earlier of the last BRDY# or the clock after NA#, or until AHOLD is asserted.

When an input, the address must be returned to the processor to meet setup and hold times in the clock in which EADS# is sampled asserted.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A20M#	When asserted, A20M# causes address pin A20 to be masked.
ADS#	A31–A3 are driven with ADS# (except when a external inquire cycle causes a writeback before AHOLD is deasserted, see Chapter 6, “Bus Functional Description”).
AHOLD	A31–A3 are floated one clock after AHOLD is asserted.
AP	Even address parity is driven/sampled with the address bus on AP.
APCHK#	The status of the address parity check is driven on the APCHK# pin.
BE7#–BE0#	Completes the definition of the physical area of memory or I/O accessed.
BOFF#	A31–A3 are floated one clock after BOFF# is asserted.
EADS#	A31–A5 are sampled with EADS# during inquire cycles.
HIT#	HIT# is driven to indicate whether the inquire address driven on A31–A5 is valid in an internal cache.
HITM#	HITM# is driven to indicate whether the inquire address driven on A31–A5 is in the modified state in the data cache.
HLDA	A31–A3 are floated when HLDA is asserted.
INV	INV determines whether the inquire address driven to the processor on A31–A5 should be invalidated or marked as shared if it is valid in an internal cache.

5.1.3 ADS#

ADS#	Address Strobe
	Indication that a new valid bus cycle is currently being driven by the processor.
	Synchronous Input/Output

Signal Description

The Address Strobe output indicates that a new valid bus cycle is currently being driven by the processor. The following pins are driven to their valid level in the clock ADS# is asserted: A31–A3, AP, BE7#–BE0#, CACHE#, LOCK#, M/IO#, W/R#, D/C#, SCYC, PWT, PCD.

ADS# is used by external bus circuitry as the indication that the processor has started a bus cycle. The external system may sample the bus cycle definition pins on the next rising edge of the clock after ADS# is driven active.

ADS# floats during bus HOLD and BOFF#. ADS# is not driven low to begin a bus cycle while AHOLD is asserted unless the cycle is a writeback due to an external invalidation. An active (floating low) ADS# in the clock after BOFF# is asserted should be ignored by the system.

This signal is normally identical to the ADSC# output. When operating in dual processing mode, the processor uses this signal for private snooping.

When Sampled/Driven

ADS# is driven active in the first clock of a bus cycle and is driven inactive in the second and subsequent clocks of the cycle. ADS# is driven inactive when the bus is idle.

This signal becomes an Input/Output when two embedded Pentium processors are operating together in Dual Processing Mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADSC#	ADS# is identical to the ADSC# output.
APCHK#	When operating in dual processing mode, APCHK# is driven in response to ADS# for a private snoop.
D/P#	When operating in dual processing mode, D/P# should be sampled with an active ADS#.
SMIACT#	When operating in dual processing mode, SMIACT# should be sampled with an active ADS# and qualified by D/P#.

5.1.4 ADSC#

ADSC#	Additional Address Strobe
	Indicates that a new valid bus cycle is currently being driven by the processor.
	Synchronous Output

Signal Description

This signal is identical to the ADS# output. This signal can be used to relieve tight board timings by easing the load on the Address Strobe signal.

When Sampled/Driven

Refer to the ADS# signal description.

Note: ADSC# is not tested and timings are not specified. It is recommended that ADSC# not be used.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	ADSC# is identical to the ADS# output.

5.1.5 AHOLD

AHOLD	Address Hold
	Floats the address bus so an inquire cycle can be driven to the processor.
	Synchronous Input

Signal Description

In response to the Address Hold request input, the processor stops driving A31–A3 and AP in the next clock. This pin is intended to be used for running inquire cycles to the processor. AHOLD allows another bus master to drive the processor address bus with the address for an inquire cycle. Since inquire cycles affect the entire cache line, although A31–A3 are floated during AHOLD, only A31–A5 are used by the processor for inquire cycles (and parity checking). Address pins 3 and 4 are logically ignored during inquire cycles but must be at a valid logic level when sampled.

While AHOLD is active, the address bus is floated, but the remainder of the bus can remain active. For example, data can be returned for a previously driven bus cycle when AHOLD is active. In general, the processor does not issue a bus cycle (ADS#) while AHOLD is active; the only exception to this is that writeback cycles due to an external snoop are driven while AHOLD is asserted.

Since the processor floats its bus immediately (in the next clock) in response to AHOLD, an address hold acknowledge is not required.

When AHOLD is deasserted, the processor drives the address bus in the next clock. It is the responsibility of the system designer to prevent address bus contention. This can be accomplished by ensuring that other bus masters have stopped driving the address bus before AHOLD is deasserted. Note the restrictions to the deassertion of AHOLD discussed in the inquire cycle section of the Chapter 6, “Bus Functional Description.”

AHOLD is recognized during RESET and INIT. Note that the internal caches are flushed as a result of RESET, so invalidation cycles run during RESET are unnecessary.

When Sampled

AHOLD is sampled on every rising clock edge, including during RESET and INIT.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A3	A31–A3 are floated as a result of the assertion of AHOLD.
ADS#	ADS# is not driven if AHOLD is asserted (except when a external inquire cycle causes a writeback before AHOLD is deasserted). See Chapter 6, “Bus Functional Description.”
AP	AP is floated as a result of the assertion of AHOLD.
EADS#	EADS# is recognized while AHOLD is asserted.

5.1.6 AP

AP	Address Parity
	Bidirectional address parity pin for the address lines of processor.
	Input/Output

Signal Description

This is the bidirectional Address Parity pin for the address lines of processor. There is one address parity pin for the address lines A31–A5. Note that A4 and A3 are not included in the parity determination.

When an output, AP is driven by the processor with even parity information on all processor generated cycles in the same clock as the address driven. (Even address parity means that there are an even number of HIGH outputs on A31–A5 and the AP pins.)

When an input, even parity information must be returned to the processor on this pin during inquire cycles in the same clock in which EADS# is sampled asserted to ensure that the correct parity check status is driven on the APCHK# output.

The value read on the AP pin does not affect program execution. The value returned on the AP pin is used only to determine even parity and drive the APCHK# output with the proper value. It is the responsibility of the system to take appropriate actions if a parity error occurs. If parity checks are not implemented in the system, AP may be connected to V_{CC} through a pull-up resistor and the APCHK# pin may be ignored.

When Sampled/Driven

When an output, AP is driven by the processor with even parity information on all processor generated cycles in the same clock as the address driven. The AP output remains valid from the clock in which ADS# is asserted until the earlier of the last BRDY# or the clock after NA#, or until AHOLD is asserted.

When an input, even parity information must be returned to the processor on this pin during inquire cycles in the same clock that EADS# is sampled asserted to guarantee that the proper value is driven on APCHK#. The AP input must be at a valid level and meet setup and hold times when sampled.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A5	The AP pin is used to create even parity with the A31–A5 pins.
ADS#	AP is driven with ADS# (except when a external inquire cycle causes a write-back before AHOLD is deasserted, see Chapter 6, “Bus Functional Description”).
AHOLD	AP is floated one clock after AHOLD is asserted.
APCHK#	The status of the address parity check is driven on the APCHK# output.
BOFF#	AP is floated one clock after BOFF# is asserted.
EADS#	AP is sampled with EADS# during inquire cycles.
HLDA	AP is floated when HLDA is asserted.

5.1.7 APCHK#

APCHK#	Address Parity Check
	The status of the address parity check is driven on this output.
	Asynchronous Output

Signal Description

APCHK# is asserted two clocks after EADS# is sampled active if the processor detects a parity error on the A31–A5 during inquire cycles.

Driving APCHK# is the only effect that bad address parity has on the processor. It is the responsibility of the system to take appropriate action if a parity error occurs. If parity checks are not implemented in the system, the APCHK# pin may be ignored.

Address parity is checked during every private snoop between the Primary and Dual processors. Therefore, APCHK# may be asserted due to an address parity error during this private snoop. If an error is detected, APCHK# will be asserted two clocks after ADS# for one processor clock period. The system can choose to acknowledge this parity error indication at this time or do nothing.

When Sampled/Driven

APCHK# is valid for one clock and should be sampled two clocks following ADS# and EADS# assertion. At all other times it is inactive (high). APCHK# is not floated with AHOLD, HOLD, or BOFF#. The APCHK# signal is glitch-free.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	When operating in dual processing mode, APCHK# is driven in response to a private snoop.
AP	Even address parity with the A31–A5 should be returned to the processor on the AP pin. If even parity is not driven, the APCHK# pin is asserted.
A31–A5	The AP pin is used to create even parity with A31–A5. If even parity is not driven to the processor, the APCHK# pin is asserted.
EADS#	APCHK# is driven in response to an external snoop.

5.1.8 APICEN

APICEN	APIC Enable
	This pin enables the APIC on the processor.
	Synchronous Configuration Input
	Needs external pull-up resistors.

Signal Description

APICEN, if sampled high at the falling edge of RESET, enables the on-chip APIC. If it is sampled low, then the on-chip APIC is not enabled and the processor uses the interrupts as if the APIC was not present (Bypass mode).

APICEN must be driven by the system. This pin has an internal pulldown resistor and is sampled at the falling edge of RESET. When using an active circuit to override the internal pulldown resistor, the driver should have an internal effective pullup resistance of 1 KOhms or less.

When Sampled/Driven

APICEN should be valid and stable two clocks before and after the falling edge of RESET.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BE3#–BE0#	When APICEN is sampled active, BE3#–BE0# are used to sample the APIC ID.
INTR/LINT0	When APICEN is sampled active, this input becomes the APIC local interrupt 0.
NMI/LINT1	When APICEN is sampled active, this input becomes the APIC local interrupt 1.
PICCLK	PICCLK must be tied or driven high when APICEN is sampled low at the falling edge of RESET.
PICD1	APICEN shares a pin with PICD1.
RESET	APICEN is sampled at the falling edge of RESET.

5.1.9 BE7#–BE0#

BE7#–BE0#	Byte Enable Outputs / APIC ID Inputs
	When operating in dual processing mode, BE4# is used to transfer information between the Dual and Primary processors during the atomic Flush operation.
	At RESET, the BE3#–BE0# pins read the APIC ID bits for the processor.
	After RESET, these pins are byte enables and help define the physical area of memory to I/O accessed.
	BE4#: Synchronous Input/Output, Dual Processing Mode. BE3#–BE0#: Synchronous Configuration Inputs, during RESET. BE3#–BE0#: Synchronous Outputs, following RESET.

Signal Description

As outputs, the byte enable signals are used in conjunction with the address lines to provide physical memory and I/O port addresses. The byte enables are used to determine which bytes of data must be written to external memory, or which bytes were requested by the processor for the current cycle.

- BE7# applies to D63–D56
- BE6# applies to D55–D48
- BE5# applies to D47–D40
- BE4# applies to D39–D32
- BE3# applies to D31–D24
- BE2# applies to D23–D16
- BE1# applies to D15–D8
- BE0# applies to D7–D0

In the case of cacheable reads (line fill cycles), all 8 bytes of data must be driven to the processor regardless of the state of the byte enables. If the requested read cycle is a single transfer cycle, valid data must be returned on the data lines corresponding to the active byte enables. Data lines corresponding to inactive byte enables need not be driven with valid logic levels. Even data parity is checked and driven only on the data bytes that are enabled by the byte enables.

The local APIC module on the embedded Pentium processor loads its 4-bit APIC ID value from the four least significant byte-enable pins at the falling edge of RESET. The following table shows the four pins that comprise the APIC ID.

APIC ID Register Bit	Pin Latched at RESET
bit 24	BE0#
bit 25	BE1#
bit 26	BE2#
bit 27	BE3#

Loading the APIC ID should be done with external logic that drives the proper address at reset. If the BE3#–BE0# signals are not driven, the APIC ID value defaults to 0000 for the embedded Pentium processor and 0001 for the Dual processor.

BE3#–BE0# pins establish the APIC ID for the processor and are input/output pins. These pins have strong internal pull down resistors and typically high external capacitive loading. A strong pullup on BE3#–BE0# is needed to make sure that the pins reach the correct value. In addition, since these pins are also outputs, a large resistive load would degrade the signal output during normal operation. A 50-Ohm three-state driver is recommended to drive these pins during RESET only.

Warning: An APIC ID of all 1s is an APIC special case (i.e., a broadcast) and must not be used. Because the Dual processor inverts the lowest order bit of the APIC ID placed on the lowest four BE pins, the value “1110” must not be used when operating in Dual Processing mode.

In a dual-processor configuration, the OEM socket and Socket 5/Socket 7 should have the four byte enable pairs tied together. The Primary processor loads the value seen on these four pins at RESET. The Dual processor loads the value seen on these pins and automatically inverts bit 24 of the APIC ID register. Thus, the two processors will have unique APIC ID values.

The Primary and Dual processors incorporate a mechanism to present an atomic view of the cache flush operation to the system when in dual processing mode. The Dual processor performs the cache flush operation and grants the bus to the Primary processor by PBREQ#/PBGNT# arbitration exchange. The Primary processor then flushes both of its internal caches and runs a cache flush acknowledge special cycle by asserting BE4#, to indicate to the external system that the cache line entries have been invalidated. The Dual processor halts all code execution while the processor is flushing its caches, and does not begin executing code until it recognizes the flush acknowledge special cycle. Refer to Chapter 6, “Bus Functional Description.”

When Sampled/Driven

As outputs, the byte enables are driven in the same clock as ADS#. The byte enables are driven with the same timing as the address bus (A31–A3). The byte enables remain valid from the clock in which ADS# is asserted until the earlier of the last BRDY# or the clock after NA#. The byte enables do not float with AHOLD.

The four least significant byte-enable bits are sampled for APIC ID at the falling edge of RESET. These pins should be valid and stable two clocks before and after the falling edge of RESET.

Note: Asserting the APIC ID is not specified for the rising edge of RESET. In a FRC system, the BE3#–BE0# pins must not be driven for the two clocks following the rising edge of RESET. The system design should drive these signals on the third clock or later.

There are strong pull down resistors on the byte enable pins internally that make it impractical to use pullup circuits to drive the APIC ID (on BE3#–BE0#) or enter Lock Step operation (with BE4#) at the falling edge of RESET. When not using the internal defaults on these pins, the value of the external pullup resistors would have to be 50 Ohms or less. For this reason it is suggested to use active drivers on these lines that would drive the byte enable pins during the falling edge of RESET. Passive pullups should be avoided.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A3	A31–A3 and BE7#–BE0# together define the physical area of memory or I/O accessed.
ADS#	BE7#–BE0# are driven with ADS#.
APICEN	When APICEN is sampled active, BE3#–BE0# are used to sample the APIC ID.
BOFF#	BE7#–BE0# are floated one clock after BOFF# is asserted.
D63–D0	BE7#–BE0# indicate which data bytes are being requested or driven by the processor.
DP7–DP0	Even data parity is checked/driven only on the data bytes enabled by BE7#–BE0#.
HLDA	BE7#–BE0# are floated when HLDA is asserted.
RESET	During reset the BE3#–BE0# pins are sampled to determine the APIC ID. Following RESET, they function as byte-enable outputs.

5.1.10 BF2–BF0

BF2–BF0	Bus-to-core frequency ratio
	Used to configure processor bus-to-core frequency ratio.
	Asynchronous Input
	Only the Low-power Embedded Pentium Processor with MMX technology has a BF2 pin.

Signal Description

The BF*n* pins determine whether the processor operates at a 1/2, 2/3, 2/5, 2/7 or 1/4 I/O bus-to-core frequency ratio. Since some bus-to-core ratios are not supported, these pins should *always* be connected to the proper level.

Note: External pulldowns of 500 Ohms or less must be used between the pins and ground to effectively override the default (internal) pullups, while external pullups of 2.2 KOhms or less should be used to override the default pulldowns on BF1–BF0.

Each embedded Pentium processor is specified to operate within a single bus-to-core ratio and a specific minimum to maximum bus frequency range (corresponding to a minimum to maximum core frequency range). Operation in other bus-to-core ratios or outside the specified operating frequency range is not supported. Tables 5-1 through 5-3 summarize these specifications.

Table 5-1. Bus-to-Core Frequency Ratios for the Embedded Pentium® Processor (at 100/133/166 MHz)

BF1	BF0	Embedded Pentium® Processor Bus/Core Ratio	Max Bus/Core Frequency (MHz)	Min Bus/Core Frequency (MHz)
0	0	2/5	66/166	33/83
1	0	1/2	66/133	33/66
1	1	2/3†	66/100	33/50

† This is the default bus fraction for the embedded Pentium processor (at 100/133/166 MHz). If the BF pins are left floating, the processor will be configured for the 2/3 bus to core frequency ratio.

Table 5-2. Bus-to-Core Frequency Ratios for the Embedded Pentium® Processor with MMX™ Technology

BF1	BF0	Embedded Pentium Processor with MMX™ Technology Bus/Core Ratio	Max Bus/Core Frequency (MHz)	Min Bus/Core Frequency (MHz)
1	1	2/7	66/233	33/117
0	1	1/3	66/200	33/100
1	0	1/2†	N/A	N/A

† This is the default bus-to-core ratio for the Pentium processor with MMX technology. If the BF pins are left floating, the processor will be configured for the 1/2 bus-to-core frequency ratio, which is unsupported. *Do not float the BFn pins at RESET.*

Table 5-3. Bus-to-Core Frequency Ratios for the Low-Power Embedded Pentium® Processor with MMX™ Technology

BF2	BF1	BF0	Low-Power Embedded Pentium Processor with MMX™ Technology Bus/Core Ratio	Max Bus/Core Frequency (MHz)	Min Bus/Core Frequency (MHz)
0	0	0	2/5	66/166	
1	0	0	1/4	66/266	

If BF1–BF0 are left unconnected on the embedded Pentium processor with MMX technology, the bus-to-core ratio defaults to 1/2. If BF1–BF0 are left unconnected on the embedded Pentium processor the bus-to-core ratio defaults to 2/3. If BF2–BF0 are left unconnected on the low-power embedded Pentium processor with MMX technology, the bus-to-core ratio defaults to 2/5. This ratio is not supported by the low-power embedded Pentium processor with MMX technology; do *not* float the BF*n* pins when using the low-power embedded Pentium processor with MMX technology.

When Sampled/Driven

The BF*n* pins are sampled at RESET and cannot be changed until another non-warm (1 ms) assertion of RESET. BF*n* must not change values while RESET is active.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
RESET	BF2–BF0 are sampled at the falling edge of RESET.

5.1.11 BOFF#

BOFF#	Backoff
	The back off input is used to force the processor off the bus in the next clock.
	Synchronous Input

Signal Description

In response to BOFF#, the processor aborts all outstanding bus cycles that have not yet completed and floats the processor bus in the next clock. The processor floats all pins normally floated during bus hold. Note that since the bus is floated in the clock after BOFF# is asserted, an acknowledge is not necessary (HLDA is not asserted in response to BOFF#).

The processor remains in bus hold until BOFF# is negated, at which time the processor restarts any aborted bus cycle(s) in their entirety by driving out the address and status and asserting ADS#.

This pin can be used to resolve a deadlock situation between two bus masters.

Any data with BRDY# returned to the processor while BOFF# is asserted is ignored.

BOFF# has higher priority than BRDY#. If both BOFF# and BRDY# occur in the same clock, BOFF# takes effect.

BOFF# also has precedence over BUSCHK#. When BOFF# and BUSCHK# are both asserted during a bus cycle, BUSCHK# is ignored.

When Sampled

BOFF# is sampled on every rising clock edge, including when RESET and INIT are asserted.

When a read cycle is running on the bus and an internal snoop of that read cycle hits a modified line in the data cache, causing the system to assert BOFF#, the sequence of bus cycles is as follows: Upon negation of BOFF#, the processor drives out a writeback resulting from the internal snoop hit. After completion of the writeback, the processor then restarts the original read cycle. Thus, like external snoop writebacks, internal snoop writebacks may also be reordered in front of cycles that encounter a BOFF#. Also note that, although the original read encountered both an external BOFF# and an internal snoop hit to an M-state line, it is restarted only once.

This circumstance can occur during accesses to the page tables/directories and during prefetch cycles (these accesses cause a bus cycle to be generated before the internal snoop to the data cache is performed).

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A3–A31 ADS# AP BE7#–BE3# CACHE# D/C# D63–D0 DP7–DP0 LOCK# M/IO# PCD PWT SCYC W/R#	These signals float in response to BOFF#.
BRDY#	If BRDY# and BOFF# are asserted simultaneously, BOFF# takes priority and BRDY# is ignored.
EADS#	EADS# is recognized when BOFF# is asserted.
HLDA	The same pins are floated when HLDA or BOFF# is asserted.
BUSCHK#	If BUSCHK# and BOFF# are both asserted during a bus cycle, BOFF# takes priority and BUSCHK# is forgotten.
NA#	If NA# and BOFF# are asserted simultaneously, BOFF# takes priority and NA# is ignored.

5.1.12 BP3–BP0

BP3–BP0	Breakpoint signals
	BP3–BP0 externally indicate a breakpoint match.
	Synchronous Output

Signal Description

The Breakpoint pins (BP3–BP0) 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.

Because of the fractional-speed bus, each assertion of a processor BP pin indicates that one or more BP matches occurred. The maximum number of matches per assertion is two when using the 2/3 or 1/2 bus-to-core ratios. Similarly, the maximum number of matches per assertion is three when using the 2/5 or 1/3 bus-to-core ratios.

When Sampled/Driven

The BP3–BP0 pins are driven in every clock and are not floated during bus HOLD of BOFF#.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
PM1–PM0	BP1 and BP0 share pins with PM1 and PM0, respectively.

5.1.13 BRDY#

BRDY#	Burst Ready
	Transfer complete indication.
	Synchronous Input

Signal Description

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.

Each cycle generated by the processor is either a single transfer read (or write) or a burst cache line fill (or writeback). For single data transfer cycles, one BRDY# is expected to be returned to the processor. When this BRDY# is returned, the cycle is complete. For burst transfers, four data transfers are expected by the processor. The cycle is ended when the fourth BRDY# is returned.

When Sampled

This signal is sampled in the T2, T12 and T2P bus states.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BOFF#	If BOFF# and BRDY# are asserted simultaneously, BOFF# takes priority and BRDY# is ignored.
BUSCHK#	BUSCHK# is sampled with BRDY#.
CACHE#	In conjunction with the KEN# input, CACHE# determines whether the bus cycle consists of 1 or 4 transfers.
D63–D0	During reads, the D63–D0 pins are sampled by the processor with BRDY#. During writes, BRDY# indicates that the system has accepted D63–D0.
DP7–DP0	During reads, the DP7–DP0 pins are sampled by the processor with BRDY#. During writes, BRDY# indicates that the system has accepted DP7–DP0.
EWBE#	EWBE# is sampled with each BRDY# of a write cycle.
KEN#	KEN# is sampled and latched by the processor with the earlier of the first BRDY# or NA#. Also, in conjunction with the CACHE# input, KEN# determines whether the bus cycle will consist of 1 or 4 transfers (assertions of BRDY#).
LOCK#	LOCK# is deasserted after the last BRDY# of the locked sequence.
PCHK#	PCHK# indicates the results of the parity check two clocks after BRDY# is returned for reads.
PEN#	PEN# is sampled with BRDY# for read cycles.
WB/WT#	WB/WT# is sampled and latched by the processor with the earlier of the first BRDY# or NA#.

5.1.14 BRDYC#

BRDYC#	Burst Ready
	Transfer complete indication.
	Synchronous Input

Signal Description

This signal is identical to the BRDY# input. This signal can be used to relieve tight board timings by easing the load on the Burst Ready signal.

In addition to its normal functionality, BRDYC# is sampled with BUSCHK# at RESET to select the buffer strength for some pins. BRDYC# has an internal pullup resistor. To override the default settings for the buffer strengths, this pin should be driven and not permanently strapped to ground because this would interfere with the normal operation of this pin. The driver should have an internal resistance of 1 KOhms or less. This is a function only of BRDYC#. The BRDY# signal is not sampled to select buffer sizes.

When Sampled/Driven

Refer to the BRDY# signal description.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BRDY#	BRDYC# is identical to the BRDY# input.
RESET	BRDYC# and BUSCHK# are sampled at RESET to select the buffer strength for some pins.

5.1.15 BREQ

BREQ	Bus Request
	Indicates externally when a bus cycle is pending internally.
	Output

Signal Description

The processor asserts the BREQ output whenever a bus cycle is pending internally. BREQ is always asserted in the first clock of a bus cycle with ADS#. Furthermore, if the processor is not currently driving the bus (due to AHOLD, HOLD, or BOFF#), BREQ is asserted in the same clock that ADS# would have been asserted if the processor were driving the bus. After the first clock of the bus cycle, BREQ may change state. Every assertion of BREQ is not guaranteed to have a corresponding assertion of ADS#.

External logic can use the BREQ signal to arbitrate between multiple processors. This signal is always driven regardless of the state of AHOLD, HOLD or BOFF#.

When Driven

BREQ is always driven by the processor, and is not floated during bus HOLD or BOFF#.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	BREQ is always asserted in the clock in which ADS# is asserted.

5.1.16 BUSCHK#

BUSCHK#	Bus Check
	Allows the system to signal an unsuccessful completion of a bus cycle.
	Synchronous Input

Signal Description

The Bus Check input pin allows the system to signal an unsuccessful completion of a bus cycle. When this pin is sampled active, the processor latches the address and control signals of the failing cycle in the machine check registers. When the MCE bit in CR4 is also set, the processor vectors to the machine check exception upon completion of the current instruction.

If BUSCHK# is asserted in the middle of a cycle, the system must return all expected BRDY# signals to the processor. BUSCHK# is remembered by the processor if asserted during a bus cycle. The processor decides after the last BRDY# whether to take the machine check exception or not.

BOFF# has precedence over BUSCHK#. When BOFF# and BUSCHK# are both asserted during a bus cycle, BUSCHK# is ignored.

In addition to its normal functionality, BUSCHK# is sampled with BRDYC# at RESET to select the buffer strength for some pins. BUSCHK# has an internal pullup resistor. To override the default settings for the buffer strengths, this pin should be driven and not permanently strapped to ground, because this interferes with the normal operation of this pin. The driver should have an internal resistance of 1 KOhms or less.

When Sampled

BUSCHK# is sampled when BRDY# is returned to the processor.

Note: The embedded Pentium processor can remember only one machine check exception at a time. This exception is recognized on an instruction boundary. If BUSCHK# is sampled active while servicing the machine check exception for a previous BUSCHK#, it is remembered by the processor until the original machine check exception is completed. Then, the processor services the machine check exception for the second BUSCHK#. Note that only one BUSCHK# is remembered by the processor while the machine exception for the previous one is being serviced.

When the BUSCHK# is sampled active by the processor, the cycle address and cycle type information for the failing bus cycle is latched upon assertion of the last BRDY# of the bus cycle. The information is latched into the Machine Check Address (MCA) and Machine Check Type (MCT) registers respectively. However, if the BUSCHK# input is not deasserted before the first

BRDY# of the next bus cycle, and the machine check exception for the first bus cycle has not occurred, then new information is latched into the MCA and MCT registers, over-writing the previous information at the completion of this new bus cycle. Therefore, in order for the MCA and MCT registers to report the correct information for the failing bus cycle when the machine check exception for this cycle is taken at the next instruction boundary, the system must deassert the BUSCHK# input immediately after the completion of the failing bus cycle (i.e., before the first BRDY# of the next bus cycle is returned).

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BOFF#	If BOFF# and BUSCHK# are both asserted during a bus cycle, the BOFF# signal causes the BUSCHK# to be forgotten.
BRDY#	BUSCHK# is sampled with BRDY#.
BRDYC#	BUSCHK# is sampled with BRDYC# at RESET to select the buffer strength for some pins.
RESET	BUSCHK# and BRDYC# are sampled at RESET to select the buffer strength for some pins.

5.1.17 CACHE#

CACHE#	Cacheability
	External indication of internal cacheability.
	Synchronous Input/Output

Signal Description

The Cacheability output is a cycle definition pin. For processor initiated cycles, this pin indicates internal cacheability of the cycle (if a read), and indicates a burst writeback (if a write). CACHE# is asserted for cycles coming from the cache (writebacks) and for cycles that will go into the cache if KEN# is asserted (linefills). More specifically, CACHE# is asserted for cacheable reads, cacheable code fetches, and writebacks. It is driven inactive for non-cacheable reads, TLB replacements, locked cycles (except writeback cycles from an external snoop that interrupt a locked read/modify/write sequence), I/O cycles, special cycles and writethroughs.

For read cycles, the CACHE# pin indicates whether the processor allows the cycle to be cached. When CACHE# is asserted for a read cycle, the cycle is turned into a cache line fill if KEN# is returned active to the processor. When this pin is driven inactive during a read cycle, processor does not cache the returned data, regardless of the state of the KEN#.

If this pin is asserted for a write cycle, it indicates that the cycle is a burst writeback cycle. Writethroughs cause a non-burst write cycle to be driven to the bus. The processor does not support write allocations (cache line fills as a result of a write miss).

When operating in dual processing mode, the embedded Pentium processors use this signal for private snooping.

When Sampled/Driven

CACHE# is driven to its valid level in the same clock as the assertion of ADS# and remains valid until the earlier of the last BRDY# or the clock after NA#.

This signal becomes an Input/Output when two embedded Pentium processors are operating together in dual processing mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	CACHE# is driven to its valid level with ADS#.
BOFF#	CACHE# floats one clock after BOFF# is asserted.
BRDY#	In conjunction with the KEN# input, CACHE# determines whether the bus cycle will consist of 1 or 4 transfers (assertions of BRDY#).
HLDA	CACHE# floats when HLDA is asserted.
KEN#	KEN# and CACHE# are used together to determine if a read will be turned into a linefill.

5.1.18 CLK

CLK	Clock
	Fundamental timing source for the embedded Pentium processor.
	Input

Signal Description

The Clock input provides the fundamental timing source for the embedded Pentium processor. Its frequency is proportional to the internal operating frequency of the processor (as selected by the BF1–BF0 pins) and requires a TTL level signal. All external timing parameters except TDI, TDO, TMS, and TRST# are specified with respect to the rising edge of CLK.

Note: The CLK signal on the embedded Pentium processor with MMX technology is 3.3 V tolerant. On the embedded Pentium processor, the CLK input is 5.0 V tolerant.

When Sampled

CLK is used as a reference for sampling other signals. It is recommended that CLK begin toggling within 150 ms after V_{CC} reaches its proper operating level. This recommendation is made to ensure long term reliability of the device. V_{CC} specifications and clock duty cycle, stability, and frequency specifications must be met for 1 ms before the negation of RESET. If at any time during normal operation one of these specifications is violated, the power on RESET sequence must be repeated. This requirement is made to ensure proper operation of the phase locked loop circuitry on the clock input within the processor.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
All except TCK TDI TDO TMS TRST#	External timing parameters are measured from the rising edge of CLK for all signals except TDI, TDO, TMS, TCK, and TRST#.

5.1.19 CPUTYP

CPUTYP	Processor Type Definition Pin
	Used to configure the embedded Pentium processor as a Dual processor.
	Asynchronous Input

Signal Description

The CPUTYP pin is used to determine whether the embedded Pentium processor functions as a Primary or Dual processor. CPUTYP must be strapped to either V_{CC} or V_{SS}. When CPUTYP is strapped to V_{CC}, the embedded Pentium processor functions as a Dual processor. When CPUTYP is strapped to V_{SS} (or left unconnected), the embedded Pentium processor functions as a Primary processor. In a single socket system design, CPUTYP pin must be strapped to V_{SS}.

When Sampled/Driven

CPUTYP is sampled at RESET and cannot be changed until another non-warm (1 ms) assertion of RESET. CPUTYP must meet a 1 ms setup time to the falling edge of RESET. It is recommended that CPUTYP be strapped to V_{CC} or V_{SS}.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A20M#	When CPUTYP is strapped to V_{CC}, the processor ignores the A20M# input.
BE4#–BE0#	The BE3#–BE0# input values are sampled during RESET to determine the APIC ID. The Dual processor uses BE4# to indicate to the Primary processor that it has completed its cache flush operation. Refer to the BE4#–BE0# pin description.
D/P#	D/P# is driven by the processor only when the CPUTYP signal is strapped to V_{SS}.
DPEN#	When CPUTYP is strapped to V_{CC}, DPEN# is driven active to indicate that the second socket is occupied.
FERR#	When CPUTYP is strapped to V_{CC}, the FERR# output is undefined.
FLUSH#	When operating in dual processing mode, the FLUSH# inputs become Synchronous to the processor clock.
IGNNE#	When CPUTYP is strapped to V_{CC}, the processor ignores the IGNNE# input.
RESET	CPUTYP is sampled at the falling edge of RESET. When operating in dual processing mode, the RESET inputs become synchronous to the processor clock.

Note: It is common practice to put either a pullup or pulldown resistor on a net. If a pullup resistor is connected to the CPUTYP pin in order to operate in a Dual Processing mode, the value of this resistor must be 100 Ohms or less to override the internal pulldown. In the absence of an external pullup, the internal pulldown sufficiently pulls down the CPUTYP pin; therefore the pin can be left floating.

5.1.20 D/C#

D/C#	Data/Code
	Distinguishes a data access from a code access.
	Synchronous Input/Output

Signal Description

The Data/Code signal is one of the primary bus cycle definition pins. D/C# distinguishes between data (D/C# = 1) and code/special cycles (D/C# = 0).

When operating in dual processing mode, the embedded Pentium processor uses this signal for private snooping.

When Sampled/Driven

The D/C# pin is driven valid in the same clock as ADS# and the cycle address. It remains valid from the clock in which ADS# is asserted until the earlier of the last BRDY# or the clock after NA#.

This signal becomes an Input/Output when two embedded Pentium processors are operating together in Dual Processing Mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	D/C# is driven with ADS#.
BOFF#	D/C# floats one clock after BOFF# is asserted.
HLDA	D/C# floats when HLDA is asserted.

5.1.21 D63–D0

D63–D0	Data Lines
	Forms the 64-bit data bus.
	Input/Output

Signal Description

The bidirectional lines D63–D0 form the 64 data bus lines for the embedded Pentium 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 Sampled/Driven

When the processor is driving the data lines (during writes), they are driven during the T2, T12, or T2P clocks for that cycle.

During reads, the processor samples the data bus when BRDY# is returned.

D63–D0 are floated during T1, TD, and Ti states.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BE7#–BE0#	BE7#–BE0# indicate which data bytes are being requested or driven by the processor.
BOFF#	D63–D0 float one clock after BOFF# is asserted.
BRDY#	BRDY# indicates that the data bus transfer is complete.
DP7–DP0	Even data parity is driven/sampled with the data bus on DP7–DP0.
HLDA	D63–D0 float when HLDA is asserted.
PCHK#	The status of the data bus parity check is driven on PCHK#.
PEN#	Even data parity with D63–D0 should be returned to the processor on the DP pin. If a data parity error occurs, and PEN# is enabled, the cycle is latched and a machine check exception is taken if CR4.MCE = 1.

5.1.22 D/P#

D/P#	Dual Processor / Primary Processor
	Indicates whether the Dual processor or the Primary processor is driving the bus.
	Synchronous Output

Signal Description

The D/P# pin is driven low when the Primary processor is driving the bus. Otherwise, the Primary processor drives this pin high to indicate that the Dual processor owns the bus. The D/P# pin can be sampled for the current cycle with ADS#. This pin is defined only on the Primary processor. In a single socket system design, D/P# pin should be left NC.

When Sampled/Driven

The D/P# pin is always driven by the Primary processor and should be sampled with ADS# of the current cycle.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	D/P# is valid for the current cycle with ADS# (like a status pin).
CPUTYP	D/P# is driven by the processor when the CPUTYP signal is strapped to V_{SS} (or left unconnected).
SMIACT#	When operating in dual processing mode, D/P# qualifies the SMIACT# SMM indicator.

5.1.23 DP7–DP0

DP7–DP0	Data Parity
	Bidirectional data parity pins for the data bus.
	Input/Output

Signal Description

These are the bidirectional Data Parity pins for the processor. There is one parity pin for each byte of the data bus. For example, DP7 applies to D63–D56 and DP0 applies to D7–D0.

As outputs, the data parity pins are driven by the processor with even parity information for writes in the same clock as write data. Even parity means that there are an even number of HIGH logic values on the eight corresponding data bus pins and the parity pin.

As inputs, even parity information must be driven back to the processor on these pins in the same clock as the data to ensure that the correct parity check status is indicated by the processor.

The value read on the data parity pins does not affect program execution unless PEN# is also asserted. If PEN# is not asserted, the value returned on the DP pins is used only to determine even parity and drive the PCHK# output with the proper value. If PEN# is asserted when a parity error occurs, the cycle address and type are latched in the MCA and MCT registers. If in addition, the MCE bit in CR4 is set, a machine check exception is taken.

It is the responsibility of the system to take appropriate actions if a parity error occurs. If parity checks are not implemented in the system, the DP7–DP0 and PEN# pins should be tied to V_{CC} through a pullup resistor and the PCHK# pin may be ignored.

When Sampled/Driven

As outputs, the data parity pins are driven by the processor with even parity information in the same clock as write data. The parity remains valid until sampled by the assertion of BRDY# by the system.

As inputs, even parity information must be driven back to the processor on these pins in the same clock as the data to ensure that the correct parity check status is indicated by the processor. The data parity pins must be at a valid logic level and meet setup and hold times when sampled.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BE7#–BE0#	Even data parity is checked/driven only on the data bytes enabled by BE7#–BE0#.
BOFF#	DP7–DP0 are floated one clock after BOFF# is asserted.
BRDY#	DP7–DP0 are sampled with BRDY# for reads.
D63–D0	The DP7–DP0 pins are used to create even parity with D63–D0 on a byte by byte basis. DP7–DP0 are driven with D63–D0 for writes.
HLDA	DP7–DP0 are floated when HLDA is asserted.
PCHK#	The status of the data parity check is driven on the PCHK# output.
PEN#	The DP7–DP0 pins are used to create even parity with D63–D0. If even parity is not detected, and PEN# is enabled, the cycle address and type are latched. If in addition CR4.MCE = 1, the machine check exception is taken.

5.1.24 DPEN#

DPEN#	Second Socket Occupied
	Configuration signal which indicates that the second socket in a dual socket system is occupied.
	Synchronous Input (to the processor) Synchronous Output (from the processor, when configured as a Dual processor)

Signal Description

DPEN# is driven during RESET by the processor when configured as a Dual processor to indicate to the Primary processor in the first socket that there is a Dual processor present in the system.

This pin has an internal pullup resistor and is sampled at the falling edge of RESET. When using an active circuit to override the internal pullup resistor, the driver should have an internal effective pulldown resistance of 1 KOhms or less.

When Sampled/Driven

DPEN# is driven during RESET by the Dual processor, and sampled at the falling edge of RESET by the Primary processor. This pin becomes PICD0 following the falling edge of RESET. This pin should be valid and stable two clocks before and after the falling edge of RESET.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
CPUTYP	When CPUTYP is strapped to V_{CC}, DPEN# is driven active to indicate that the second socket is occupied.
RESET	DPEN# is valid during the falling edge of RESET.
PICD0	DPEN# shares a pin with PICD0.

5.1.25 EADS#

EADS#	External Address Strobe
	Signals the processor to run an inquire cycle with the address on the bus.
	Synchronous Input

Signal Description

The EADS# input indicates that a valid external address has been driven onto the processor address pins to be used for an inquire cycle. The address driven to the processor when EADS# is sampled asserted is checked with the current cache contents. The HIT# and HITM# signals are driven to indicate the result of the comparison. When the INV pin is returned active (high) to the processor in the same clock as EADS# is sampled asserted, an inquire hit will result in that line being invalidated. When the INV pin is returned inactive (low), an inquire hit will result in that line being marked Shared (S).

When Sampled

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

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A5	The inquire cycle address must be valid on A31–A5 when EADS# is sampled asserted.
A4–A3	These signals must be at a valid logic level when EADS# is sampled asserted.
AHOLD	EADS# is recognized while AHOLD is asserted.
AP	AP is sampled when EADS# is sampled asserted.
APCHK#	APCHK# is driven to its valid level two clocks after EADS# is sampled asserted.
BOFF#	EADS# is recognized while BOFF# is asserted.
HIT#	HIT# is driven to its valid level two clocks after EADS# is sampled asserted.
HITM#	HITM# is driven to its valid level two clocks after EADS# is sampled asserted.
HLDA	EADS# is recognized while HLDA is asserted.
INV	INV is sampled with EADS# to determine the final state of the cache line in the case of an inquire hit.

5.1.26 EWBE#

EWBE#	External Write Buffer Empty
	Provides the option of strong write ordering to the memory system.
	Synchronous Input

Signal Description

The External write Buffer Empty input, when inactive (high), indicates that a writethrough cycle is pending in the external system. When the processor generates a write (memory or I/O), and EWBE# is sampled inactive, the processor holds off all subsequent writes to all E or M-state lines until all writethrough cycles have completed, as indicated by EWBE# being active. In addition, if the processor has a write pending in a write buffer, the processor also holds off all subsequent writes to E- or M-state lines. This insures that writes are visible from outside the processor in the same order as they were generated by software.

When the processor serializes instruction execution through the use of a serializing instruction, it waits for the EWBE# pin to go active before fetching and executing the next instruction.

After the OUT or OUTS instructions are executed, the processor ensures that EWBE# has been sampled active before beginning to execute the next instruction. Note that the instruction may be prefetched if EWBE# is not active, but it does not execute until EWBE# is sampled active.

When Sampled

EWBE# is sampled with each BRDY# of a write cycle. If sampled deasserted, the processor repeatedly samples EWBE# in each clock until it is asserted. Once sampled asserted, the processor ignores EWBE# until the next BRDY# of a write cycle.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BRDY#	EWBE# is sampled with each BRDY# of a write cycle.
SMIACT#	SMIACT# is not asserted until EWBE# is asserted.

5.1.27 FERR#

FERR#	Floating-Point Error
	The floating-point error output is driven active when an unmasked floating-point error occurs.
	Synchronous Output

Signal Description

The Floating-Point Error output 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 DOS type floating-point error reporting.

In some cases, FERR# is asserted when the next floating-point instruction is encountered and in other cases it is asserted before the next floating-point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating-point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating-point instruction):

1. Stack fault, all invalid operation exceptions and denormal exceptions on: all transcendental instructions, FSCALE, FXTRACT, FPREM, FPREM(1), FBLD, FLD_extended, FRNDINT, and stack fault and invalid operation exceptions on Floating-Point arithmetic instructions with an integer operand (FIADD/FIMUL/FISUB/FIDIV, etc.).
2. All real stores (FST/FSTP), Floating-Point integer stores (FIST/FISTP) and BCD store (FBSTP) (true for all exception on stores except Precision Exception).

The following class of floating-point exceptions drive FERR# only after encountering the next floating-point instruction. Note that the embedded Pentium processor with MMX technology reports a pending floating-point exception (assert FERR#) upon encountering the next floating-point or MMX instruction.

1. Numeric underflow, overflow and precision exception on: Transcendental instructions, FSCALE, FXTRACT, FPREM, FPREM(1), FRNDINT, and Precision Exception on all types of stores to memory.
2. All exceptions on basic arithmetic instructions (FADD/FSUB/FMUL/FDIV/ FSQRT/FCOM/FUCOM...)

FERR# is deasserted when the FCLEX, FINIT, FSTENV, or FSAVE instructions are executed. In the event of a pending unmasked floating-point exception the FNINIT, FNCLEX, FNSTENV, FNSAVE, FNSTSW, FNSTCW, FNENI, FNDISI, and FNSETPM instructions assert the FERR# pin. Shortly after the assertion of the pin, an interrupt window is opened during which the processor samples and services interrupts, if any. If no interrupts are sampled within this window, the processor then executes these instructions with the pending unmasked exception. However, for the FNCLEX, FNINIT, FNSTENV, and FNSAVE instructions, the FERR# pin is deasserted to enable the execution of these instructions. For details please refer to the *Intel Architecture Software Developer's Manual*, Volume 1 (Chapter 7 and Appendix D).

This signal is undefined when the embedded Pentium processor is configured as a Dual processor.

When Sampled/Driven

FERR# is driven in every clock and is not floated during bus HOLD or BOFF#. The FERR# signal is glitch free.

The embedded Pentium processor, when configured as a Dual processor, does not drive this signal to valid levels.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
CPUTYP	When CPUTYP is strapped to V_{CC}, the FERR# output is undefined.

5.1.28 FLUSH#

FLUSH#	Cache Flush
	Writes all modified lines in the data cache back and flushes the code and data caches.
	Asynchronous Input (Normal, Uni-processor mode) Synchronous Input (Dual processor mode)

Signal Description

When asserted, the Cache Flush input forces the processor to writeback all modified lines in the data cache and invalidate both internal caches. A Flush Acknowledge special cycle is generated by the processor, indicating completion of the invalidation and writeback.

FLUSH# is implemented in the processor as an interrupt, so it is recognized on instruction boundaries. External interrupts are ignored while FLUSH# is being serviced. Once FLUSH# is sampled active, it is ignored until the flush acknowledge special cycle is driven.

If FLUSH# is sampled low when RESET transitions from high to low, three-state test mode is entered.

The processor, when operating with a second processor in dual processing mode, incorporates a mechanism to present an atomic cache flush operation to the system. The Dual processor performs the cache flush operation first, then grants the bus to the Primary processor. The Primary processor flushes its internal caches, and then runs the cache flush special cycle. This could cause the total flush latency of two embedded Pentium processors in dual processor mode to be up to twice that of the embedded Pentium processor in uni-processor mode.

The flush latency of the embedded Pentium processor with MMX technology may be up to twice that of the embedded Pentium processor due to the implementation of larger on-chip caches.

When Sampled/Driven

FLUSH# is sampled on every rising clock edge. FLUSH# is falling edge sensitive and is recognized on instruction boundaries. Recognition of FLUSH# is guaranteed in a specific clock if it is asserted synchronously and meets the setup and hold times. If it meets setup and hold times, FLUSH# need only be asserted for one clock. To guarantee recognition if FLUSH# is asserted

asynchronously, it must have been deasserted for a minimum of two clocks before being returned active to the embedded Pentium processor and remain asserted for a minimum pulse width of two clocks.

If the processor is in the HALT or Shutdown state, FLUSH# is still recognized. The processor returns to the HALT or Shutdown state after servicing the FLUSH#.

If FLUSH# is sampled low when RESET transitions from high to low, three-state test mode is entered. If RESET is negated synchronously, FLUSH# must be at its valid level and meet setup and hold times on the clock before the falling edge of RESET. If RESET is negated asynchronously, FLUSH# must be at its valid level two clocks before and after RESET transitions from high to low.

When operating in a dual processing system, FLUSH# must be sampled synchronously to the rising CLK edge to ensure both processors recognize an active FLUSH# signal in the same clock.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS# and cycle definition pins.	Writeback cycles are driven as a result of FLUSH# assertion. The Flush Special Cycle is driven as a result of FLUSH# assertion.
RESET	If FLUSH# is sampled low when RESET transitions from high to low, three-state test mode is entered.
CPUTYP	When operating in dual processing mode, the FLUSH# inputs become synchronous to the processor clock.

5.1.29 FRCMC#

FRCMC#	Functional Redundancy Checking Master/Checker Configuration
	Determines whether the processor is configured as a Master or Checker.
	Asynchronous Input

Note: Functional Redundancy Checking is not supported on the embedded Pentium processor with MMX technology. The FRCMC# pin is defined only for the embedded Pentium processor. This pin should be left as a "NC" or tied to V_{CC3} via an external pullup resistor on the embedded Pentium processor with MMX technology.

Signal Description

The Functional Redundancy Checking Master/Checker Configuration input is sampled in every clock that RESET is asserted to determine whether the processor is configured in master mode (FRCMC# high) or checker mode (FRCMC# low). When configured as a master, the processor drives its output pins as required by the bus protocol. When configured as a checker, the processor three-states all outputs (except IERR# and TDO) and samples the output pins that would normally be driven in master mode. If the sampled value differs from the value computed internally, the Checker processor asserts IERR# to indicate an error.

Note that the final configuration as a master or checker is set after RESET and may not be changed other than by a subsequent RESET. FRCMC# is sampled in every clock that RESET is asserted to prevent bus contention before the final mode of the processor is determined.

When Sampled

This pin is sampled in any clock in which RESET is asserted. FRCMC# is sampled in the clock before RESET transitions from high to low to determine the final mode of the processor. If RESET is negated synchronously, FRCMC# must be at its valid level and meet setup and hold times on the clock before the falling edge of RESET. If RESET is negated asynchronously, FRCMC# must be at its valid level two clocks before and after RESET transitions from high to low.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
IERR#	IERR# is asserted by the Checker processor in the event of an FRC error.
RESET	FRCMC# is sampled when RESET is asserted to determine if the processor is in Master or Checker mode.

5.1.30 HIT#

HIT#	Inquire Cycle Hit/Miss
	Externally indicates whether an inquire cycle resulted in a hit or miss.
	Synchronous Input/Output

Signal Description

The HIT# output is driven to reflect the outcome of an inquire cycle. If an inquire cycle hits a valid line (M, E, or S) in either the processor data or instruction cache, HIT# is asserted two clocks after EADS# has been sampled asserted by the processor. If the inquire cycle misses the processor cache, HIT# is negated two clocks after EADS# is sampled asserted. This pin changes its value only as a result of an inquire cycle and retains its value between cycles.

When operating in dual processing mode, the embedded Pentium processor uses this signal for private snooping.

When Sampled/Driven

HIT# reflects the hit or miss outcome of the inquire cycle two clocks after EADS# is sampled asserted. After RESET, this pin is driven high. It changes it value only as a result of an inquire cycle. This pin is always driven. It is not floated during bus HOLD or BOFF#.

This signal becomes an Input/Output when two embedded Pentium processors are operating together in dual processing mode.

Relation to Other Signals

Pin Symbol	Relative to Other Signals
A31–A5	HIT# is driven to indicate whether the inquire address driven on A31–A5 is valid in an internal cache.
EADS#	HIT# is driven two clocks after EADS# is sampled asserted to indicate the outcome of the inquire cycle.
HITM#	HITM# is never asserted without HIT# also being asserted.

5.1.31 HITM#

HITM#	Inquire Cycle Hit/Miss to a Modified Line
	Externally indicates whether an inquire cycle hit a modified line in the data cache.
	Synchronous Input/Output

Signal Description

The HITM# output is driven to reflect the outcome of an inquire cycle. If an inquire cycle hits a modified line in the embedded Pentium processor data cache, HITM# is asserted two clocks after EADS# has been sampled asserted by the processor and a writeback cycle is scheduled to be driven to the bus. If the inquire cycle misses the processor cache, HITM# is negated two clocks after EADS# is sampled asserted.

HITM# can be used to inhibit another bus master from accessing the data until the line is completely written back.

HITM# is asserted two clocks after an inquire cycle hits a modified line in the processor cache. ADS# for the writeback cycle is asserted no earlier than two clocks after the assertion of HITM#. ADS# for the writeback cycle is driven even if AHOLD for the inquire cycle is not yet deasserted. ADS# for a writeback of an external snoop cycle is the only ADS# that is driven while AHOLD is asserted.

When operating in dual processing mode, the embedded Pentium processor uses this signal for private snooping.

When Sampled/Driven

HITM# is driven two clocks after EADS# is sampled asserted to reflect the outcome of the inquire cycle. HITM# remains asserted until two clocks after the last BRDY# of writeback is returned. This pin is always driven. It is not floated during bus HOLD or BOFF#.

This signal becomes an input/output when two embedded Pentium processors are operating together in dual processing mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A5	HITM# is driven to indicate whether the inquire address driven on A31–A5 is in the modified state in the data cache.
EADS#	HITM# is driven two clocks after EADS# is sampled asserted.
HIT#	HITM# is never asserted without HIT# also being asserted.

5.1.32 HLDA

HLDA	Bus Hold Acknowledge
	External indication that the processor outputs are floated.
	Synchronous Input/Output

Signal Description

The Bus Hold Acknowledge output goes active in response to a hold request presented on the HOLD pin. HLDA indicates that the processor has given the bus to another local bus master. Internal instruction execution continues from the internal caches during bus HOLD/HLDA.

When leaving bus hold, HLDA is driven inactive and the processor resumes driving the bus. A pending bus cycle is driven in the same clock in which HLDA is deasserted by the processor and one clock after HLDA is deasserted by the processor.

The operation of HLDA is not affected by the assertion of BOFF#. If HOLD is asserted while BOFF# is asserted, HLDA is asserted two clocks later. If HOLD goes inactive while BOFF# is asserted, HLDA is deasserted two clocks later.

When operating in dual processing mode, the embedded Pentium processor uses this signal for private snooping.

When Sampled/Driven

When the embedded Pentium processor bus is idle, HLDA is driven high two clocks after HOLD is asserted, otherwise, HLDA is driven high two clocks after the last BRDY# of the current cycle is returned. It is driven active in the same clock that the embedded Pentium processor floats its bus. When leaving bus hold, HLDA is driven inactive two clocks after HOLD is deasserted and the embedded Pentium processor resumes driving the bus. The HLDA signal is glitch free.

This signal becomes an input/output when two embedded Pentium processors are operating together in dual processing mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A3 ADS# AP BE7#–BE3# CACHE# D/C# D63–D0 DP7–DP0 LOCK# M/IO# PCD PWT SCYC W/R#	These signals float in response to HLDA.
BOFF#	The same pins are floated when HLDA or BOFF# is asserted.
EADS#	EADS# is recognized while HLDA is asserted.
HOLD	The assertion of HOLD causes HLDA to be asserted when all outstanding cycles are complete.

5.1.33 HOLD

HOLD	Bus Hold
	The bus hold request input allows another bus master complete control of the processor bus.
	Synchronous Input

Signal Description

The Bus Hold request input allows another bus master complete control of the embedded Pentium processor bus. In response to HOLD, after completing all outstanding bus cycles the embedded Pentium processor floats most of its output and input/output pins and asserts HLDA. The embedded Pentium processor maintains its bus in this state until HOLD is deasserted. Cycles that are locked together are not interrupted by bus HOLD. HOLD is recognized during RESET.

When Sampled

HOLD is sampled on every rising clock edge including during RESET and INIT.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A3 ADS# AP BE7#–BE3# CACHE# D/C# D63–D0 DP7–DP0 LOCK# M/IO# PCD PWT SCYC W/R#	These are the signals floated in response to HOLD.
HLDA	HLDA is asserted when the processor relinquishes the bus in response to the HOLD request.

5.1.34 IERR#

IERR#	Internal or Functional Redundancy Check Error. (Functional Redundancy Checking is not supported on the embedded Pentium® processor with MMX™ technology or the low-power embedded Pentium processor with MMX technology).
	Alerts the system of internal parity errors and functional redundancy errors.
	Output

Signal Description

The Internal Error output is used to alert the system of two types of errors, internal parity errors and functional redundancy errors.

If a parity error occurs on a read from an internal array (reads during normal instruction execution, reads during a flush operation, reads during BIST and testability cycles, and reads during inquire cycles), the embedded Pentium processor asserts the IERR# pin for one clock and then shuts down. Shutdown occurs provided the processor is not prevented from doing so by the error.

If the embedded Pentium processor is configured as a checker (by FRCMC# being sampled low while RESET is asserted) and a mismatch occurs between the value sampled on the pins and the value computed internally, the embedded Pentium processor asserts IERR# two clocks after the mismatched value is returned. Shutdown is not entered as a result of a function redundancy error.

It is the responsibility of the system to take appropriate action if an internal parity or FRC error occurs.

When Driven

IERR# is driven in every clock. While RESET is active IERR# is driven high. After RESET is deasserted, IERR# is not asserted due to an FRC mismatch until after the first clock of the first bus cycle. Note however that IERR# may be asserted due to an internal parity error before the first bus cycle. IERR# is asserted for one clock for each detected FRC or internal parity error, two clocks after the error is detected. IERR# is asserted for each detected mismatch, so IERR# may be asserted for more than one consecutive clock.

IERR# is not floated with HOLD or BOFF#. IERR# is a glitch free signal.

When paging is turned on, an additional parity check occurs to page 0 for all TLB misses. If this access is a valid entry in the cache and this entry also has a parity error, then IERR# is asserted and shutdown occurs even though the pipeline is frozen to service the TLB miss.

During a TLB miss, a cache lookup occurs (to the data cache for a data TLB miss, or the code cache for a code TLB miss) to a default page 0 physical address until the correct page translation becomes available. At this time, if a valid cache entry is found at the page 0 address, then parity is checked on the data read out of the cache. However, the data is not used until after the correct page address becomes available. If this valid line contains a true parity error, then the error is reported. This does not cause an unexpected parity error. It can cause a parity error and shutdown at a time when the data is not being used because the pipeline is frozen to service the TLB miss. However, it still remains that a true parity error must exist within the cache in order for IERR# assertion and shutdown to occur. For more details on TLB, refer to Section 3.7 of the *Intel Architecture Software Developer's Manual*, Volume 1.

Relation to Other Signals

Pin Symbol	Relative to Other Signals
FRCMC#	If the processor is configured as a Checker, IERR# is asserted in the event of an FRC error.

5.1.35 IGNNE#

IGNNE#	Ignore Numeric Exception
	Determines whether or not numeric exceptions should be ignored.
	Asynchronous Input

Signal Description

This is the Ignore Numeric Exception input. This pin has no effect when the NE bit in CR0 is set to 1. When the CR0.NE bit is 0, this pin functions as follows:

When the IGNNE# pin is asserted, the embedded Pentium processor ignores any pending unmasked numeric exception and continues executing floating-point instructions for the entire duration that this pin is asserted.

When IGNNE# is not asserted and a pending unmasked numeric exception exists, (SW.ES = 1), the embedded Pentium processor behaves as follows:

On encountering a floating-point instruction that is one of FNINIT, FNCLEX, FNSTENV, FNSAVE, FNSTSW, FNSTCW, FNENI, FNDISI, or FNSETPM, the embedded Pentium processor asserts the FERR# pin. Subsequently, the processor opens an interrupt sampling window. The interrupts are checked and serviced during this window. If no interrupts are sampled within this window, the processor then executes these instructions in spite of the pending unmasked exception. For further details please refer to the *Intel Architecture s Software Developer's Manual,* Volume1 (Chapter 7 and Appendix D).

On encountering any floating-point instruction other than FINIT, FCLEX, FSTENV, FSAVE, FSTSW, FSTCW, FENI, FDISI, or FSETPM, the embedded Pentium processor stops execution and waits for an external interrupt.

The embedded Pentium processor, when configured as a Dual processor, ignores the IGNNE# input.

When Sampled/Driven

IGNNE# is sampled on every rising clock edge. Recognition of IGNNE# is guaranteed in a specific clock if it is asserted synchronously and meets setup and hold times. To guarantee recognition if IGNNE# is asserted asynchronously, it must have been deasserted for a minimum of two clocks before being returned active to the embedded Pentium processor and remain asserted for a minimum pulse width of two clocks.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
CPUTYP	When strapped to V_{CC}, the processor ignores the IGNNE# input.

5.1.36 INIT

INIT	Initialization
	Forces the processor to begin execution in a known state without flushing the caches or affecting the floating-point state.
	Asynchronous Input

Signal Description

The Initialization input forces the embedded Pentium 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, model specific registers, and floating-point registers retain the values they had prior to INIT. The embedded Pentium processor starts execution at physical address FFFFFFF0H.

INIT can be used to help performance for DOS extenders written for the 80286. INIT provides a method to switch from protected to real mode while maintaining the contents of the internal caches and floating-point state. INIT may not be used instead of RESET after power-up.

Once INIT is sampled active, the INIT sequence begins on the next instruction boundary (unless a higher priority interrupt is requested before the next instruction boundary). The INIT sequence continues to completion and then normal processor execution resumes, independent of the deassertion of INIT. ADS# is asserted to drive bus cycles even if INIT is not deasserted.

If INIT is sampled high when RESET transitions from high to low, the embedded Pentium processor performs built-in self test (BIST) prior to the start of program execution.

When Sampled

INIT is sampled on every rising clock edge. INIT is an edge sensitive interrupt. Recognition of INIT is guaranteed in a specific clock if it is asserted synchronously and meets the setup and hold times. To guarantee recognition if INIT is asserted asynchronously, it must have been deasserted for a minimum of two clocks before being returned active to the embedded Pentium processor and remain asserted for a minimum pulse width of two clocks. INIT must remain active for three clocks prior to the BRDY# of an I/O write cycle to guarantee that the embedded Pentium processor recognizes and processes INIT right after an I/O write instruction.

If INIT is sampled high when RESET transitions from high to low the embedded Pentium processor performs built-in self test. If RESET is driven synchronously, INIT must be at its valid level the clock before the falling edge of RESET. If RESET is driven asynchronously, INIT must be at its valid level two clocks before and after RESET transitions from high to low.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
RESET	If INIT is sampled high when RESET transitions from high to low, BIST will be performed.

5.1.37 INTR

INTR	External Interrupt
	Indicates that an external interrupt has been generated.
	Asynchronous Input

Signal Description

The INTR input indicates that an external interrupt has been generated. The interrupt is maskable by the IF bit in the EFLAGS register. If the IF bit is set, the embedded Pentium processor will vector to an interrupt handler after the current instruction execution is completed. Upon recognizing the interrupt request, the embedded Pentium processor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the first interrupt acknowledge cycle is completed to assure that the interrupt is recognized.

When the local APIC is hardware disabled, this pin is the INTR input for the processor. It bypasses the local APIC in that case.

When the local APIC is hardware enabled, this pin becomes the programmable interrupt LINT0. It can be programmed in software in any of the interrupt modes. Since this pin is the INTR input when the APIC is disabled, it is logical to program the vector table entry for this pin as ExtINT (i.e., through local mode). In this mode, the interrupt signal is passed on to the processor through the local APIC. The processor generates the interrupt acknowledge, INTA, cycle in response to this interrupt and receives the vector on the processor data bus.

When Sampled/Driven

INTR is sampled on every rising clock edge. INTR is an asynchronous input, but recognition of INTR is guaranteed in a specific clock if it is asserted synchronously and meets the setup and hold times. To guarantee recognition if INTR is asserted asynchronously it must have been deasserted for a minimum of two clocks before being returned active to the embedded Pentium processor.

Note: This applies only when using the APIC in the through local (virtual wire) mode. Once INTR has been asserted (by a rising edge), it must not be asserted again until after the end of the first resulting interrupt acknowledge cycle. Otherwise, the new interrupt may not be recognized. The end of an interrupt acknowledge cycle is defined by the end of the system's BRDY# response to the processor cycle. Note that the APIC through local mode was designed to match the protocol of an 8259A PIC, and an 8259A will always satisfy this requirement.

To ensure INTR is not recognized inadvertantly a second time, deassert INTR no later than the BRDY# of the second INTA cycle and no earlier than the BRDY# of the first INTA cycle.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS# and cycle definition pins	An interrupt acknowledge cycle is driven as a result of the INTR pin assertion.
APICEN	When the APICEN configuration input is sampled inactive, this input becomes the INTR interrupt.
LINT0	INTR shares a pin with LINT0.
LOCK#	LOCK# is asserted for interrupt acknowledge cycles.

5.1.38 INV

INV	Invalidation Request
	Determines final state of a cache line as a result of an inquire hit.
	Synchronous Input

Signal Description

The INV input is driven to the embedded Pentium processor during an inquire cycle to determine the final cache line state (S or I) in case of an inquire cycle hit. If INV is returned active (high) to the embedded Pentium processor in the same clock as EADS# is sampled asserted, an inquire hit will result in that line being invalidated. If the INV pin is returned inactive (low), an inquire hit will result in that line being marked Shared (S). If the inquire cycle is a miss in the cache, the INV input has no effect.

If an inquire cycle hits a modified line in the data cache, the line will be written back regardless of the state of INV.

When Sampled

The INV input is sampled with the EADS# of the inquire cycle.

Relation to Other Signals

Pin Symbol	Relative to Other Signals
A31–A5	INV determines if the inquire address driven to the processor on A31–A5 should be invalidated or marked as shared if it is valid in an internal cache.
EADS#	INV is sampled with EADS#.

5.1.39 KEN#

KEN#	Cache Enable
	Indicates to the processor whether or not the system can support a cache line fill for the current cycle.
	Synchronous Input

Signal Description

KEN# is the cache enable input. It is used to determine whether the current cycle is cacheable or not and consequently is used to determine cycle length.

When the embedded Pentium processor generates a read cycle that can be cached (CACHE# asserted) and KEN# is active, the cycle will be transformed into a burst cache linefill. During a cache line fill the byte enable outputs should be ignored and valid data must be returned on all 64 data lines. The embedded Pentium processor will expect 32 bytes of valid data to be returned in four BRDY# transfers.

If KEN# is not sampled active, a linefill will not be performed (regardless of the state of CACHE#) and the cycle will be a single transfer read.

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

When Sampled

KEN# is sampled once in a cycle to determine cacheability. It is sampled and latched with the earlier of the first BRDY# or NA# of a cycle, however it must meet setup and hold times on every clock edge.

Relation to Other Signals

Pin Symbol	Relative to Other Signals
BRDY#	KEN# is sampled with the earlier of the first BRDY# or NA# for that cycle. Also, in conjunction with the CACHE# input, KEN# determines whether the bus cycle will consist of 1 or 4 transfers (assertions of BRDY#).
CACHE#	KEN# determines cacheability only if the CACHE# pin is asserted.
NA#	KEN# is sampled with the earlier of the first BRDY# or NA# for that cycle.
W/R#	KEN# determines cacheability only if W/R# indicates a read.

5.1.40 LINT1–LINT0

LINT1–LINT0	Local Interrupts 1 and 0
	APIC Programmable Interrupts.
	Asynchronous Inputs

Signal Description

When the local APIC is hardware enabled, these pins become the programmable interrupts (LINT1–LINT0). They can be programmed in software in any of the interrupt modes. Since these pins are the INTR and NMI inputs when the APIC is disabled, it is logical to program the vector table entry for them as ExtINT (i.e. through local mode) and NMI, respectively. In this mode, the interrupt signals are passed on to the processor through the local APIC.

When the local APIC is hardware disabled, these pins are the INTR and NMI inputs for the processor. They bypass the APIC in that case.

When Sampled

LINT1–LINT0 are sampled on every rising clock edge. LINT1–LINT0 are asynchronous inputs, but recognition of LINT1–LINT0 are guaranteed in a specific clock if they are asserted synchronously and meets the setup and hold times. To guarantee recognition if LINT1–LINT0 are asserted asynchronously they must have been deasserted for a minimum of two clocks before being returned active to the embedded Pentium processor.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
APICEN	When the APICEN configuration input is sampled inactive, these inputs become the INTR and NMI interrupts.
INTR	INTR shares a pin with LINT0.
NMI	NMI shares a pin with LINT1.

5.1.41 LOCK#

LOCK#	Bus Lock
	Indicates to the system that the current sequence of bus cycles should not be interrupted.
	Synchronous Input/Output

Signal Description

The bus lock output indicates that the embedded Pentium processor is running a read-modify-write cycle where the external bus must not be relinquished between the read and write cycles. Read-modify-write cycles of this type are used to implement memory based semaphores. Interrupt Acknowledge cycles are also locked.

If a cycle is split due to a misaligned memory operand, two reads followed by two writes may be locked together. When LOCK# is asserted, the current bus master should be allowed exclusive access to the system bus.

The embedded Pentium processor will not allow a bus hold when LOCK# is asserted, but address holds (AHOLD) and BOFF# are allowed. LOCK# is floated during bus hold.

All locked cycles will be driven to the external bus. If a locked address hits a valid location in one of the internal caches, the cache location is invalidated (if the line is in the modified state, it is written back before it is invalidated). Locked read cycles will not be transformed into cache line fill cycles regardless of the state of KEN#.

LOCK# is guaranteed to be deasserted for at least one clock between back to back locked cycles.

When operating in dual processing mode, the embedded Pentium processor uses this signal for private snooping.

When Sampled/Driven

LOCK# goes active with the ADS# of the first locked bus cycle and goes inactive after the BRDY# is returned for the last locked bus cycle. The LOCK# signal is glitch free.

This signal becomes an input/output when two embedded Pentium processors are operating together in dual processing mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	LOCK# is driven with the ADS# of the first locked cycle.
BOFF#	LOCK# floats one clock after BOFF# is asserted.
BRDY#	LOCK# is deasserted after the last BRDY# of the locked sequence.
HLDA	LOCK# floats when HLDA is asserted.
NA#	ADS# is not asserted to pipeline an additional cycle if LOCK# is asserted, regardless of the state of NA#.
INTR	LOCK# is asserted for interrupt acknowledge cycles.
SCYC	SCYC is driven active if the locked cycle is misaligned.

5.1.42 M/IO#

M/IO#	Memory Input/Output
	Distinguishes a memory access from an I/O access.
	Synchronous Input/Output

Signal Description

The Memory/Input-Output signal is one of the primary bus cycle definition pins. M/IO# distinguishes between memory (M/IO# =1) and I/O (M/IO# =0) cycles.

When operating in dual processing mode, the embedded Pentium processor uses this signal for private snooping.

When Sampled/Driven

M/IO# is driven valid in the same clock as ADS# and the cycle address. It remains valid from the clock in which ADS# is asserted until the earlier of the last BRDY# or the clock after NA#.

This signal becomes an input/output when two embedded Pentium processors are operating together in dual processing mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	M/IO# is driven to its valid state with ADS#.
BOFF#	M/IO# floats one clock after BOFF# is asserted.
HLDA	M/IO# floats when HLDA is asserted.

5.1.43 NA#

NA#	Next Address
	Indicates that external memory is prepared for a pipelined cycle.
	Synchronous Input

Signal Description

The Next Address input, when active, indicates that external memory is ready to accept a new bus cycle although all data transfers for the current cycle have not yet completed. This is referred to as bus cycle pipelining.

The embedded Pentium processor will drive out a pending cycle in response to NA# no sooner than two clocks after NA# is asserted. The embedded Pentium processor supports up to 2 outstanding bus cycles. ADS# is not asserted to pipeline an additional cycle if LOCK# is asserted, or during a writeback cycle. In addition, ADS# will not be asserted to pipeline a locked cycle or a writeback cycle into the current cycle.

NA# is latched internally, so once it is sampled active during a cycle, it need not be held active to be recognized. The KEN#, and WB/WT# inputs for the current cycle are sampled with the first NA#, if NA# is asserted before the first BRDY# of the current cycle.

When Sampled

NA# is sampled in all T2, TD and T2P clocks.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	If NA# is sampled asserted and an internal bus request is pending, the processor drives out the next bus cycle and asserts ADS#.
KEN#	KEN# is sampled with the earlier of the first BRDY# or NA# for that cycle.
WB/WT#	WB/WT# is sampled with the earlier of the first BRDY# or NA# for that cycle.
LOCK#	ADS# is not asserted to pipeline an additional cycle if LOCK# is asserted, regardless of the state of NA#.
BOFF#	If NA# and BOFF# are asserted simultaneously, BOFF# takes priority and NA# is ignored.

5.1.44 NMI

NMI	Non-Maskable Interrupt
	Indicates that an external non-maskable interrupt has been generated.
	Asynchronous Input

Signal Description

The Non-Maskable interrupt request input indicates that an external non-maskable interrupt has been generated. Asserting NMI causes an interrupt with an internally supplied vector value of 2. External interrupt acknowledge cycles are not generated.

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

When the local APIC is hardware enabled, this pin becomes the programmable interrupt LINT1. It can be programmed in software in any of the interrupt modes. Since this pin is the NMI input when the APIC is disabled, it is logical to program the vector table entry for this pin as NMI. In this mode, the interrupt signal is passed on to the processor through the local APIC.

When the local APIC is hardware disabled, this pin is the NMI input for the processor. It bypasses the APIC in that case.

When Sampled

NMI is sampled on every rising clock edge. NMI is rising edge sensitive. Recognition of NMI is guaranteed in a specific clock if it is asserted synchronously and meets the setup and hold times. To guarantee recognition if NMI is asserted asynchronously, it must have been deasserted for a minimum of two clocks before being returned active to the embedded Pentium processor and remain asserted for a minimum pulse width of two clocks.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
APICEN	When the APICEN configuration input is sampled inactive, this input becomes the NMI interrupt.
LINT1	NMI shares a pin with LINT1.

5.1.45 PBGNT#

PBGNT#	Dual Processor Bus Grant
	Indicates to the LRM processor that it will become the MRM in the next clock.
	Synchronous Input (to the Least Recent Master, LRM, processor) Synchronous Output (of the Most Recent Master, MRM, processor)

Signal Description

Two embedded Pentium processors, when configured as dual processors, will arbitrate for the system bus via two private arbitration pins (PBREQ# and PBGNT#). The processor that currently owns the system bus is referred to as the MRM processor. The processor that does not own the bus is referred to as the LRM processor.

PBGNT# is used by the dual processing private arbitration mechanism to indicate that bus ownership will change in the next clock. The LRM processor will request ownership of the processor bus by asserting the private arbitration request pin, PBREQ#. The processor that is currently the MRM and owns the bus, will grant the bus to the LRM as soon as any pending bus transactions have completed. The MRM will notify that the LRM can assume ownership by asserting the private arbitration grant pin, PBGNT#. The PBGNT# pin is always the output of the MRM and an input to the LRM.

Note: In a single socket system design, PBGNT# pin should be left NC. For proper operation, PBGNT# must not be loaded by the system.

When Sampled/Driven

PBGNT# is driven by the MRM processor in response to the PBREQ# signal from the LRM processor. It is asserted following the completion of the current cycle on the processor bus, or in the clock following the request if the bus is idle.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
PBREQ#	PBGNT# is asserted in response to a bus request, PBREQ#, from the LRM processor.
A31–A3, AP, BE7#–BE0#, CACHE#, D/C#, M/IO#, PCD, PWT, SCYC, W/R#	These signals are three-stated for one CLK in response to PBGNT# (when the MRM becomes the LRM).

5.1.46 PBREQ#

PBREQ#	Dual Processor Bus Request
	Indicates to the MRM processor that the LRM processor requires ownership of the bus.
	Synchronous Input (to the Most Recent Master, MRM, processor) Synchronous Output (of the Least Recent Master, LRM, processor)

Signal Description

Two embedded Pentium processors, when configured as dual processors, will arbitrate for the system bus via two private arbitration pins (PBREQ# and PBGNT#). The processor that currently owns the system bus is referred to as the MRM processor. The processor that does not own the bus is referred to as the LRM processor.

PBREQ# is used by the dual processing private arbitration mechanism to indicate that the LRM processor requests bus ownership. The processor that is currently the MRM and owns the bus, will grant the bus to the LRM as soon as any pending bus transactions have completed. The MRM will notify that the LRM can assume ownership by asserting the private arbitration grant pin, PBGNT#. The PBREQ**#** pin is always the output of the LRM and an input to the MRM.

Note: In a single socket system design, PBREQ# pin should be left NC. For proper operation, PBREQ# must not be loaded by the system.

When Sampled/Driven

PBREQ# is driven by the LRM processor, and sampled by the MRM processor.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
PBGNT#	PBGNT# is asserted in response to a bus request, PBREQ#, from the LRM processor.

5.1.47 PCD

PCD	Page Cacheability Disable
	Externally reflects the cacheability paging attribute bit in CR3, PDE, or PTE.
	Output

Signal Description

PCD is driven to externally reflect the cache disable paging attribute bit for the current cycle. PCD corresponds to bit 4 of CR3, the Page Directory Entry, or the Page Table Entry. For cycles that are not paged when paging is enabled (for example I/O cycles), PCD corresponds to bit 4 in CR3. In real mode or when paging is disabled, the PCD pin reflects the cache disable bit in control register 0 (CR0.CD).

PCD is masked by the CD (cache disable) bit in CR0. When CD=1, the embedded Pentium processor forces PCD high. When CD=0, PCD is driven with the value of the Page Table Entry/Directory.

The purpose of PCD is to provide an external cacheability indication on a page by page basis.

When Driven

The PCD pin is driven valid in the same clock as ADS# and the cycle address. It remains valid from the clock in which ADS# is asserted until the earlier of the last BRDY# or the clock after NA#.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	PCD is driven valid with ADS#.
BOFF#	PCD floats one clock after BOFF# is asserted.
HLDA	PCD floats when HLDA is asserted.

5.1.48 PCHK#

PCHK#	Data Parity Check
	Indicates the result of a parity check on a data read.
	Synchronous Output

Signal Description

The data parity check pin indicates the result of a parity check on a data read. Data parity is checked during code reads, memory reads, and I/O reads. Data parity is not checked during the first Interrupt Acknowledge cycle. PCHK# indicates the parity status only for the bytes on which valid data is expected. Parity is checked for all data bytes for which a byte enable is asserted. In addition, during a cache linefill, parity is checked on the entire data bus regardless of the state of the byte enables.

PCHK# is driven low two clocks after BRDY# is returned if incorrect parity was returned.

Driving PCHK# is the only effect that bad data parity has on the embedded Pentium processor unless PEN# is also asserted. The data returned to the processor is not discarded.

If PEN# is asserted when a parity error occurs, the cycle address and type will be latched in the MCA and MCT registers. If in addition, the MCE bit in CR4 is set, a machine check exception will be taken.

It is the responsibility of the system to take appropriate actions if a parity error occurs. If parity checks are not implemented in the system, the PCHK# pin may be ignored, and PEN# pulled high (or CR4.MCE cleared).

When operating in dual processing mode, the PCHK# signal can be asserted either 2 OR 3 CLKs following incorrect parity being detected on the data bus. When operating in Dual Processing mode, the PCHK# pin circuit is implemented as a weak driving high output that operates similar to

an open drain output. This implementation allows connection of the two processor PCHK# pins together in a dual processing system with no ill effects. Nominally, this circuit acts like a 360 Ohm resistor tied to V_{CC}.

When Sampled/Driven

PCHK# is driven low two clocks after BRDY# is returned if incorrect parity was returned. PCHK# remains low one clock for each clock in which a parity error was detected. At all other times PCHK# is inactive (high). PCHK# is not floated during bus HOLD or BOFF#. PCHK# is a glitch free signal.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BRDY#	PCHK# is driven to its valid level two clocks after the assertion of BRDY#.
D63–D0	The DP7–DP0 pins are used to create even parity with D63–D0. If even parity is not returned, the PCHK# pin is asserted.
DP7–DP0	Even data parity with D63–D0 should be returned on to the processor on the dual processor pin. If even parity is not returned, the PCHK# pin is asserted.

5.1.49 PHIT#

PHIT#	Private Inquire Cycle Hit/Miss Indication
	Indicates whether a private, dual processor, inquire cycle resulted in a hit or miss.
	Synchronous Input (to the Most Recent Master, MRM, processor) Synchronous Output (of the Least Recent Master, LRM, processor)

Signal Description

A private snoop interface has been added to the embedded Pentium processor for use in dual processing. The interface consists of two pins (PHIT# and PHITM#).

The LRM processor will initiate a snoop sequence for all ADS# cycles that are initiated by the MRM. The LRM processor will assert the private hit indication (PHIT#) if the data requested by the MRM matches a valid cache line in the LRM. In addition, if the data requested by the MRM matches a valid cache line in the LRM that is in the modified state, the LRM will also assert the PHITM# signal. The system snooping indication signals (HIT#, HITM#) will not change state as a result of a private snoop.

The MRM will use an assertion of the PHIT# signal as an indication that the requested data is being shared with the LRM. Independent of the WB/WT# pin, a cache line will be placed in the shared state if PHIT# is asserted. This will make all subsequent writes to that line externally visible until the state of the line becomes exclusive (E or M states). In a uni-processor system, the line may have been placed in the cache in the E state. In this situation, all subsequent writes to that line will not be visible on the bus until the state is changed to I.

PHIT# will also be driven by the LRM during external snoop operations (e.g., following EADS#) to indicate the private snoop results.

Note: In a single socket system, PHIT# pin should be left NC. For proper operation, PHIT# must not be loaded by the system.

When Sampled/Driven

PHIT# is driven by the LRM processor, and sampled by the MRM processor. It is asserted within two clocks following an assertion of ADS# or EADS#.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A5	PHIT# is driven to indicate whether the private inquire address driven on A31–A5 is valid in the LRM's on-chip cache.
ADS#	PHIT# is driven within two clocks after ADS# is sampled asserted to indicate the outcome of the private inquire cycle.
EADS#	PHIT# is driven within two clocks after EADS# is sampled asserted to indicate the outcome of the external inquire cycle.
PHITM#	PHITM# is never asserted without PHIT# also being asserted.
WB/WT#	The state of the WB/WT# pin will be ignored by the MRM if the PHIT# pin is sampled active, and the cache line placed in the shared state.

5.1.50 PHITM#

PHITM#	Private Inquire Cycle Hit/Miss to a Modified Line Indication
	Indicates whether a private, dual processor, inquire cycle resulted in a hit or miss to a Modified line.
	Synchronous Input (to the Most Recent Master, MRM, processor) Synchronous Output (of the Least Recent Master, LRM, processor)

Signal Description

A private snoop interface has been added to the embedded Pentium processor for use in dual processing. The interface consists of two pins (PHIT# and PHITM#).

The LRM processor will initiate a snoop sequence for all ADS# cycles that are initiated by the MRM. The LRM processor will assert the private hit indication (PHIT#) if the data requested by the MRM matches a valid cache line in the LRM. In addition, if the data requested by the MRM matches a valid cache line in the LRM that is in the modified state, the LRM will also assert the PHITM# signal. The system snooping indication signals (HIT#, HITM#) will not change state as a result of a private snoop.

PHITM# will also be driven by the LRM during external snoop operations (e.g. following EADS#) to indicate the private snoop results.

Note: In a single socket system, PHITM# pin should be left NC. For proper operation, PHITM# must not be loaded by the system.

When Sampled/Driven

PHITM# is driven by the LRM processor, and sampled by the MRM processor. It is asserted within two clocks following an assertion of ADS# or EADS#.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A31–A5	PHITM# is driven to indicate whether the private inquire address driven on A31–A5 is modified in the LRM's on-chip cache.
ADS#	PHITM# is driven within two clocks after ADS# is sampled asserted to indicate the outcome of the private inquire cycle.
EADS#	PHITM# is driven within two clocks after EADS# is sampled asserted to indicate the outcome of the external inquire cycle.
PHIT#	PHITM# is never asserted without PHIT# also being asserted.

5.1.51 PICCLK

PICCLK	Processor Interrupt Controller Clock
	This pin drives the clock for the APIC serial data bus operation.
	Input

Signal Description

This pin provides the clock timings for the on-chip APIC unit of the processor. This clock input controls the frequency for the APIC operation and data transmission on the 2-wire APIC serial data bus. All the timings on APIC bus are referenced to this clock.

When hardware disabled, PICCLK must be tied high.

Note that the PICCLK signal on the embedded Pentium processor with MMX technology is 3.3V tolerant, while on the embedded Pentium processor the PICCLK input is 5.0V tolerant.

When Sampled

PICCLK is a clock signal and is used as a reference for sampling the APIC data signals.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
APICEN	PICCLK must be tied or driven high when APICEN is sampled low at the falling edge of RESET.
PICD0–PICD1	External timing parameters for the PICD0–PICD1 pins are measured with respect to this clock.

5.1.52 PICD1–PICD0

PICD1–PICD0	Processor Interrupt Controller Data
	These are the data pins for the 3-wire APIC bus.
	Synchronous Input/Output to PICCLK
	Needs external pull-up resistors.

Signal Description

The PICD1–PICD0 are bidirectional pins which comprise the data portion of the 3-wire APIC bus.

When Sampled/Driven

These signals are sampled with the rising edge of PICCLK.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
APICEN	PICD1 shares a pin with APICEN.
DPEN#	PICD0 shares a pin with DPEN#.

5.1.53 PEN#

PEN#	Parity Enable
	Indicates to the processor that the correct data parity is being returned by the system. Determines if a Machine Check Exception should be taken if a data parity error is detected.
	Synchronous Input

Signal Description

The PEN# 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 embedded Pentium 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 embedded Pentium processor will vector to the machine check exception before the beginning of the next instruction. If this pin is sampled inactive, it does not prevent PCHK# from being asserted in response to a bus parity error. If systems are using PCHK#, they should be aware of this usage of PEN#.

This pin may be tied to V_{SS}.

When Sampled

This signal is sampled when BRDY# is asserted for memory and I/O read cycles and the second interrupt acknowledge cycle.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BRDY#	PEN# is sampled with BRDY# for read cycles.
D63–D0	The DP7–DP0 pins are used to create even parity with D63–D0. If even parity is not returned, and PEN# is enabled, the cycle will be latched and an MCE will be taken if CR4.MCE = 1.
DP7–DP0	Even data parity with D63–D0 should be returned to the processor on the dual-processor pins. If even parity is not returned, and PEN# is enabled, the cycle will be latched and a MCE will be taken if CR4.MCE = 1.

5.1.54 PM1–PM0

PM1/BP1–PM0/BP0	Performance Monitoring
	PM1–PM0 externally indicate the status of the performance monitor counter.
	Output pins

Signal Description

The performance monitoring pins can be individually configured to externally indicate either that the associated performance monitoring counter has incremented or that it has overflowed. PM1 indicates the status of CTR1; PM0 indicates the status of CTR0.

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.

When Driven

The BP3–BP2, PM1/BP1–PM0/BP0 pins are driven in every clock and are not floated during bus HOLD or BOFF#.

The PM1/PM0 pins externally indicate the status of the performance monitoring counters on the embedded Pentium processor. These counters are undefined after RESET, and must be cleared or pre-set (using the WRMSR instruction) before they are assigned to specific events.

However, it is possible for these pins to toggle even during RESET. This may occur ONLY if the RESET pin was asserted while the embedded Pentium processor was in the process of counting a particular performance monitoring event. Since the event counters continue functioning until the CESR (Control and Event Select Register) is cleared by RESET, it is possible for the event counters to increment even during RESET. Externally, the state of the event counters would also be reflected on the PM1/PM0 pins. Any assertion of the PM1/PM0 pins during RESET should be ignored until after the start of the first bus cycle.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BP1–BP0	PM1 and PM0 are share pins with BP1 and BP0.

5.1.55 PRDY

PRDY	Probe Ready
	For use with the Intel debug port.
	Output

Signal Description

The PRDY pin is provided for use with the Intel debug port described in the Chapter 13, “Debugging.”

When Driven

This output is always driven by the embedded Pentium processor. It is not floated during bus HOLD or BOFF#.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
R/S#	R/S# is also used with the Intel debug port.

5.1.56 PWT

PWT	Page Writethrough
	Externally reflects the writethrough paging attribute bit in CR3, PDE, or PTE.
	Output

Signal Description

PWT is driven to externally reflect the cache writethrough paging attribute bit for the current cycle. PWT corresponds to bit 3 of CR3, the Page Directory Entry, or the Page Table Entry. For cycles that are not paged when paging is enabled (for example I/O cycles), PWT corresponds to bit 3 in CR3. In real mode or when paging is disabled, the embedded Pentium processor drives PWT low.

PWT can override the effect of the WB/WT# pin. If PWT is asserted for either reads or writes, the line is saved in, or remains in, the Shared (S) state.

When Driven

The PWT pin is driven valid in the same clock as ADS# and the cycle address. It remains valid from the clock in which ADS# is asserted until the earlier of the last BRDY# or the clock after NA#.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	PWT is driven valid with ADS#.
BOFF#	PWT floats one clock after BOFF# is asserted.
HLDA	PWT floats when HLDA is asserted.
WB/WT#	PWT is used in conjunction with the WB/WT# pin to determine the MESI state of cache lines.

5.1.57 R/S#

R/S#	Run/Stop
	For use with the Intel debug port.
	Asynchronous Input

Signal Description

The R/S# pin is provided for use with the Intel debug port described in Chapter 13, “Debugging.”

When Sampled

This pin should not be driven except in conjunction with the Intel debug port.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
PRDY	PRDY is also used with the Intel debug port.

5.1.58 RESET

RESET	Reset
	Forces the processor to begin execution at a known state.
	Asynchronous Input (Normal, Uni-processor, mode) Synchronous Input (Dual processor mode)

Signal Description

The RESET input forces the embedded Pentium processor to begin execution at a known state. All the embedded Pentium processor internal caches (code and data caches, the translation lookaside buffers, branch target buffer and segment descriptor cache) will be invalidated upon the RESET. Modified lines in the data cache are not written back. When RESET is asserted, the embedded Pentium processor will immediately abort all bus activity and perform the RESET sequence. The embedded Pentium processor starts execution at FFFFFFF0H.

When RESET transitions from high to low, FLUSH# is sampled to determine if three-state test mode is to be entered, FRCMC# is sampled to determine if the embedded Pentium processor will be configured as a master or a checker (only on the embedded Pentium processor), and INIT is sampled to determine if BIST will be run.

When Sampled/Driven

RESET is sampled on every rising clock edge. RESET must remain asserted for a minimum of 1 millisecond after V_{CC} and CLK have reached their AC/DC specifications for the “cold” or “power on” reset. During power up, RESET should be asserted while V_{CC} is approaching nominal operating voltage (the simplest way to insure this is to place a pullup resistor on RESET). RESET must remain active for at least 15 clocks while V_{CC} and CLK are within their operating limits for a “warm reset.” Recognition of RESET is guaranteed in a specific clock if it is asserted synchronously and meets the setup and hold times. To guarantee recognition if RESET is asserted asynchronously, it must have been deasserted for a minimum of two clocks before being returned active to the embedded Pentium processor.

FLUSH#, FRCMC# and INIT are sampled when RESET transitions from high to low to determine if three-state test mode or checker mode will be entered, or if BIST will be run. If RESET is driven synchronously, these signals must be at their valid level and meet setup and hold times on the clock before the falling edge of RESET. If RESET is driven asynchronously, these signals must be at their valid level two clocks before and after RESET transitions from high to low.

When operating in a dual processing system, RESET is sampled synchronously to the rising CLK edge to ensure both processors recognize the falling edge in the same clock.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
APICEN	APICEN is sampled at the falling edge of RESET.
BE3#–BE0#	During reset the BE3#–BE0# pins are sampled to determine the APIC ID. Following RESET, they function as Byte Enable outputs.
BF1–BF0	BF1–BF0 are sampled at the falling edge of RESET.
CPUTYP	CPUTYP is sampled at the falling edge of RESET.
DPEN#	DPEN# is valid during RESET.
FLUSH#	If FLUSH# is sampled low when RESET transitions from high to low, three-state test mode will be entered.
FRCMC#	FRCMC# is sampled when RESET transitions from high to low to determine if the embedded Pentium processor is in Master or Checker mode.
INIT	If INIT is sampled high when RESET transitions from high to low, BIST will be performed.

5.1.59 SCYC

SCYC	Split Cycle Indication
	Indicates that a misaligned locked transfer is on the bus.
	Synchronous Input/Output

Signal Description

The Split Cycle output is activated during misaligned locked transfers. It is asserted 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.

The embedded Pentium processor defines misaligned transfers as a 16-bit or 32-bit transfer which crosses a 4-byte boundary, or a 64-bit transfer which crosses an 8-byte boundary.

When operating in dual processing mode, the embedded Pentium processor uses this signal for private snooping.

When Sampled/Driven

SCYC is only driven during the length of the locked cycle that is split. SCYC is asserted with the first ADS# of a misaligned split cycle and remains valid until the earlier of the last BRDY# of the last split cycle or the clock after NA# of the last split cycle.

This signal becomes an input/output when two embedded Pentium processors are operating together in dual processing mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	SCYC is driven valid in the same clock as ADS#.
BOFF#	SCYC is floated one clock after BOFF# is asserted.
HLDA	SCYC is floated when HLDA is asserted.
LOCK#	SCYC is defined for locked cycles only.

5.1.60 SMI#

SMI#	System Management Interrupt
	Latches a System Management Interrupt request.
	Asynchronous Input
	Internal Pullup Resistor

Signal Description

The System Management Interrupt input latches a System Management Interrupt request. After SMI# is recognized on an instruction boundary, the embedded Pentium processor waits for all writes to complete and EWBE# to be asserted, then asserts the SMIACT# output. The processor will then save its register state to SMRAM space and begin to execute the SMM handler. The RSM instruction restores the registers and returns to the user program.

SMI# has greater priority than debug exceptions and external interrupts. This means that if more than one of these conditions occur at an instruction boundary, only the SMI# processing occurs, not a debug exception or external interrupt. Subsequent SMI# requests are not acknowledged while the processor is in system management mode (SMM). The first SMI# interrupt request that occurs while the processor is in SMM is latched, and serviced when the processor exits SMM with the RSM instruction. Only one SMI# will be latched by the processor while it is in SMM.

When Sampled

SMI# is sampled on every rising clock edge. SMI# is a falling edge sensitive input. Recognition of SMI# is guaranteed in a specific clock if it is asserted synchronously and meets the setup and hold times. To guarantee recognition if SMI# is asserted asynchronously, it must have been deasserted for a minimum of two clocks before being returned active to the embedded Pentium processor and remain asserted for a minimum pulse width of two clocks.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
SMIACT#	When the SMI# input is recognized, the processor asserts SMIACT#.

5.1.61 SMIACT#

SMIACT#	System Management Interrupt Active
	Indicates that the processor is operating in SMM.
	Synchronous Output

Signal Description

The System Management Interrupt Active output is asserted in response to the assertion of SMI#. It indicates that the processor is operating in System Management Mode (SMM). It will remain active (low) until the processor executes the RSM instruction to leave SMM.

When the system is operating in dual processing mode, the D/P# signal alternates between asserted and deasserted based on whether the Primary or Dual processor owns the bus (MRM). The SMIACT# pins may be tied together or be used separately to insure SMRAM access by the correct processor.

Caution: If SMIACT# is used separately, note that the SMIACT# signal is only driven by the processor when it is the MRM (so this signal must be qualified with the D/P# signal).

Connecting the SMIACT# signals on the Primary and Dual processors together is strongly recommended for operation with the Dual processor and upgradability with the Pentium OverDrive® processor.

In dual processing systems, SMIACT# may not remain low (e.g., may toggle) if both processors are not in SMM mode. The SMIACT# signal is asserted by either the Primary or Dual processor based on two conditions: the processor is in SMM mode and is the bus master (MRM). If one processor is executing in normal address space, the SMIACT# signal will go inactive when that processor is MRM. The LRM processor, even if in SMM mode, will not drive the SMIACT# signal low.

When Sampled/Driven

SMIACT# is driven active in response to the assertion of SMI# after all internally pending writes are complete and the EWBE# pin is active (low). It will remain active (low) until the processor executes the RSM instruction to leave SMM. This signal is always driven. It does not float during bus HOLD or BOFF#.

When operating in dual processing mode, the SMIACT# output must be sampled with an active ADS# and qualified with the D/P# signal to determine which embedded Pentium processor (i.e., the Primary or Dual) is driving the SMM cycle.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	SMIACT# should be sampled with an active ADS# during dual processing operation.
D/P#	When operating in dual processing mode, D/P# qualifies the SMIACT# SMM indicator.
EWBE#	SMIACT# is not asserted until EWBE# is active.
SMI#	SMIACT# is asserted when the SMI# is recognized.

5.1.62 STPCLK#

STPCLK#	Stop Clock
	Used to stop the internal processor clock and consume less power.
	Asynchronous Input

Signal Description

Assertion of STPCLK# causes the embedded Pentium processor to stop its internal clock and consume less power while still responding to interprocessor and external snoop requests. This low-power state is called the stop grant state. When the processor recognizes a STPCLK# interrupt, the processor will do the following:

1. Wait for all instructions being executed to complete.
2. Flush the instruction pipeline of any instructions waiting to be executed.
3. Wait for all pending bus cycles to complete and EWBE# to go active.
4. Drive a special bus cycle (stop grant bus cycle) to indicate that the clock is being stopped.
5. Enter low power mode.

The stop grant bus cycle consists of the following signal states: M/IO# = 0, D/C# = 0, W/R# = 1, Address Bus = 0000 0010H (A_4 = 1), BE7#–BE0# = 1111 1011, Data bus = undefined.

STPCLK# must be driven high (not floated) to exit the stop grant state. The rising edge of STPCLK# will tell the processor that it can return to program execution at the instruction following the interrupted instruction.

When Sampled/Driven

STPCLK# is treated as a level triggered interrupt to the embedded Pentium processor and is prioritized below all of the external interrupts. When the embedded Pentium processor recognizes the STPCLK# interrupt, the processor will stop execution on the instruction boundary following the STPCLK# assertion.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
A4, Cycle Control signals (M/IO#, D/C#, W/R#, BE7#–BE0#, D/P#)	The Stop Grant Special Bus Cycle is driven on these pins in response to an assertion of the STPCLK# signal. M/IO# = 0, D/C# = 0, W/R# = 1. Address Bus = 0000 0010H (A4 = 1), BE7#–BE0# = 1111 1011.
EWBE#	After STPCLK# has been recognized, all pending cycles must be completed and EWBE# must go active before the internal clock will be disabled.
External Interrupt signals (FLUSH#, INIT, INTR, NMI, R/S#, SMI#)	While in the Stop Grant state, the processor will latch transitions on the external interrupt signals. All of these interrupts are taken after the deassertion of STPCLK#. The processor requires that INTR be held active until the processor issues an interrupt acknowledge cycle in order to guarantee recognition.
HLDA	The processor will not respond to a STPCLK# request from a HLDA state because it cannot generate a Stop Grant cycle.

5.1.63 TCK

TCK	Test Clock Input
	Provides Boundary Scan clocking function.
	Input

Signal Description

This is the Testability Clock input that provides the clocking function for the embedded Pentium processor boundary scan in accordance with the boundary scan interface (IEEE Std 1149.1). It is used to clock state information and data into and out of the embedded Pentium processor during boundary scan. State select information and data are clocked into the embedded Pentium processor on the rising edge of TCK on TMS and TDI inputs respectively. Data is clocked out of the embedded Pentium processor on the falling edge of TCK on TDO.

When TCK is stopped in a low state, the boundary scan latches retain their state indefinitely. When boundary scan is not used, TCK should be tied high or left as a no-connect.

When Sampled

TCK is a clock signal and is used as a reference for sampling other boundary scan signals.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
TDI	Serial data is clocked into the processor on the rising edge of TCK.
TDO	Serial data is clocked out of the processor on the falling edge of TCK.
TMS	TAP controller state transitions occur on the rising edge of TCK.

5.1.64 TDI

TDI	Test Data Input
	Input to receive serial test data and instructions.
	Synchronous Input to TCK

Signal Description

This is the serial input for the Boundary Scan test logic. TAP instructions and data are shifted into the embedded Pentium processor on the TDI pin on the rising edge of TCK when the TAP controller is in the SHIFT-IR and SHIFT-DR states. During all other states, TDI is a “don’t care.”

An internal pull-up resistor is provided on TDI to ensure a known logic state if an open circuit occurs on the TDI path. Note that when “1” is continuously shifted into the instruction register, the BYPASS instruction is selected.

When Sampled

TDI is sampled on the rising edge of TCK during the SHIFT-IR and SHIFT-DR states. During all other states, TDI is a “don’t care.”

Relation to Other Signals

Pin Symbol	Relation to Other Signals
TCK	TDI is sampled on the rising edge of TCK.
TDO	In the SHIFT-IR and SHIFT-DR TAP controller states, TDO contains the output data of the register being shifted, and TDI provides the input.
TMS	TDI is sampled only in the SHIFT-IR and SHIFT DR states (controlled by TMS).

5.1.65 TDO

TDO	Test Data Output
	Outputs serial test data and instructions.
	Output

Signal Description

This is the serial output of the Boundary Scan test logic. TAP instructions and data are shifted out of the embedded Pentium processor on the TDO pin on the falling edge of TCK when the TAP controller is in the SHIFT-IR and SHIFT-DR states. During all other states, the TDO pin is driven to the high impedance state to allow connecting TDO of different devices in parallel.

When Driven

TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times, TDO is driven to the high impedance state. TDO does not float during bus HOLD or BOFF#.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
TCK	TDO is driven on the falling edge of TCK.
TDI	In the SHIFT-IR and SHIFT-DR TAP controller states, TDI provides the input data to the register being shifted, and TDO provides the output.
TMS	TDO is driven only in the SHIFT-IR and SHIFT DR states (controlled by TMS).

5.1.66 TMS

TMS	Test Mode Select
	Controls TAP controller state transitions.
	Synchronous Input to TCK

Signal Description

This a Boundary Scan test logic control input. The value of this input signal sampled at the rising edge of TCK controls the sequence of TAP controller state changes.

To ensure deterministic behavior of the TAP controller, TMS is provided with an internal pullup resistor. If boundary scan is not used, TMS may be tied to V_{CC} or left unconnected.

When Sampled

TMS is sampled on every rising edge of TCK.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
TCK	TMS is sampled on every rising edge of TCK.
TDI	TDI is sampled only in the SHIFT-IR and SHIFT DR states (controlled by TMS).
TDO	TDO is driven only in the SHIFT-IR and SHIFT DR states (controlled by TMS).

5.1.67 TRST#

TRST#	Test Reset
	Allows the TAP controller to be asynchronously initialized.
	Asynchronous Input

Signal Description

This is a Boundary Scan test logic reset or initialization pin. When asserted, it allows the TAP controller to be asynchronously initialized. When asserted, TRST# will force the TAP controller into the Test Logic Reset State. When in this state, the test logic is disabled so that normal operation of the device can continue unhindered. During initialization, the embedded Pentium processor initializes the instruction register with the IDCODE instruction.

An alternate method of initializing the TAP controller is to Drive TMS high for at least 5 TCK cycles. In addition, the embedded Pentium processor implements a power on TAP controller reset function. When the embedded Pentium processor is put through its normal power on/RESET function, the TAP controller is automatically reset by the processor. The user does not have to assert the TRST# pin or drive TMS high after the falling edge of RESET.

When Sampled

TRST# is an asynchronous input.

Relation to Other Signals

None

5.1.68 V_{CC}

V_{CC}	Supply Voltage for the processor
	V_{CC} is used to supply power to the embedded Pentium processor.
	Power Input

Signal Description

The embedded Pentium processor requires 3.3 V V_{CC} inputs.

5.1.69 V_{CC2}

V_{CC2}	Core Supply Voltage
	V_{CC2} is used to supply the core of the embedded Pentium processor with MMX technology and the low-power embedded Pentium processor with MMX technology.
	Power Input

Signal Description

The embedded Pentium processor with MMX technology requires a 2.8 V V_{CC2} (core) voltage.

The low-power embedded Pentium processor with MMX technology core voltage V_{CC2} is 1.9 V for the PPGA package. The core voltage V_{CC2} for the HL-PBGA package is 1.8 V (166 MHz) or 2.0 V (266 MHz).

5.1.70 V_{CC3}

V_{CC3}	I/O Supply Voltage
	V_{CC3} is used to supply the I/O of the embedded Pentium processor with MMX technology and the low-power embedded Pentium processor with MMX technology.
	Power Input

Signal Description

The embedded Pentium processor with MMX technology requires a 3.3 V V_{CC3} (I/O) voltage. This enables compatibility with embedded Pentium processor system components.

The low-power embedded Pentium processor with MMX technology requires a 2.5 V V_{CC3} (I/O) voltage.

5.1.71 VCC2DET#

VCC2DET#	V_{CC2} Detect
	VCC2DET# can be used in flexible motherboard implementations to configure the voltage regulator output set-point appropriately for the V_{CC2} inputs of the embedded Pentium® processor with MMX™ technology. This pin can also be used to differentiate between the Pentium Processor with MMX technology and the low-power embedded Pentium processor with MMX technology
	Output

NOTE: This pin is an INC on the embedded Pentium processor.

Signal Description

The embedded Pentium processor with MMX technology requires 2.8 V on the V_{CC2} pins and 3.3 V on the V_{CC3} pins. By using the VCC2DET# signal the system can adjust the core voltage to the processor when a embedded Pentium processor with MMX technology is inserted into Socket 7.

VCC2DET# is driven active (low) to indicate that a embedded Pentium processor with MMX technology is installed in the system and can be used in flexible motherboard designs to configure the voltage regulator output set-point appropriately for the V_{CC2} inputs of the embedded Pentium processor with MMX technology.

This pin can be used to differentiate between the Pentium Processor with MMX technology and the low-power embedded Pentium processor with MMX technology. This is an Internal No Connect (INC) pin on the low-power embedded Pentium processor with MMX technology. This pin is not defined on the HL-PBGA package.

When Sampled/Driven

This pin is internally strapped to V_{SS}.

5.1.72 W/R#

W/R#	Write/Read
	Distinguishes a Write cycle from a Read cycle.
	Synchronous Input/Output

Signal Description

The Write/Read signal is one of the primary bus cycle definition pins. W/R# distinguishes between write (W/R# = 1) and read cycles (W/R# = 0).

When operating in dual processing mode, the embedded Pentium processor uses this signal for private snooping.

When Sampled/Driven

W/R# is driven valid in the same clock as ADS# and the cycle address. It remains valid from the clock in which ADS# is asserted until the earlier of the last BRDY# or the clock after NA#.

This signal becomes an input/output when two embedded Pentium processors are operating together in dual processing mode.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
ADS#	W/R# is driven to its valid state with ADS#.
BOFF#	W/R# floats one clock after BOFF# is asserted.
HLDA	W/R# floats when HLDA is asserted.
KEN#	KEN# determines cacheability only if W/R# indicates a read.

5.1.73 WB/WT#

WB/WT#	Writeback/Writethrough
	This pin allows a cache line to be defined as writeback or writethrough on a line by line basis.
	Synchronous Input

Signal Description

This pin allows a cache line to be defined as writeback or writethrough on a line by line basis. As a result, in conjunction with the PWT pin, it controls the MESI state in which the line is saved.

If WB/WT# is sampled high during a memory read cycle and the PWT pin is low, the line is saved in the Exclusive (E) state in the cache. If WB/WT# is sampled low during a memory read cycle, the line is saved in the Shared (S) state in the cache.

If WB/WT# is sampled high during a write to a shared line in the cache and the PWT pin is low, the line transitions to the E state. If WB/WT# is sampled low during a write to a shared line in the cache, the line remains in the S state.

If for either reads or writes the PWT pin is high, the line is saved in, or remains in, the Shared (S) state.

When Sampled

This pin is sampled with KEN# on the clock in which NA# or the first BRDY# is returned, however it must meet setup and hold times on every clock edge.

Relation to Other Signals

Pin Symbol	Relation to Other Signals
BRDY# NA#	WB/WT# is sampled with the earlier of the first BRDY# or NA# for that cycle.
PWT	If PWT is high, WB/WT# is a “don’t care.”

Bus Functional Description 6

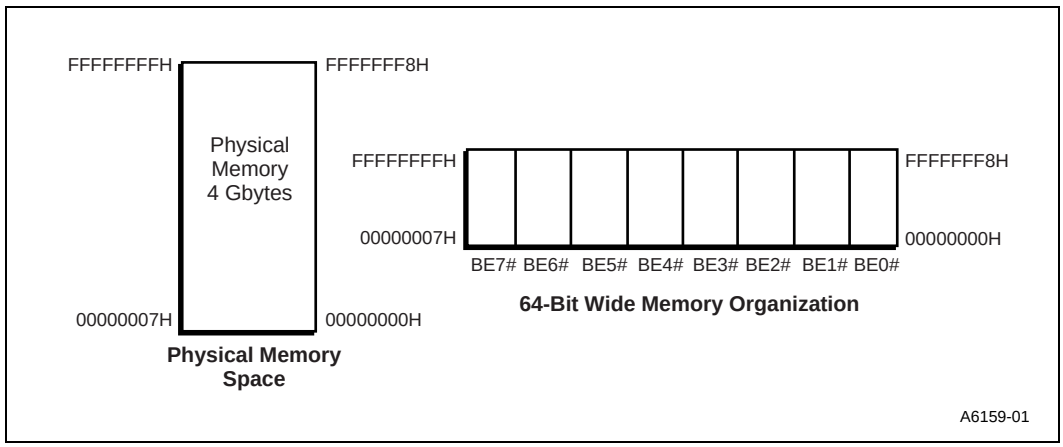
Embedded Pentium® family processors support the same bus functionality. The processor bus supports a 528-Mbyte/s data transfer rate at 66 MHz. All data transfers occur as a result of one or more bus cycles. This chapter describes the processor bus cycles and the processor data transfer mechanism.

6.1 Physical Memory and I/O Interface

Processor memory is accessible in 8-, 16-, 32-, and 64-bit quantities. Processor I/O is accessible in 8-, 16-, and 32-bit quantities. The processor can directly address up to 4 Gbytes of physical memory, and up to 64 Kbytes of I/O.

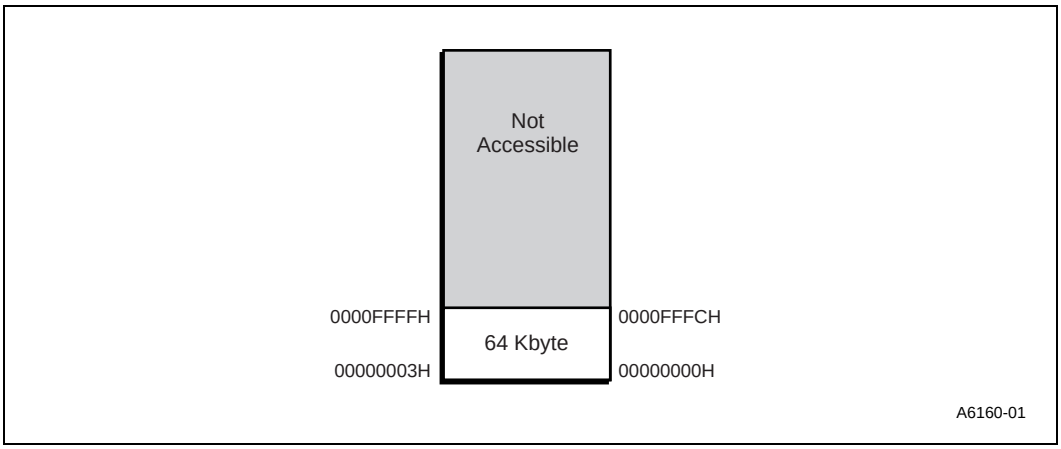
In hardware, memory space is organized as a sequence of 64-bit quantities. Each 64-bit location has eight individually addressable bytes at consecutive memory addresses (see Figure 6-1).

Figure 6-1. Memory Organization



The I/O space is organized as a sequence of 32-bit quantities. Each 32-bit quantity has four individually addressable bytes at consecutive memory addresses. See Figure 6-2 for a conceptual diagram of the I/O space.

Figure 6-2. I/O Space Organization



Sixty-four-bit memories are organized as arrays of physical quadwords (8-byte words). Physical quadwords begin at addresses evenly divisible by 8. The quadwords are addressable by physical address lines A31–A3.

Thirty-two-bit memories are organized as arrays of physical dwords (4-byte words). Physical dwords begin at addresses evenly divisible by 4. The dwords are addressable by physical address lines A31–A3 and A2. A2 can be decoded from the byte enables according to Table 6-2.

Sixteen-bit memories are organized as arrays of physical words (2-byte words). Physical words begin at addresses evenly divisible by two. The words are addressable by physical address lines A31–A3, A2–A1, BHE#, and BLE#. A2 and A1 can be decoded from the byte enables according to Table 6-2, BHE# and BLE# can be decoded from the byte enables according to Table 6-3 and Table 6-4.

To address 8-bit memories, the lower three address lines (A2–A0) must be decoded from the byte enables as indicated in Table 6-2.

6.2 Data Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word, dword, and quadword lengths may be transferred. Data may be accessed at any byte boundary, but two cycles may be required for misaligned data transfers. The processor considers a 2-byte or 4-byte operand that crosses a 4-byte boundary to be misaligned. In addition, an 8-byte operand that crosses an 8-byte boundary is misaligned.

Like the Intel486™ processor, the processor address signals are split into two components. High-order address bits are provided by the address lines A31–A3. The byte enables BE7#–BE0# form the low-order address and select the appropriate byte of the 8-byte data bus.

The byte enable outputs are asserted when their associated data bus bytes are involved with the present bus cycle as shown in Table 6-1. For both memory and I/O accesses, the byte enable outputs indicate which of the associated data bus bytes are driven valid for write cycles and on which bytes data is expected back for read cycles. Non-contiguous byte enable patterns never occur.

Address bits A2–A0 of the physical address can be decoded from the byte enables according to Table 6-2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable) to address 16-bit memory systems (see Table 6-3 and Table 6-4).

Table 6-1. Embedded Pentium® Processor Byte Enables and Associated Data Bytes

Byte Enable Signal	Associated Data Bus Signals
BE0#	D0–D7 (byte 0 — least significant)
BE1#	D8–D15 (byte 1)
BE2#	D16–D23 (byte 2)
BE3#	D24–D31 (byte 3)
BE4#	D32–D39 (byte 4)
BE5#	D40–D47 (byte 5)
BE6#	D48–D55 (byte 6)
BE7#	D56–D63 (byte 7 — most significant)

Address bits A2–A0 of the physical address can be decoded from the byte enables according to Table 6-2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable) to address 16-bit memory systems (see Table 6-3 and Table 6-4).

Table 6-2. Generating A2–A0 from BE7#–BE0#

A2	A1	A0	BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#
0	0	0	X	X	X	X	X	X	X	Low
0	0	1	X	X	X	X	X	X	Low	High
0	1	0	X	X	X	X	X	Low	High	High
0	1	1	X	X	X	X	Low	High	High	High
1	0	0	X	X	X	Low	High	High	High	High
1	0	1	X	X	Low	High	High	High	High	High
1	1	0	X	Low	High	High	High	High	High	High
1	1	1	Low	High	High	High	High	High	High	High

Table 6-3. When BLE# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BLE#
X	X	X	X	X	X	X	Low	Low
X	X	X	X	X	Low	High	High	Low
X	X	X	Low	High	High	High	High	Low
X	Low	High	High	High	High	High	High	Low

Table 6-4. When BHE# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BHE#
X	X	X	X	X	X	Low	X	Low
X	X	X	X	Low	X	High	High	Low
X	X	Low	X	High	High	High	High	Low
Low	X	High	High	High	High	High	High	Low

Table 6-5. When BE3'# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BE3'#
Low	X	X	X	Low	X	X	X	Low

Table 6-6. When BE2'# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BE2'#
X	Low	X	X	X	Low	X	X	Low

Table 6-7. When BE1'# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BE1'#
X	X	Low	X	X	X	Low	X	Low

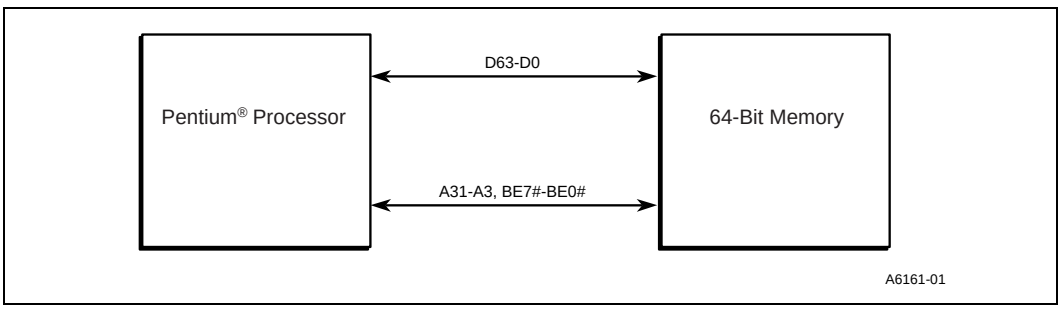
Table 6-8. When BE0'# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BE0'#
X	X	X	Low	X	X	X	Low	Low

6.2.1 Interfacing With 8-, 16-, 32-, and 64-Bit Memories

In 64-bit physical memories such as Figure 6-3, each 8-byte quadword begins at a byte address that is a multiple of eight. A31–A3 are used as an 8-byte quadword select and BE7#–BE0# select individual bytes within the word.

Figure 6-3. Embedded Pentium® Processor with 64-Bit Memory



Memories that are 32 bits wide require external logic for generating A2 and BE3'#–BE0'#. Memories that are 16 bits wide require external logic for generating A2, A1, BHE# and BLE#. Memories that are 8 bits wide require external logic for generating A2, A1, and A0. All memory systems that are less than 64 bits wide require external byte swapping logic for routing data to the appropriate data lines.

The processor expects all the data requested by the byte enables to be returned as one transfer (with one BRDY#), so byte assembly logic is required to return all requested bytes to the processor at one time. Note that the processor does not support BS8#, BS16# or BS32#, so this logic must be implemented externally if necessary.

Figure 6-4 shows the processor address bus interface to 64, 32, 16 and 8-bit memories. Address bits A2, A1, and A0 and BHE#, BLE#, and BE3'#–BE0'# are decoded as shown in Table 6-2 through Table 6-8.

Figure 6-4. Addressing 32-, 16- and 8-Bit Memories

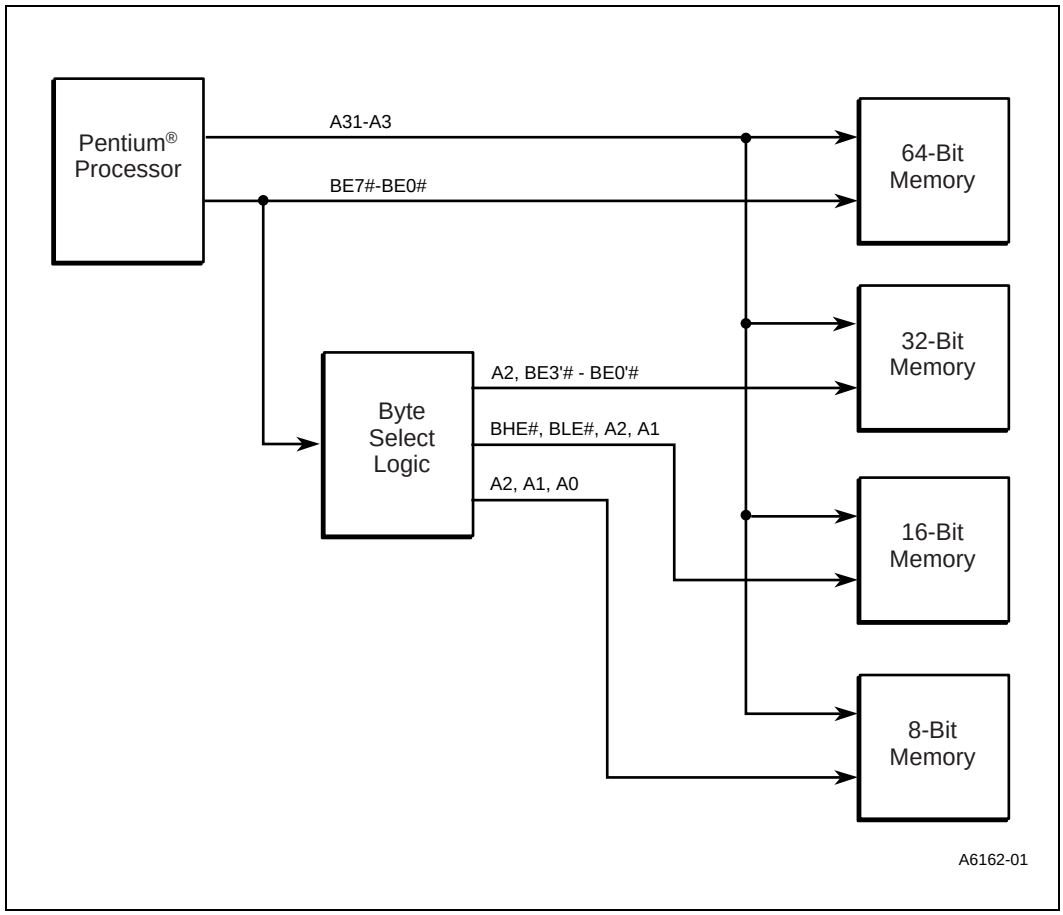
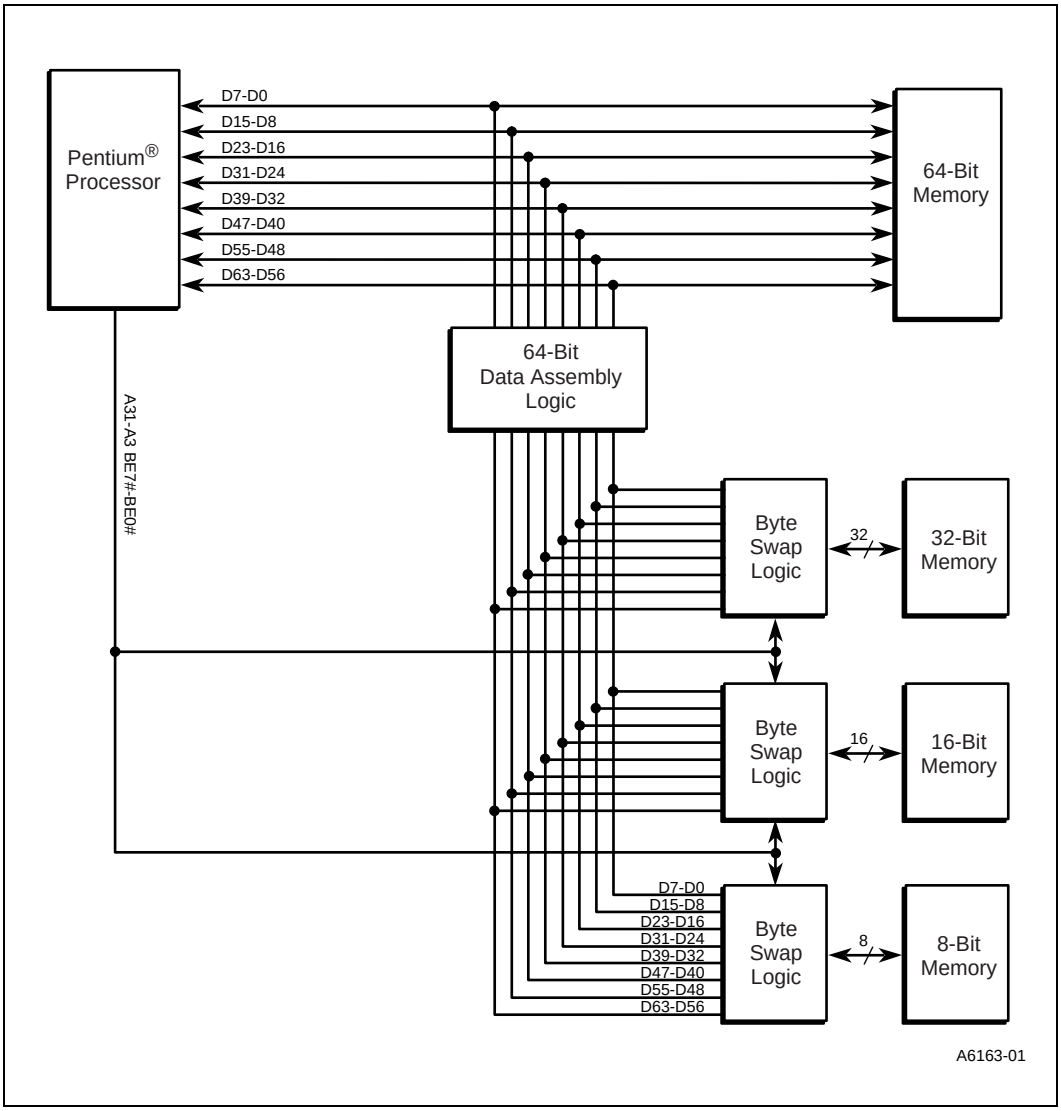


Figure 6-5 shows the processor data bus interface to 32-, 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to and received from the processor on the correct data pins (see Table 6-1). For memory widths smaller than 64 bits, byte assembly logic is needed to return all bytes of data requested by the processor in one cycle.

Figure 6-5. Data Bus Interface to 32-, 16- and 8-Bit Memories



Operand alignment and size dictate when two cycles are required for a data transfer. Table 6-9 shows the transfer cycles generated by the processor for all combinations of logical operand lengths and alignment and applies to both locked and unlocked transfers. When multiple cycles are required to transfer a multi-byte logical operand, the highest order bytes are transferred first.

Table 6-9. Transfer Bus Cycles for Bytes, Words, Dwords and Quadwords

Length of Transfer	1 Byte	2 Bytes							
Low Order Address	**xxx**	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
1st transfer	b	w	w	w	hb	w	w	w	hb
Byte enables driven	0	BE0–1#	BE1–2#	BE2–3#	BE4#	BE4–5#	BE5–6#	BE6–7#	BE0#
Value driven on A3		0	0	0	0	0	0	0	1
2nd transfer (if needed)					lb				lb
Byte enables driven					BE3#				BE7#
Value driven on A3					0				0
Length of Transfer	**4 Bytes**								
Low Order Address	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**	
1st transfer	d	hb	hw	h3	d	hb	hw	h3	
Byte enables driven	BE0–3#	BE4#	BE4–5#	BE4–6#	BE4–7#	BE0#	BE0–1#	BE0–2#	
Low order address	0	0	0	0	0	1	1	1	
2nd transfer (if needed)		l3	lw	lb		l3	lw	lb	
Byte enables driven		BE1–3#	BE2–3#	BE3#		BE5–7#	BE6–7#	BE7#	
Value driven on A3		0	0	0		0	0	0	
Length of Transfer	**8 Bytes**								
Low Order Address	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**	
1st transfer	q	hb	hw	h3	hd	h5	h6	h7	
Byte enables driven	BE0–7#	BE0#	BE0–1#	BE0–2#	BE0–3#	BE0–4#	BE0–5#	BE0–6#	
Value driven on A3	0	1	1	1	1	1	1	1	
2nd transfer (if needed)		l7	l6	l5	ld	l3	lw	lb	
Byte enables driven		BE1–7#	BE2–7#	BE3–7#	BE4–7#	BE5–7#	BE6–7#	BE7#	
Value driven on A3		0	0	0	0	0	0	0	

Key:

b = byte transfer w = 2-byte transfer 3 = 3-byte transfer d = 4-byte transfer

5 = 5-byte transfer 6 = 6-byte transfer 7 = 7-byte transfer q = 8-byte transfer

h = high order ll = low order

8-byte operand:

high order byte	byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	low order byte

↑ byte with highest address ↑ byte with lowest address

6.3 Bus State Definition

This section describes the processor bus states in detail. See Figure 6-6 for the bus state diagram.

Ti: This is the bus idle state. In this state, no bus cycles are being run. The processor may or may not be driving the address and status pins, depending on the state of the HLDA, AHOLD, and BOFF# inputs. An asserted BOFF# or RESET always forces the state machine back to this state. HLDA is only driven in this state.

T1: This is the first clock of a bus cycle. Valid address and status are driven out and ADS# is asserted. There is one outstanding bus cycle.

T2: This is the second and subsequent clock of the first outstanding bus cycle. In state T2, data is driven out (if the cycle is a write), or data is expected (if the cycle is a read), and the BRDY# pin is sampled. There is one outstanding bus cycle.

T12: This state indicates there are two outstanding bus cycles, and that the processor is starting the second bus cycle at the same time that data is being transferred for the first. In T12, the processor drives the address and status and asserts ADS# for the second outstanding bus cycle, while data is transferred and BRDY# is sampled for the first outstanding cycle.

T2P: This state indicates there are two outstanding bus cycles, and that both are in their second and subsequent clocks. In T2P, data is being transferred and BRDY# is sampled for the first outstanding cycle. The address, status and ADS# for the second outstanding cycle were driven sometime in the past (in state T12).

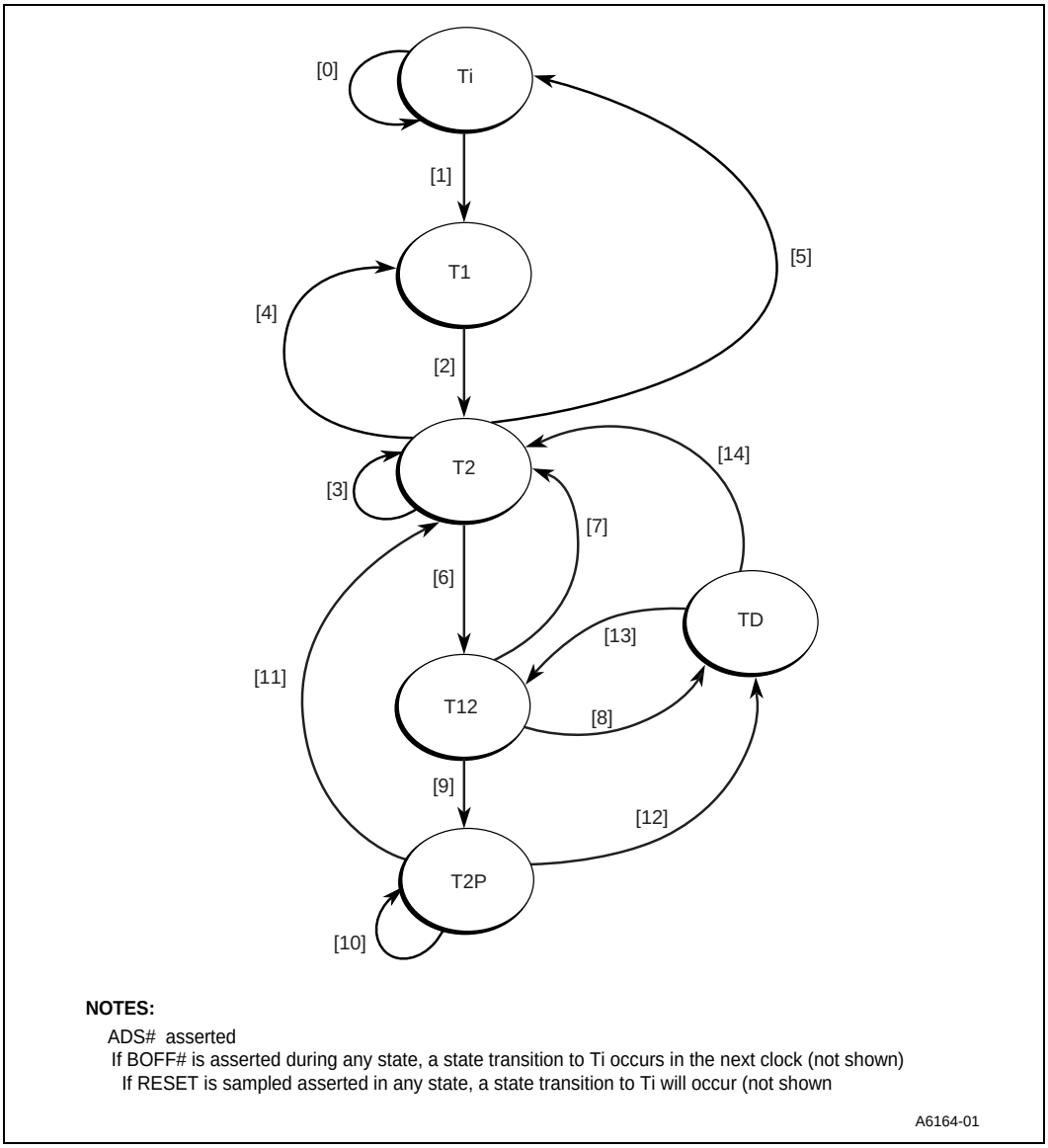
TD: This state indicates there is one outstanding bus cycle, that its address, status and ADS# have already been driven sometime in the past (in state T12), and that the data and BRDY# pins are not being sampled because the data bus requires one dead clock to turn around between consecutive reads and writes, or writes and reads. The processor enters TD if in the previous clock there were two outstanding cycles, the last BRDY# was returned, and a dead clock is needed. The timing diagrams in the next section give examples when a dead clock is needed.

Table 6-10 gives a brief summary of bus activity during each bus state. Figure 6-6 shows the processor bus state diagram.

Table 6-10. Processor Bus Activity

Bus State	Cycles Outstanding	ADS# Asserted New Address Driven	BRDY# Sampled Data Transferred
Ti	0	No	No
T1	1	Yes	No
T2	1	No	Yes
T12	2	Yes	Yes
T2P	2	No	Yes
TD	1	No	No

Figure 6-6. Processor Bus Control State Machine



6.3.1 State Transitions

The state transition equations with descriptions are listed below. In the equations, “&” means logical AND, “+” means logical OR, and “#” placed after label means active low. The NA# used here is actually a delayed version of the external NA# pin (delayed by one clock). The definition of request pending is:

- The processor has generated a new bus cycle internally & HOLD (delayed by one clock) negated & BOFF# negated & (AHOLD negated + HITM# asserted).

Note that once NA# is sampled asserted, the processor latches NA#. The processor pipelines a cycle when one becomes pending even if NA# is subsequently deasserted.

(0)	No Request Pending	
(1)	Request Pending:	The processor starts a new bus cycle & ADS# is asserted in the T1 state.
(2)	Always:	With BOFF# negated, and a cycle outstanding the processor always moves to T2 to process the data transfer.
(3)	Not Last BRDY# & (No Request Pending + NA# Negated):	The processor stays in T2 until the transfer is over if no new request becomes pending or if NA# is not asserted.
(4)	Last BRDY# & Request Pending & NA# Sampled Asserted:	If there is a new request pending when the current cycle is complete, and if NA# was sampled asserted, the processor begins from T1.
(5)	Last BRDY# & (No Request Pending + NA# Negated):	If no cycle is pending when the processor finishes the current cycle or NA# is not asserted, the processor goes back to the idle state.
(6)	Not Last BRDY# & Request Pending & NA# Sampled Asserted:	While the processor is processing the current cycle (one outstanding cycle), if another cycle becomes pending and NA# is asserted, the processor moves to T12 indicating that the processor now has two outstanding cycles. ADS# is asserted for the second cycle.
(7)	Last BRDY# & No dead clock:	When the processor finishes the current cycle, and no dead clock is needed, it goes to the T2 state.
(8)	Last BRDY# & Need a dead clock:	When the processor finishes the current cycle and a dead clock is needed, it goes to the TD state.
(9)	Not Last BRDY#:	With BOFF# negated, and the current cycle not complete, the processor always moves to T2P to process the data transfer.
(10)	Not Last BRDY#:	The processor stays in T2P until the first cycle transfer is over.
(11)	Last BRDY# & No dead clock:	When the processor finishes the first cycle and no dead clock is needed, it goes to T2 state.
(12)	Last BRDY# & Need a dead clock:	When the first cycle is complete, and a dead clock is needed, it goes to TD state.
(13)	Request Pending & NA# sampled asserted:	If NA# was sampled asserted and there is a new request pending, it goes to T12 state.
(14)	No Request Pending + NA# Negated:	If there is no new request pending, or NA# was not asserted, it goes to T2 state.

6.4 Bus Cycles

The following terminology is used in this document to describe the processor bus functions. The processor requests data transfer cycles, bus cycles, and bus operations. A *data transfer cycle* is one data item, up to 8 bytes in width, being returned to the processor or accepted from the processor with BRDY# asserted. A *bus cycle* begins with the embedded Pentium processor driving an address and status and asserting ADS#, and ends when the last BRDY# is returned. A bus cycle may have 1 or 4 data transfers. A *burst cycle* is a bus cycle with 4 data transfers. A *bus operation* is a sequence of bus cycles to carry out a specific function, such as a locked read-modify-write or an interrupt acknowledge.

“Bus State Definition” on page 6-8 describes each of the bus states, and shows the bus state diagram.

Table 6-11 lists all of the bus cycles that are generated by the processor. Note that inquire cycles (initiated by EADS#) may be generated from the system to the processor.

Table 6-11. Processor Initiated Bus Cycles

M/IO#	D/C#	W/R#	CACHE#†	KEN#	Cycle Description	# of Transfers
0	0	0	1	x	Interrupt Acknowledge (2 locked cycles)	1 transfer each cycle
0	0	1	1	x	Special Cycle (Table 6-13)	1
0	1	0	1	x	I/O Read, 32-bits or less, non-cacheable	1
0	1	1	1	x	I/O Write, 32-bits or less, non-cacheable	1
1	0	0	1	x	Code Read, 64-bits, non-cacheable	1
1	0	0	x	1	Code Read, 64-bits, non-cacheable	1
1	0	0	0	0	Code Read, 256-bit burst line fill	4
1	0	1	x	x	Intel Reserved (is not driven by the processor)	n/a
1	1	0	1	x	Memory Read, 64 bits or less, non-cacheable	1
1	1	0	x	1	Memory Read, 64 bits or less, non-cacheable	1
1	1	0	0	0	Memory Read, 256-bit burst line fill	4
1	1	1	1	x	Memory Write, 64 bits or less, non-cacheable	1
1	1	1	0	x	256-bit Burst Writeback	4

† CACHE# is not asserted for any cycle in which M/IO# is driven low or for any cycle in which PCD is driven high.

Note that all burst reads are cacheable, and all cacheable read cycles are bursted. There are no non-cacheable burst reads or non-burst cacheable reads.

The remainder of this chapter describes all of the above bus cycles in detail. In addition, locked operations and bus cycle pipelining is discussed.

6.4.1 Single-Transfer Cycle

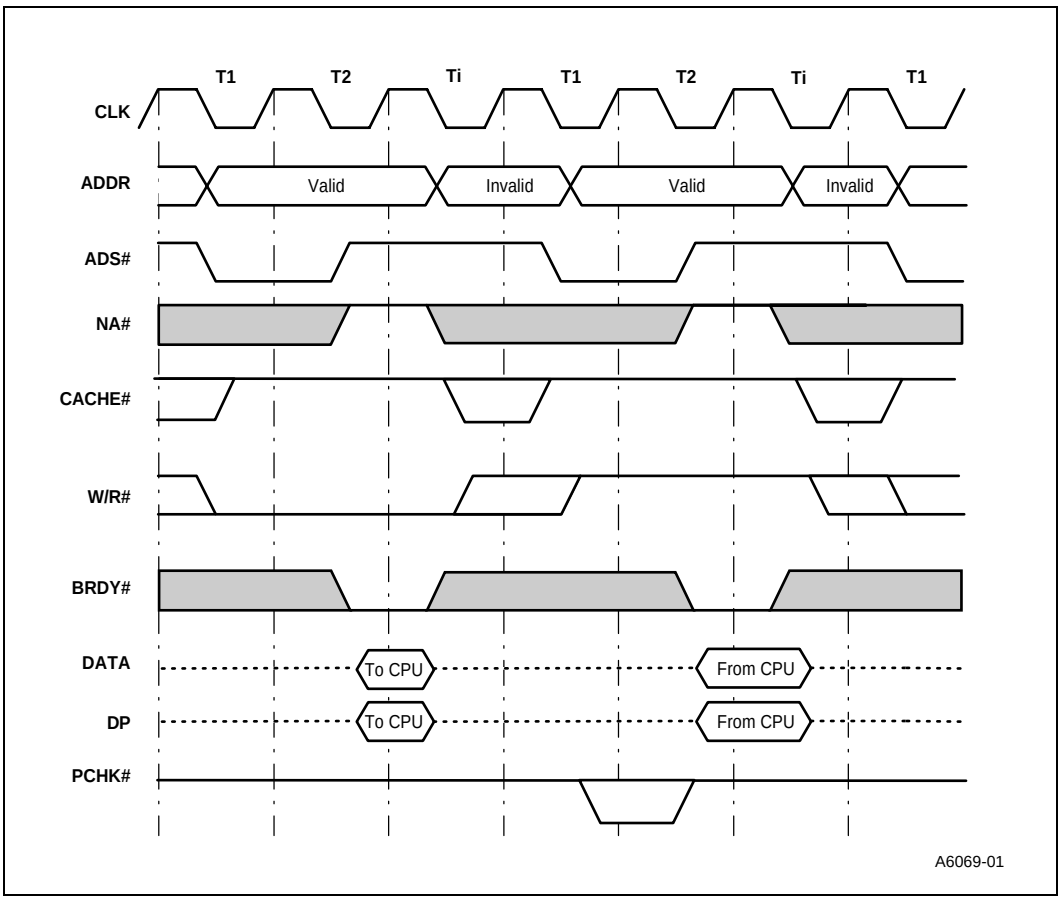
The processor supports a number of different types of bus cycles. The simplest type of bus cycle is a single-transfer non-cacheable 64-bit cycle, either with or without wait states. Non-pipelined read and write cycles with 0 wait states are shown in Figure 6-7.

The processor initiates a cycle by asserting the address status signal (ADS#) in the first clock. The clock in which ADS# is asserted is by definition the first clock in the bus cycle. The ADS# output indicates that a valid bus cycle definition and address is available on the cycle definition pins and the address bus. The CACHE# output is deasserted (high) to indicate that the cycle is a single transfer cycle.

For a zero wait state transfer, BRDY# is returned by the external system in the second clock of the bus cycle. BRDY# indicates that the external system has presented valid data on the data pins in response to a read or the external system has accepted data in response to a write. The processor samples the BRDY# input in the second and subsequent clocks of a bus cycle (the T2, T12 and T2P bus states; see the "Bus State Definition" on page 6-8 for more information).

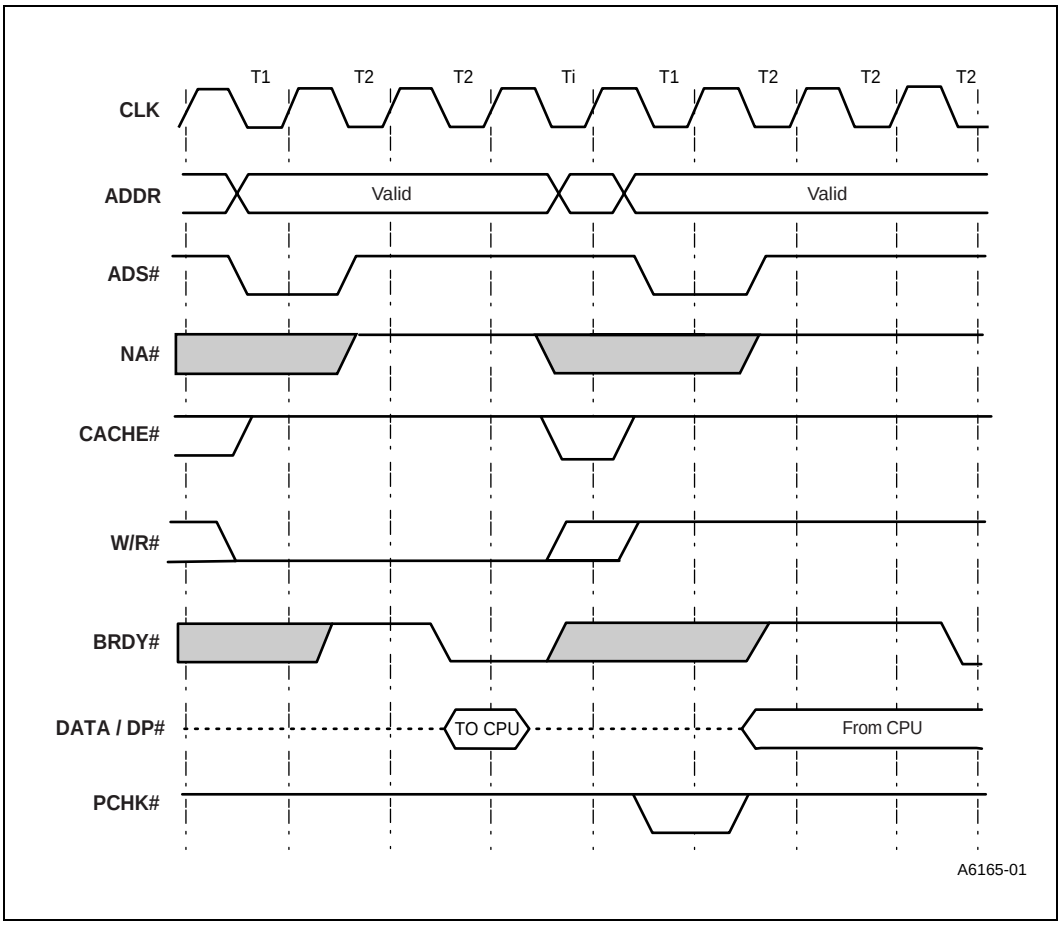
The timing of the data parity input, DP, and the parity check output, PCHK#, is also shown in Figure 6-7. DP is driven by the processor and returned to the processor in the same clock as the data. PCHK# is driven two clocks after BRDY# is returned for reads with the results of the parity check.

Figure 6-7. Non-Pipelined Read and Write



If the system is not ready to drive or accept data, wait states can be added to these cycles by not returning BRDY# to the processor at the end of the second clock. Cycles of this type, with one and two wait states added are shown in Figure 6-8. Note that BRDY# must be driven inactive at the end of the second clock. Any number of wait states can be added to processor bus cycles by maintaining BRDY# inactive.

Figure 6-8. Non-Pipelined Read and Write with Wait States



6.4.2 Burst Cycles

For bus cycles that require more than a single data transfer (cacheable cycles and writeback cycles), the processor uses the burst data transfer. In burst transfers, a new data item can be sampled or driven by the processor in consecutive clocks. In addition the addresses of the data items in burst cycles all fall within the same 32-byte aligned area (corresponding to an internal processor cache line).

The implementation of burst cycles is via the BRDY# pin. While running a bus cycle of more than one data transfer, the processor requires that the memory system perform a burst transfer and follow the burst order (see Table 6-12). Given the first address in the burst sequence, the address of subsequent transfers must be calculated by external hardware. This requirement exists because the processor address and byte-enables are asserted for the first transfer and are not re-driven for each transfer. The burst sequence is optimized for two bank memory subsystems and is shown in Table 6-12.

Table 6-12. Processor Burst Order

1st Address	2nd Address	3rd Address	4th Address
0	8	10	18
8	0	18	10
10	18	0	8
18	10	8	0

NOTE: The addresses are represented in hexadecimal format

The cycle length is driven by the processor together with cycle specification (see Table 6-11), and the system should latch this information and terminate the cycle on time with the appropriate number of transfers. The fastest burst cycle possible requires two clocks for the first data item to be returned/driven with subsequent data items returned/driven every clock.

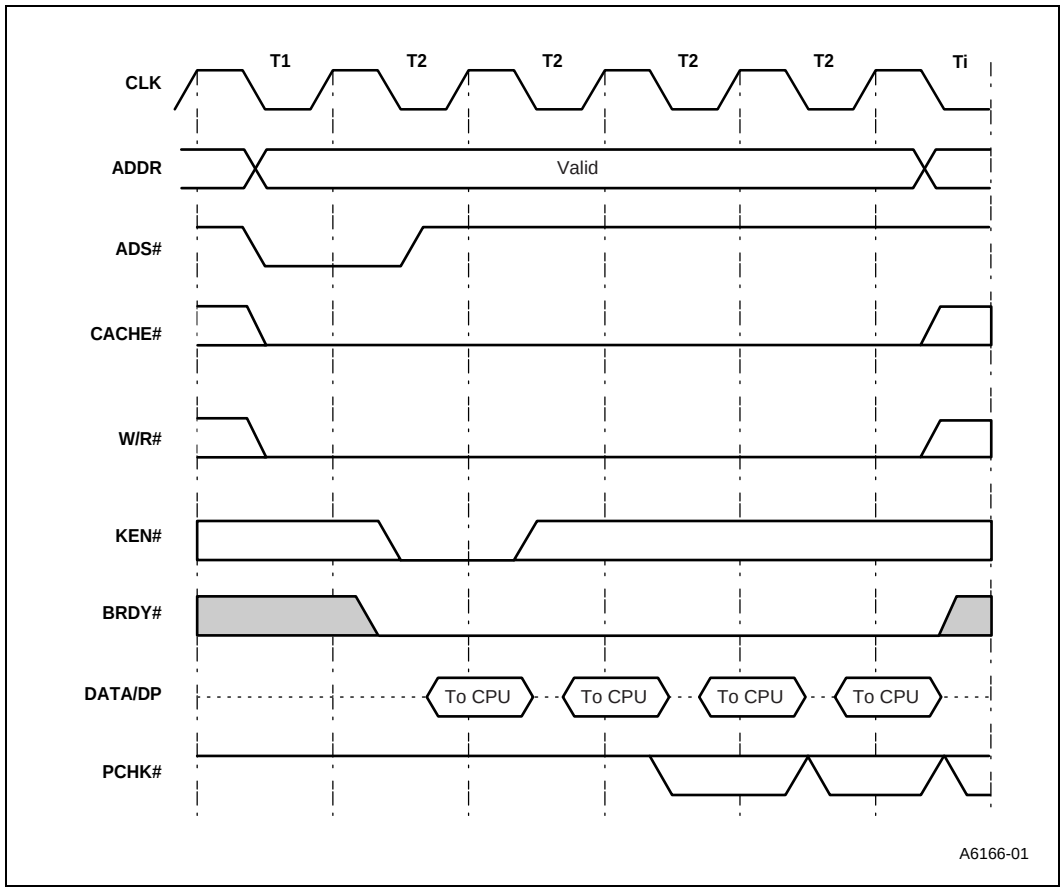
6.4.2.1 Burst Read Cycles

When initiating any read, the processor presents the address and byte enables for the data item requested. When the cycle is converted into a cache linefill, the first data item returned should correspond to the address sent out by the processor; however, the byte enables should be ignored, and valid data must be returned on all 64 data lines. In addition, the address of the subsequent transfers in the burst sequence must be calculated by external hardware since the address and byte enables are not re-driven for each transfer.

Figure 6-9 shows a cacheable burst read cycle. Note that in this case the initial cycle generated by the processor might have been satisfied by a single data transfer, but was transformed into a multiple-transfer cache fill by KEN# being returned active on the clock that the first BRDY# is returned. In this case KEN# has such an effect because the cycle is internally cacheable in the processor (CACHE# pin is driven active). KEN# is only sampled once during a cycle to determine cacheability.

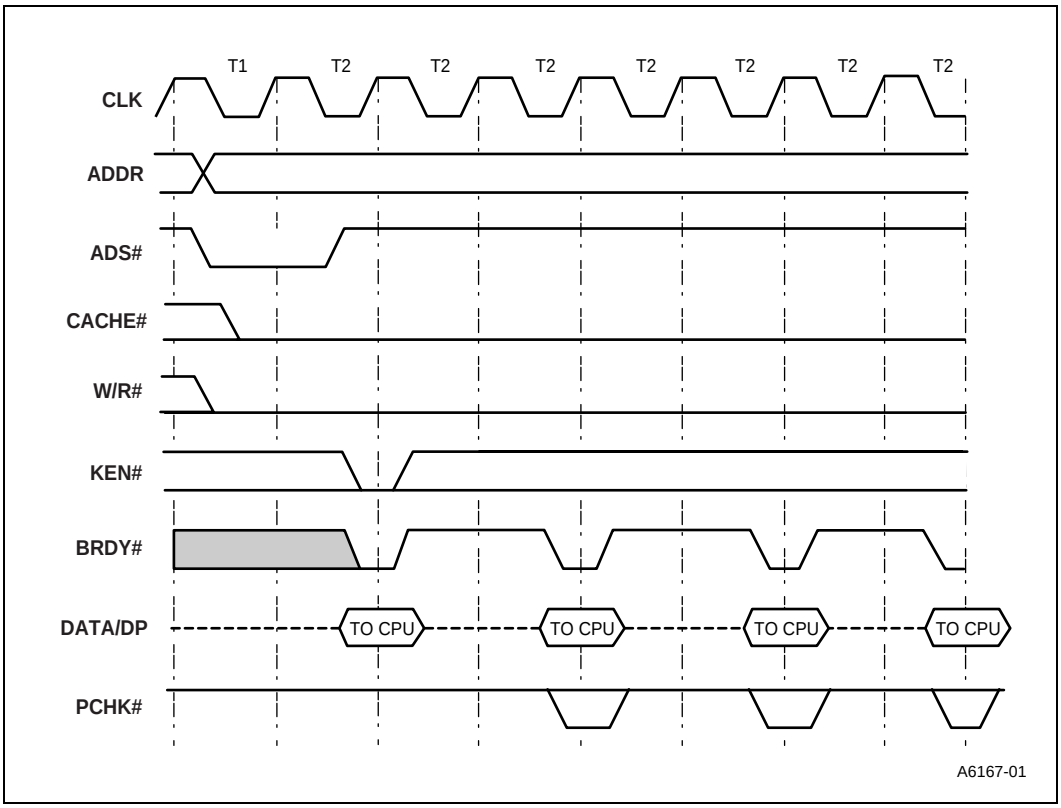
PCHK# is driven with the parity check status two clocks after each BRDY#.

Figure 6-9. Basic Burst Read Cycle



Data is sampled only in the clock that BRDY# is returned, which means that data need not be sent to the processor every clock in the burst cycle. An example burst cycle where two clocks are required for every burst item is shown in Figure 6-10.

Figure 6-10. Slow Burst Read Cycle



6.4.2.2 Burst Write Cycles

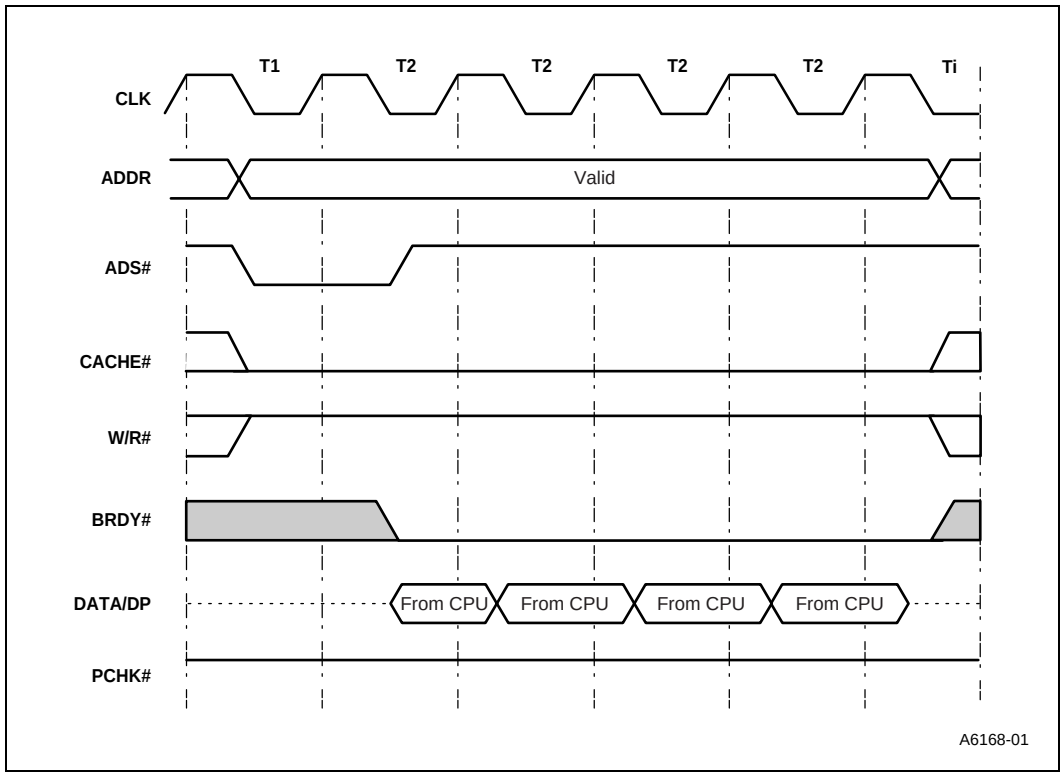
Figure 6-11 shows the timing diagram of basic burst write cycle. KEN# is ignored in burst write cycle. If the CACHE# pin is active (low) during a write cycle, it indicates that the cycle is a burst writeback cycle. Burst write cycles are always writebacks of modified lines in the data cache. Writeback cycles have several causes:

1. Writeback due to replacement of a modified line in the data cache.
2. Writeback due to an inquire cycle that hits a modified line in the data cache.
3. Writeback due to an internal snoop that hits a modified line in the data cache.
4. Writebacks caused by asserting the FLUSH# pin.
5. Writebacks caused by executing the WBINVD instruction.

Writeback cycles are described in more detail in the Inquire Cycle section of this chapter.

The only write cycles that are burstable by the processor are writeback cycles. All other write cycles are 64 bits or less, single transfer bus cycles.

Figure 6-11. Basic Burst Write Cycle



For writeback cycles, the lower five bits of the first burst address always starts at zero; therefore, the burst order becomes 0, 8H, 10H, and 18H. Again, note that the address of the subsequent transfers in the burst sequence must be calculated by external hardware since the processor does not drive the address and byte enables for each transfer.

6.4.3 Locked Operations

The embedded Pentium processor family architecture provides a facility to perform atomic accesses of memory. For example, a programmer can change the contents of a memory-based variable and be assured that the variable was not accessed by another bus master between the read of the variable and the update of that variable. This functionality is provided for select instructions using a LOCK prefix, and also for instructions which implicitly perform locked read modify write cycles such as the XCHG (exchange) instruction when one of its operands is memory based. Locked cycles are also generated when a segment descriptor or page table entry is updated and during interrupt acknowledge cycles.

In hardware, the LOCK functionality is implemented through the LOCK# pin, which indicates to the outside world that the processor is performing a read-modify-write sequence of cycles, and that the processor should be allowed atomic access for the location that was accessed with the first locked cycle. Locked operations begin with a read cycle and end with a write cycle. Note that the data width read is not necessarily the data width written. For example, for descriptor access bit updates the processor fetches eight bytes and writes one byte.

A locked operation is a combination of one or multiple read cycles followed by one or multiple write cycles. Programmer generated locked cycles and locked page table/directory accesses are treated differently and are described in the following sections.

6.4.3.1 Programmer Generated Locks and Segment Descriptor Updates

For programmer generated locked operations and for segment descriptor updates, the sequence of events is determined by whether or not the accessed line is in the internal cache and what state that line is in.

Cached Lines in the Modified (M) State

Before a programmer initiated locked cycle or a segment descriptor update is generated, the processor first checks if the line is in the Modified (M) state. If it is, the processor drives an unlocked writeback first, leaving the line in the Invalid (I) state, and then runs the locked read on the external bus. Since the operand may be misaligned, it is possible that the processor may do two writeback cycles before starting the first locked read. In the misaligned scenario the sequence of bus cycles is: writeback, writeback, locked read, locked read, locked write, then the last locked write. Note that although a total of six cycles are generated, the LOCK# pin is active only during the last four cycles. In addition, the SCYC pin is asserted during the last four cycles to indicate that a misaligned lock cycle is occurring. In the aligned scenario the sequence of cycles is: writeback, locked read, locked write. The LOCK# pin is asserted for the last two cycles (SCYC is not asserted, indicating that the locked cycle is aligned). The cache line is left in the Invalid state after the locked operation.

Non-Cached (I-State), S-State and E-State Lines

A programmer initiated locked cycle or a segment descriptor update to an I, S, or E -state line is always forced out to the bus and the line is transitioned to the Invalid state. Since the line is not in the M-State, no writeback is necessary. Because the line is transitioned to the Invalid state, the locked write is forced out to the bus also. The cache line is left in the Invalid state after the locked operation.

6.4.3.2 Page Table/directory Locked Cycles

In addition to programmer generated locked operations, the processor performs locked operations to set the dirty and accessed bits in page tables/page directories. The processor runs the following sequence of bus cycles to set the dirty/accessed bit.

Cached Lines in the Modified (M) State

If there is a TLB miss, the processor issues an (unlocked) read cycle to determine if the dirty or accessed bits need to be set. If the line is modified in the internal data cache, the line is written back to memory (lock not asserted). If the dirty or accessed bits need to be set, the processor then issues a locked read-modify-write operation. The sequence of bus cycles to set the dirty or accessed bits in a page table/directory when the line is in the M-state is: unlocked read, unlocked writeback, locked read, then locked write. The line is left in the Invalid state after the locked operation. Note that accesses to the page tables/directories will not be misaligned.

Non-Cached (I-State), S-State and E-State Lines

If the line is in the I, S or E-state, the locked cycle is always forced out to the bus and the line is transitioned to the Invalid state. The sequence of bus cycles for an internally generated locked operation is locked read, locked write. The line is left in the Invalid state. Note that accesses to the page tables/directories are not misaligned.

6.4.3.3 LOCK# Operation During AHOLD/HOLD/BOFF#

LOCK# is not deasserted if AHOLD is asserted in the middle of a locked cycle.

LOCK# is floated during bus HOLD, but if HOLD is asserted during a sequence of locked cycles, HLDA is not asserted until the locked sequence is complete.

LOCK# floats if BOFF# is asserted in the middle of a locked cycle, and is driven low again when the cycle is restarted. If BOFF# is asserted during the read cycle of a locked read-modify write, the locked cycle is redriven from the read when BOFF# is deasserted. If BOFF# is asserted during the write cycle of a locked read-modify-write, only the write cycle is re-driven when BOFF# is deasserted. The system is responsible for ensuring that other bus masters do not access the operand being locked if BOFF# is asserted during a LOCKed cycle.

6.4.3.4 Inquire Cycles During LOCK#

This section describes the processor bus cycles that occur when an inquire cycle is driven while LOCK# is asserted. Note that inquire cycles are only recognized if AHOLD, BOFF# or HLDA is asserted and the external system returns an external snoop address to the processor. If AHOLD, BOFF# or HLDA is not asserted when EADS# is driven, EADS# is ignored. Note also that an inquire cycle cannot hit the “locked line” because the LOCK cycle invalidated it.

Because HOLD is not acknowledged when LOCK# is asserted, inquire cycles run in conjunction with the assertion of HOLD cannot be driven until LOCK# is deasserted and HLDA is asserted.

BOFF# takes priority over LOCK#. Inquire cycles are permitted while BOFF# is asserted. If an inquire cycle hits a modified line in the data cache, the writeback due to the snoop hit is driven before the locked cycle is re-driven. LOCK# is asserted for the writeback.

An inquire cycle with AHOLD may be run concurrently with a locked cycle. If the inquire cycle hits a modified line in the data cache, the writeback may be driven between the locked read and the locked write. If the writeback is driven between the locked read and write, LOCK# is asserted for the writeback.

Note: Only writebacks due to an external snoop hit to a modified line may be driven between the locked read and the locked write of a LOCKed sequence. No other writebacks (due to an internal snoop hit or data cache replacement) are allowed to invade a LOCKed sequence.

6.4.3.5 LOCK# Timing and Latency

The timing of LOCK# is shown in Figure 6-12. Note that LOCK# is asserted with the ADS# of the read cycle and remains active until the BRDY# of the write cycle is returned. Figure 6-13 shows an example of two consecutive locked operations. Note that the processor automatically inserts at least one idle clock between two consecutive locked operations to allow the LOCK# pin to be sampled inactive by external hardware. Figure 6-14 shows an example of a misaligned locked operation with SCYC asserted.

The maximum number of processor initiated cycles that can be locked together is four. Four cycles are locked together when data is misaligned for programmer generated locks (read, read, write, write). SCYC is asserted for misaligned locked cycles. Note that accesses to the page tables/directories are not misaligned.

Figure 6-12. LOCK# Timing

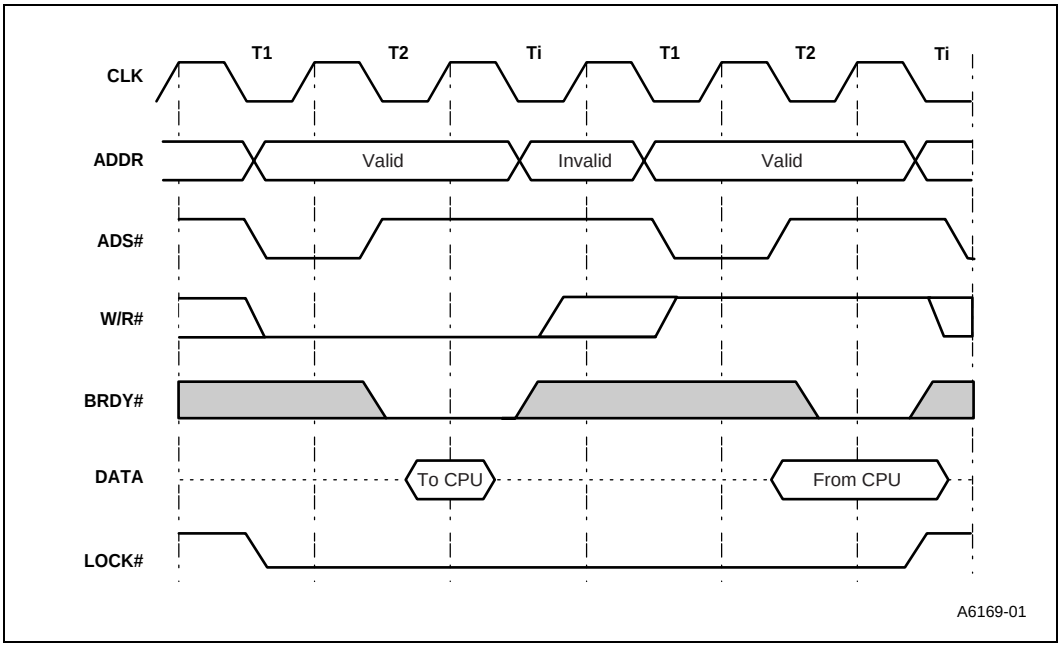


Figure 6-13. Two Consecutive Locked Operations

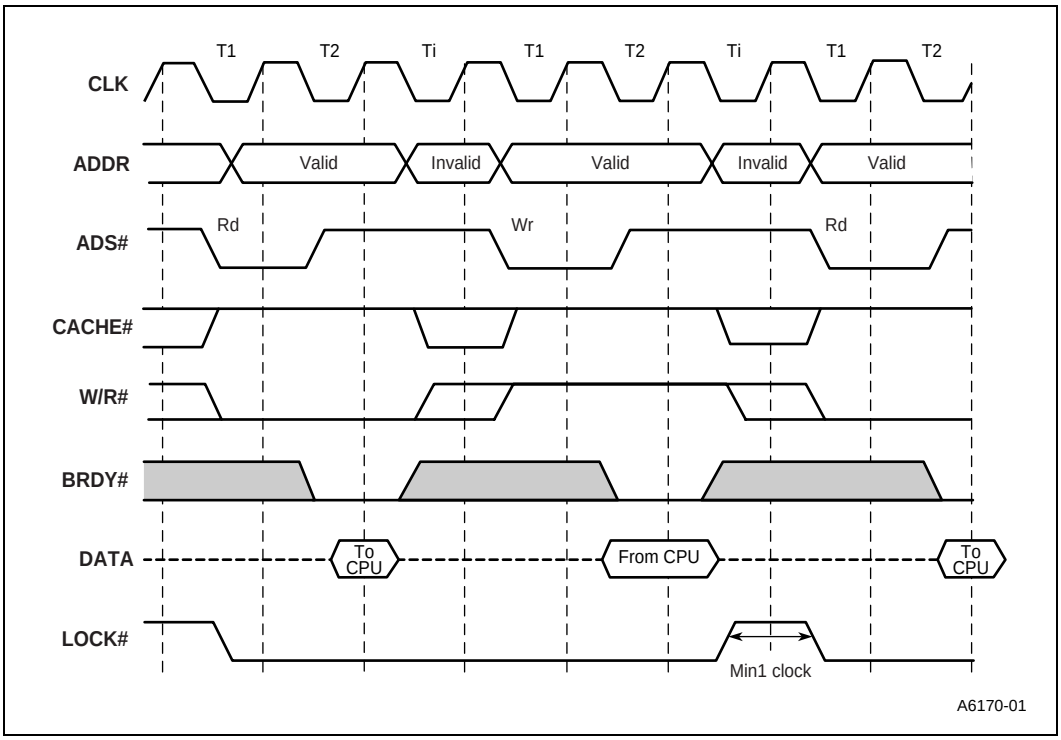
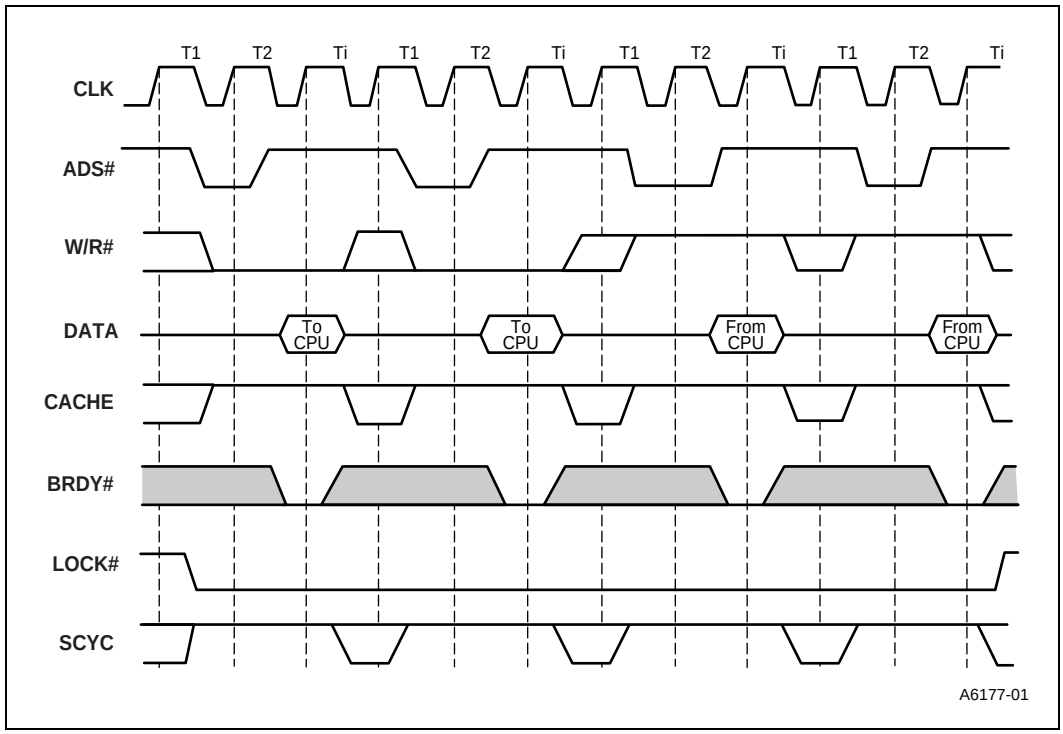


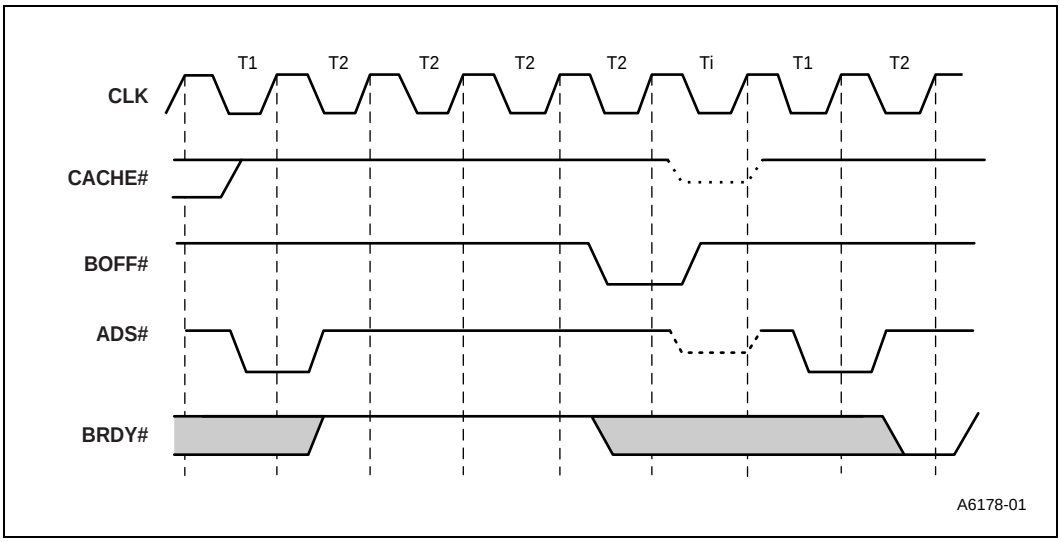
Figure 6-14. Misaligned Locked Cycles



6.4.4 BOFF#

In a multi-master system, another bus master may require the use of the bus to enable the processor to complete its current cycle. The BOFF# pin is provided to prevent this deadlock situation. If BOFF# is asserted, the processor immediately (in the next clock) floats the bus (see Figure 6-15). Any bus cycles in progress are aborted and any data returned to the processor in the clock BOFF# is asserted is ignored. In response to BOFF#, the processor floats the same pins as HOLD, but HLDA is not asserted. BOFF# overrides BRDY#, so if both are sampled active in the same clock, BRDY# is ignored. The processor samples the BOFF# pin every clock.

Figure 6-15. Back Off Timing



The device that asserts BOFF# to the processor is free to run any bus cycle while the processor is in the high impedance state. If BOFF# is asserted after the processor has started a cycle, the new master should wait for memory to return BRDY# before driving a cycle. Waiting for BRDY# provides a handshake to insure that the memory system is ready to accept a new cycle. If the bus is idle when BOFF# is asserted, the new master can start its cycle two clocks after issuing BOFF#. The system must wait two clocks after the assertion of BOFF# to begin its cycle to prevent address bus contention.

The bus remains in the high impedance state until BOFF# is negated. At that time, the processor restarts all aborted bus cycles from the beginning by driving out the address and status and asserting ADS#. Any data returned before BOFF# was asserted is used to continue internal execution, however that data is not placed in an internal cache. Any aborted bus cycles are restarted from the beginning.

External hardware should assure that if the cycle attribute KEN# was returned to the processor (with the first BRDY# or NA#) before the cycle was aborted, it must be returned with the same value after the cycle is restarted. In other words, backoff cannot be used to change the cacheability property of the cycle. The WB/WT# attribute may be changed when the cycle is restarted.

If more than one cycle is outstanding when BOFF# is asserted, the processor restarts both outstanding cycles in their original order. The cycles are not pipelined unless NA# is asserted appropriately.

A pending writeback cycle due to an external snoop hit is reordered in front of any cycles aborted due to BOFF#. For example, if a snoop cycle is run concurrently with a line fill, and the snoop hits an M-state line and then BOFF# is asserted, the writeback cycle due to the snoop is driven from the processor before the cache linefill cycle is restarted.

The system must not rely on the original cycle, that was aborted due to BOFF#, from restarting immediately after BOFF# is deasserted. In addition to reordering writebacks due to external snoop hit in front of cycles that encounter a BOFF#, the processor may also reorder bus cycles in the following situations:

- A pending writeback cycle due to an internal snoop hit is reordered in front of any cycles aborted due to BOFF#. If a read cycle is running on the bus, and an internal snoop of that read cycle hits a modified line in the data cache, and the system asserts BOFF#, the processor drives out a writeback cycle resulting from the internal snoop hit. After completion of the writeback cycle, the processor then restarts the original read cycle. This circumstance can occur during accesses to the page tables/directories, and during prefetch cycles, since these accesses cause a bus cycle to be generated before the internal snoop to the data cache is performed.
- If BOFF# is asserted during a data cache replacement writeback cycle, the writeback cycle is aborted and then restarted once BOFF# is deasserted. However, if the processor encounters a request to access the page table/directory in memory during the BOFF#, this request is reordered in front of the replacement writeback cycle that was aborted due to BOFF#. The processor is first run the sequence of bus cycles to service the page table/directory access and then restart the original replacement writeback cycle.

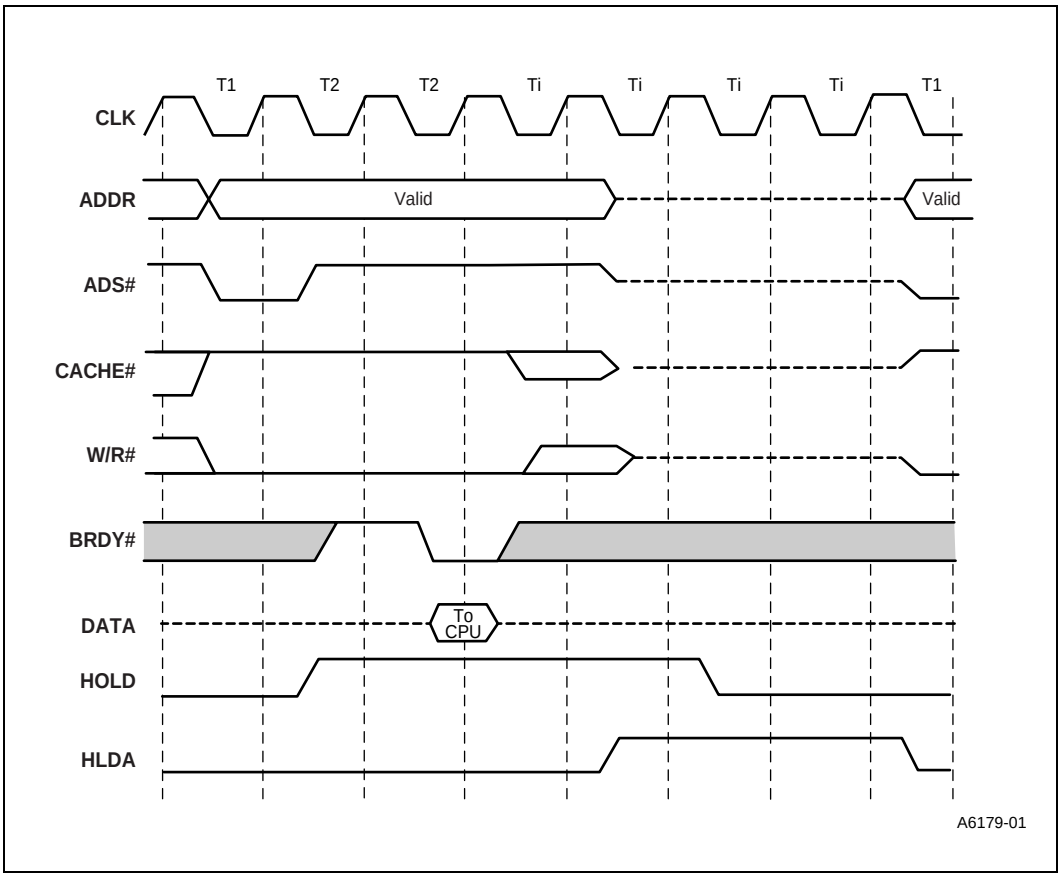
Asserting BOFF# in the same clock as ADS# may cause the processor to leave the ADS# signal floating low. Since ADS# is floating low, a peripheral device may think that a new bus cycle has begun even though the cycle was aborted. There are several ways to approach this situation:

- Design the system's state machines/logic such that ADS# is not recognized the clock after ADS# is sampled active.
- Recognize a cycle as ADS# asserted and BOFF# negated in the previous clock.
- Assert AHOLD one clock before asserting BOFF#.

6.4.5 Bus Hold

The embedded Pentium processor provides a bus hold, hold acknowledge protocol using the HOLD and HLDA pins. HOLD is used to indicate to the processor that another bus master wants control of the bus. When the processor completes all outstanding bus cycles, it releases the bus by floating its external bus, and drives HLDA active. An example HOLD/HLDA transaction is shown in Figure 6-16.

Figure 6-16. HOLD/HLDA Cycles



The processor recognizes HOLD while RESET is asserted, when BOFF# is asserted, and during BIST (built in self test). HOLD is not recognized when LOCK# is asserted. Once HOLD is recognized, HLDA is asserted two clocks after the later of the last BRDY# or HOLD assertion. Because of this, it is possible that a cycle may begin after HOLD is asserted, but before HLDA is driven. The maximum number of cycles that are driven after HOLD is asserted is one. BOFF# may be used if it is necessary to force the processor to float its bus in the next clock. Figure 6-16 shows the latest HOLD may be asserted relative to ADS# to guarantee that HLDA is asserted before another cycle is begun.

The operation of HLDA is not affected by the assertion of BOFF#. If HOLD is asserted while BOFF# is asserted, HLDA is asserted two clocks later. If HOLD goes inactive while BOFF# is asserted, HLDA is deasserted two clocks later.

Note that HOLD may be acknowledged between two bus cycles in a misaligned access.

All outputs are floated when HLDA is asserted except: APCHK#, BREQ, FERR#, HIT#, HITM#, HLDA, IERR#, PCHK#, PRDY, BP3–BP2, PM1/BP1, PM0/BP0, SMIACT# and TDO.

6.4.6 Interrupt Acknowledge

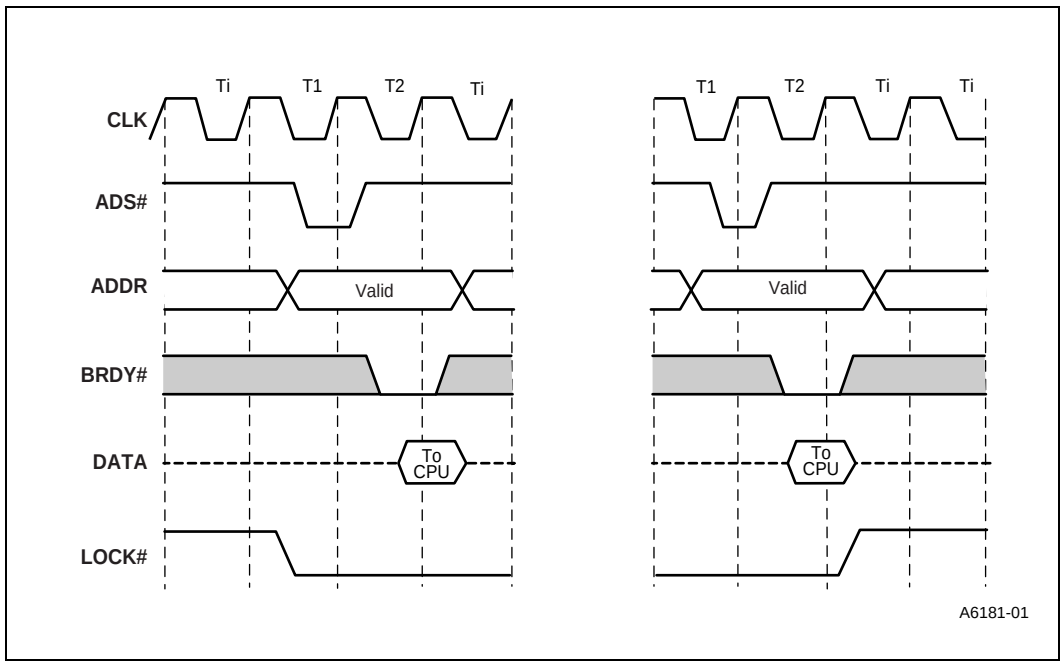
The processor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin (if interrupts are enabled). Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

An example interrupt acknowledge transaction is shown in Figure 6-17. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored, however the specified data setup and hold times must be met. The interrupt vector is returned during the second cycle on the lower 8 bits of the data bus. The processor has 256 possible interrupt vectors.

The state of address bit 2 (as decoded from the byte enables) distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4: A31–A3 = 0, BE4# = 0, BE7#–BE5# = 1, and BE3#–BE0# = 1. The address driven during the second interrupt acknowledge cycle is 0: A31–A3 = 0, BE0# = 0 and BE7#–BE1# = 1H.

Interrupt acknowledge cycles are terminated when the external system returns BRDY#. Wait states can be added by withholding BRDY#. The processor automatically generates at least one idle clock between the first and second cycles; however the external system is responsible for interrupt controller (8259A) recovery.

Figure 6-17. Interrupt Acknowledge Cycles



6.4.7 Flush Operations

The FLUSH# input is implemented in the processor as an asynchronous interrupt, similar to NMI. Therefore, unlike the Intel486™ microprocessor, FLUSH# is recognized on instruction boundaries only. FLUSH# is latched internally. Once setup, hold and pulse width times have been met, FLUSH# may be deasserted, even if a bus cycle is in progress.

To execute a flush operation, the processor first writes back all modified lines to external memory. The lines in the internal caches are invalidated as they are written back. After the write-back and invalidation operations are complete, a special cycle, flush acknowledge, is generated by the processor to inform the external system.

6.4.8 Special Bus Cycles

The processor provides six special bus cycles to indicate that certain instructions have been executed, or certain conditions have occurred internally. The special bus cycles in Table 6-13 are defined when the bus cycle definition pins are in the following state: M/IO# = 0, D/C# = 0 and W/R# = 1. During most special cycles the data bus is undefined and the address lines A31–A3 are driven to “0.” The external hardware must acknowledge all special bus cycles by returning BRDY#.

Table 6-13. Special Bus Cycles Encoding

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	Special Bus Cycle
1	1	1	1	1	1	1	0	Shutdown
1	1	1	1	1	1	0	1	Flush (INVD,WBINVD instr)
1	1	1	1	1	0	1	1	Halt/Stop Grant†
1	1	1	1	0	1	1	1	Writeback (WBINVD instruction)
1	1	1	0	1	1	1	1	Flush Acknowledge (FLUSH# assertion)
1	1	0	1	1	1	1	1	Branch Trace Message

† The definition of the Stop Grant bus cycle is the same as the HALT cycle definition, with the exception that the address bus is driven with the value 0000 0010H during the Stop Grant bus cycle.

Shutdown can be generated due to the following reasons:

- If any other exception occurs while the processor is attempting to invoke the double-fault handler.
- An internal parity error is detected.

Prior to going into shutdown, the processor does not writeback the M-state lines. Upon entering shutdown, the state of the processor is unpredictable and may or may not be recoverable. RESET or INIT should be asserted to return the system to a known state. Although some system operations (i.e., FLUSH# and R/S#) are generally recognized during shutdown, these operations may not complete successfully in some cases once shutdown is entered. During shutdown, the internal caches remain in the same state unless an inquire cycle is run or the cache is flushed.

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

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

The Flush Special Cycle is driven after the INVD (invalidate cache) or WBINVD (writeback invalidate cache) instructions are executed. The Flush Special Cycle is driven to indicate to the external system that the internal caches were invalidated and that external caches should also be invalidated.

Note: INVD should be used with care. This instruction does not write back modified cache lines.

The Halt Special Cycle is driven when a Halt instruction is executed. Externally, halt differs from shutdown in only two ways:

- In the resulting byte enables that are asserted.
- The processor exits the Halt state if INTR is asserted and maskable interrupts are enabled in addition to the assertion of NMI, INIT or RESET.

A special Stop Grant bus cycle is driven after the processor recognizes the STPCLK# interrupt. The definition of the Stop Grant bus cycle is the same as the HALT cycle definition, with the exception that the address bus is driven with the value 0000 0010H during the Stop Grant bus cycle.

The Writeback Special Cycle is driven after the WBINVD instruction is executed and it indicates that modified lines in the processor data cache were written back to memory or a second level cache. The Writeback Special Cycle also indicates that modified lines in external caches should be written back. After the WBINVD instruction is executed, the Writeback Special cycle is generated, followed by the Flush Special Cycle. Note that INTR is not recognized while the WBINVD instruction is being executed.

When the FLUSH# pin is asserted to the processor, all modified lines in the data cache are written back and all lines in the code and data caches are invalidated. The Flush Acknowledge Special Cycle is driven after the writeback and invalidations are complete. The Flush Acknowledge Special Cycle is driven only in response to the FLUSH# pin being activated. Note that the Flush Acknowledge Special Cycle indicates that all modified lines were written back and all cache lines were invalidated while the Flush special cycle only indicates that all cache lines were invalidated.

The Branch Trace Message Special Cycle is part of the processor’s execution tracing protocol. The Branch Trace Message Special Cycle is the only special cycle that does not drive 0’s on the address bus, however like the other special cycles, the data bus is undefined. When the branch trace message is driven, bits 31–3 of the branch target linear address are driven on A31–A3.

6.4.9 Bus Error Support

The processor provides basic support for bus error handling through data and address parity check. Even data parity is generated by the processor for every enabled byte in write cycles and is checked for all valid bytes in read cycles. The PCHK# output signals if a data parity error is encountered for reads.

Even address parity is generated for A31–A5 during write and read cycles, and checked during inquire cycles. The APCHK# output signals if an address parity error is encountered during inquire cycles.

External hardware is free to take whatever actions are appropriate after a parity error. For example, external hardware may signal an interrupt if PCHK# or APCHK# is asserted. See Chapter 10, “Error Detection” for details.

6.4.10 Pipelined Cycles

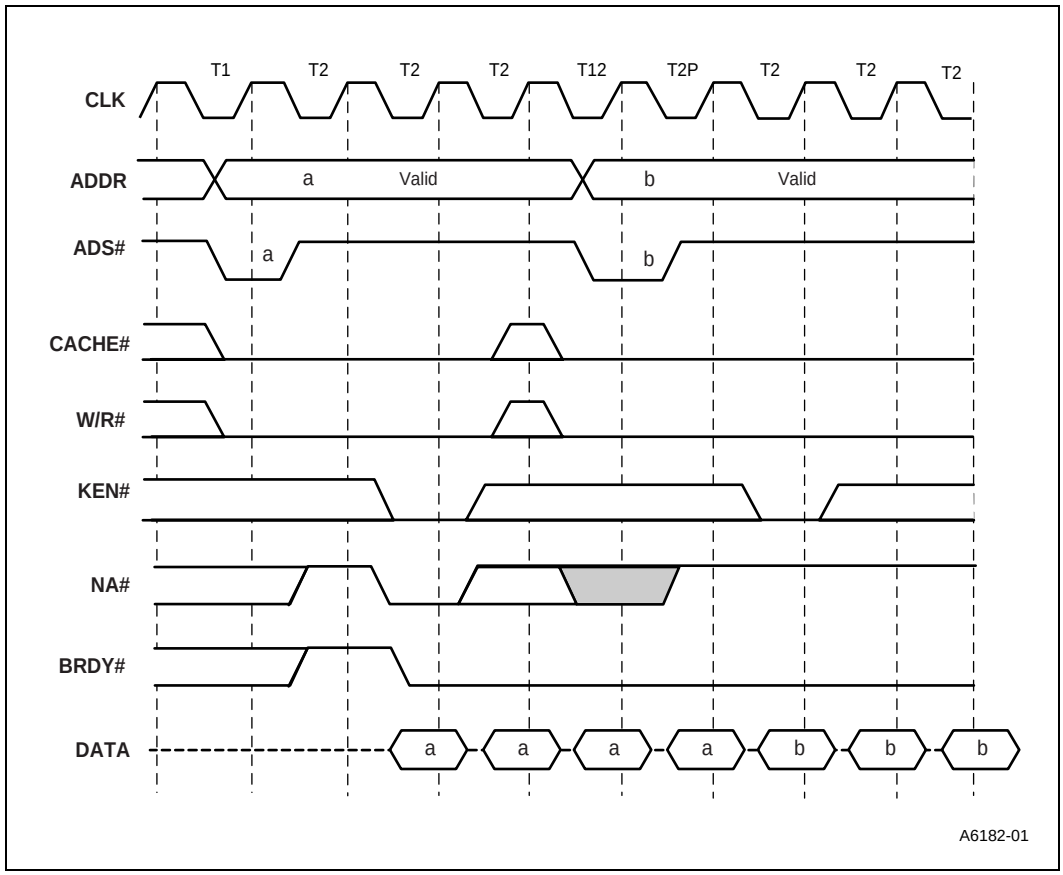
The NA# input indicates to the processor that it may drive another cycle before the current one is completed. Cacheability (KEN#) and cache policy (WB/WT#) indicators for the current cycle are sampled in the same clock NA# is sampled active (or the first BRDY# for that cycle, whichever comes first). Note that the WB/WT# and KEN# inputs are sampled with the first of BRDY# or NA# even if NA# does not cause a pipelined cycle to be driven because there was no pending cycle internally or two cycles are already outstanding.

The NA# input is latched internally, so even if a cycle is not pending internally in the clock that NA# is sampled active, but becomes pending before the current cycle is complete, the pending cycle is driven to the bus even if NA# is subsequently deasserted.

LOCK# and writeback cycles are not pipelined into other cycles and other cycles are not pipelined into them (regardless of the state of NA#). Special cycles and I/O cycles may be pipelined.

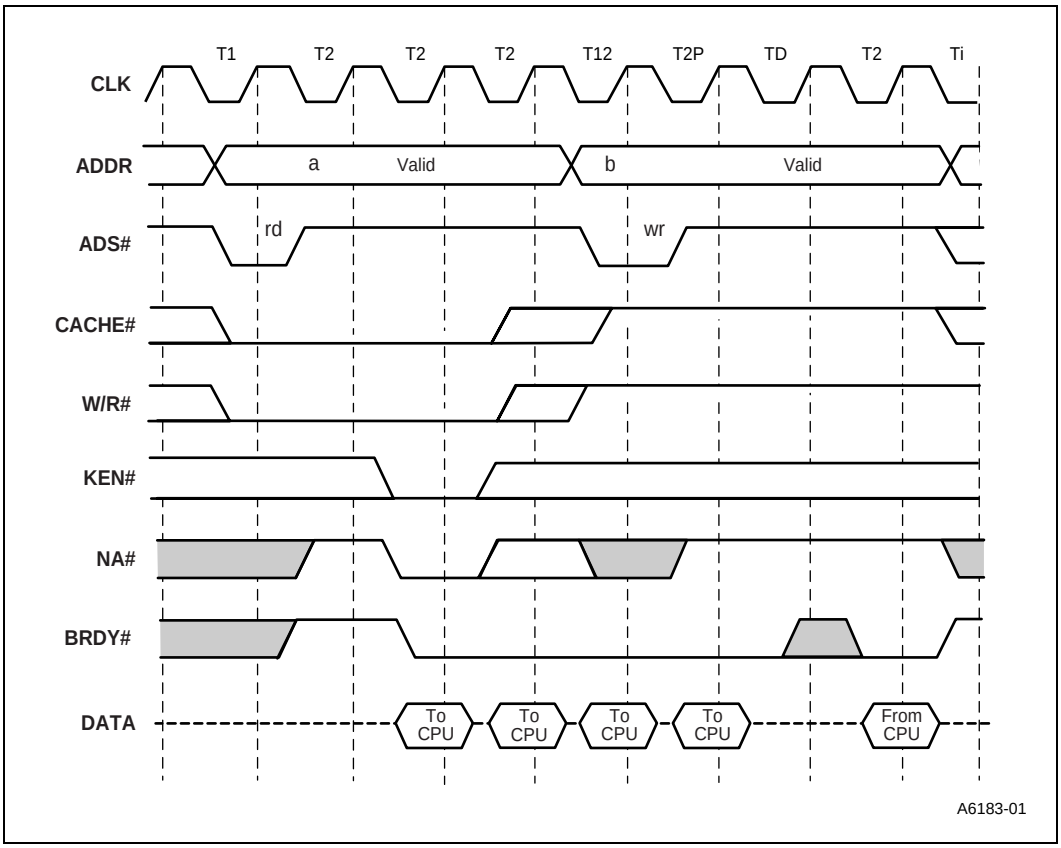
An example of burst pipelined back to back reads is shown in Figure 6-18. The assertion of NA# causes a pending cycle to be driven two clocks later. Note KEN# timing.

Figure 6-18. Two Pipelined Cache Linefills



Write cycles can be pipelined into read cycles and read cycles can be pipelined into write cycles, but one dead clock is inserted between read and write cycles to allow bus turnover (see Figure 6-6, “Processor Bus Control State Machine” on page 6-9). Pipelined back-to-back read/write cycles are shown in Figure 6-19.

Figure 6-19. Pipelined Back-to-Back Read/Write Cycles



6.4.10.1 KEN# and WB/WT# Sampling for Pipelined Cycles

KEN# and WB/WT# are sampled with NA# or BRDY# for that cycle, whichever comes first. Figure 6-20 and Figure 6-21 clarify this specification.

Figure 6-20 shows that even though two cycles have been driven, the NA# for the second cycle still causes KEN# and WB/WT# to be sampled for the second cycle. A third ADS# is not driven until all the BRDY#s for cycle 1 are returned to the processor.

Figure 6-20. KEN# and WB/WT# Sampling with NA#

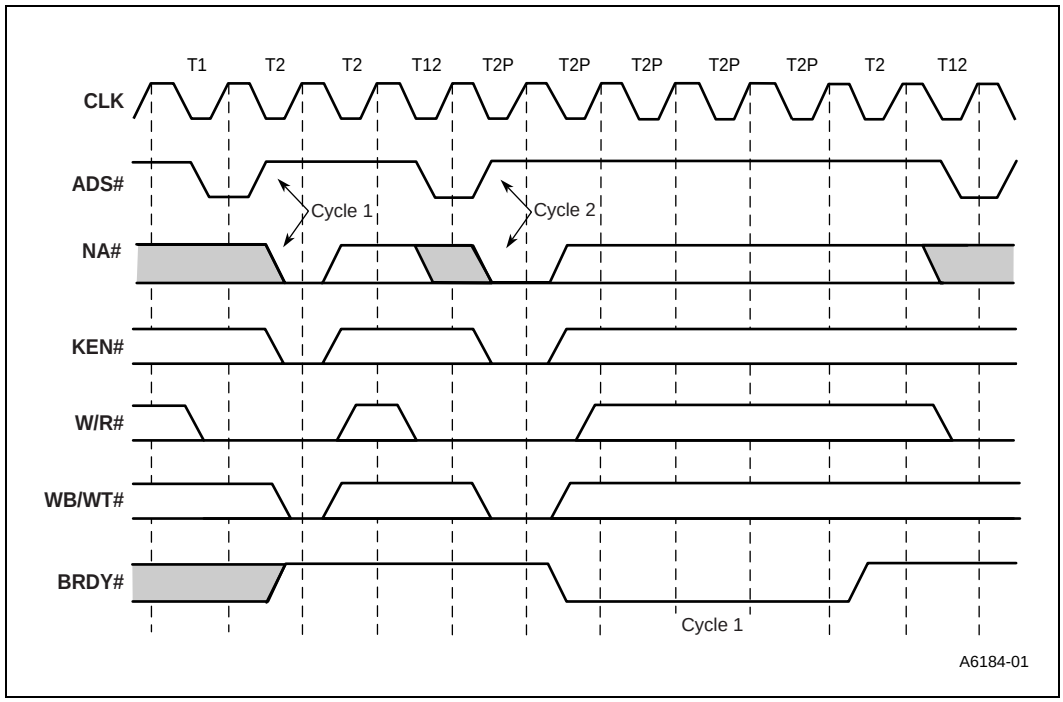
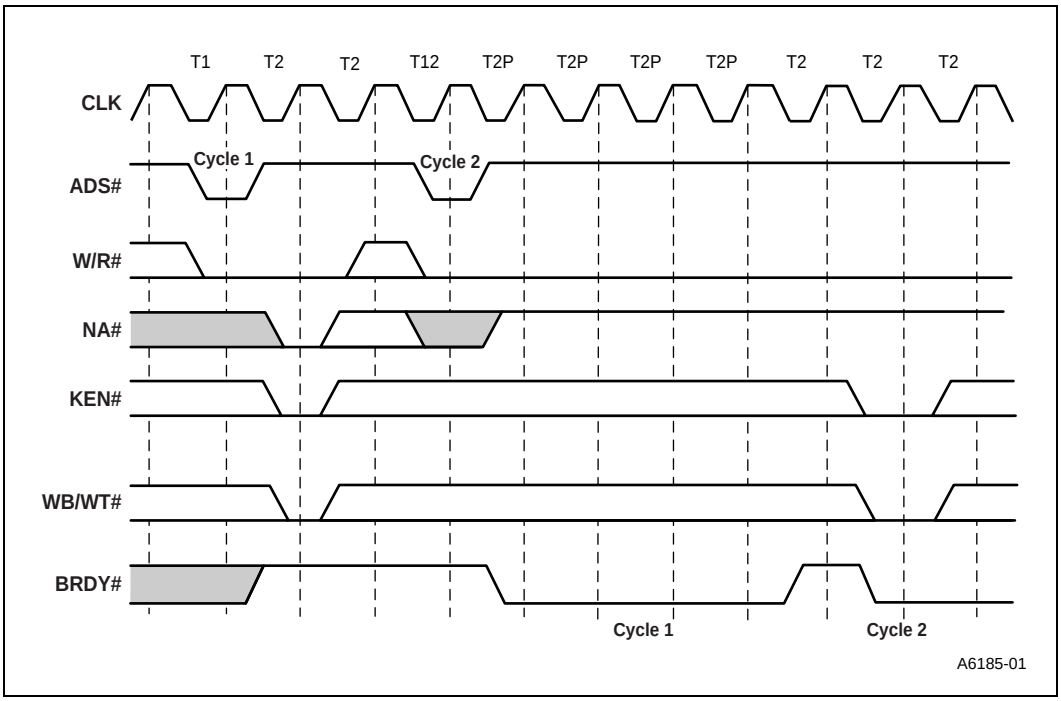


Figure 6-21 shows that two cycles are outstanding on the processor bus. The assertion of NA# caused the sampling of KEN# and WB/WT# for the first cycle. The assertion of the four BRDY#s for the first cycle DO NOT cause the KEN# and WB/WT# for the second cycle to be sampled. In this example, KEN# and WB/WT# for the second cycle are sampled with the first BRDY# for the second cycle.

Figure 6-21. KEN# and WB/WT# Sampling with BRDY#

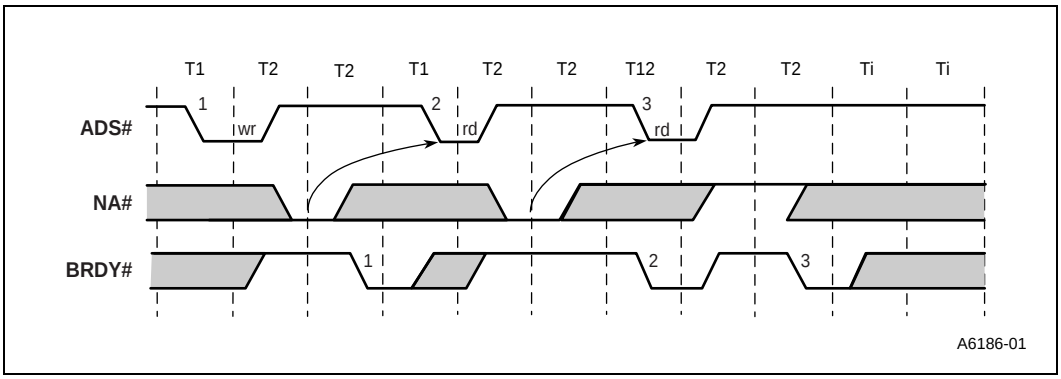


6.4.11 Dead Clock Timing Diagrams

The timing diagrams in Figure 6-22 and Figure 6-23 show bus cycles with and without a dead clock.

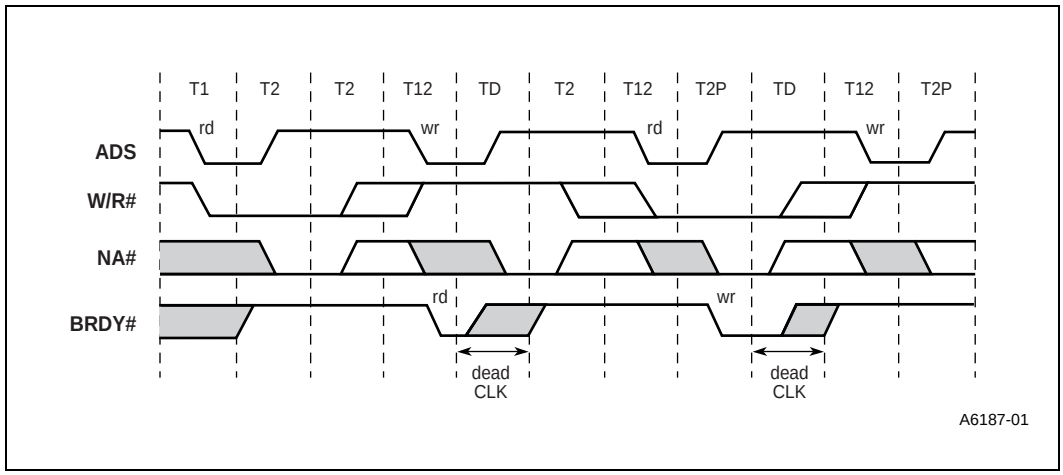
In Figure 6-22, cycles 1 and 2 can be either read or write cycles and no dead clock would be needed because only one cycle is outstanding when those cycles are driven. To prevent a dead clock from being necessary after cycle 3 is driven, it must be of the “same type” as cycle 2. That is if cycle 2 is a read cycle, cycle 3 must also be a read cycle in order to prevent a dead clock. If cycle 2 is a write cycle, cycle 3 must also be a write cycle to prevent a dead clock.

Figure 6-22. Bus Cycles without Dead Clock



Note: Although the processor ignores BRDY# during this dead clock when configured in uni-processor mode, BRDY# may be falsely recognized in an inter-processor pipelined cycle. As such, dual processing system designs must not drive BRDY# low during this dead clock.

Figure 6-23. Bus Cycles with TD Dead Clock



6.5 Cache Consistency Cycles (Inquire Cycles)

The purpose of an inquire cycle is to check whether a particular address is cached in a processor internal cache and optionally invalidate it. After an inquire cycle is complete, the system has information on whether or not a particular address location is cached and what state it is in.

An inquire cycle is typically performed by first asserting AHOLD to force the processor to float its address bus, waiting two clocks, and then driving the inquire address and INV and asserting EADS#. Inquire cycles may also be executed while the processor is forced off the bus due to HLDA, or BOFF#. Because the entire cache line is affected by an inquire cycle, only A31–A5 need to be driven with the valid inquire address. Although the value of A4–A3 is ignored, these inputs should be driven to a valid logic level during inquire cycles for circuit reasons. The INV pin is driven along with the inquire address to indicate whether the line should be invalidated (INV high) or marked as shared (INV low) in the event of an inquire hit.

After the processor determines if the inquire cycle hit a line in either internal cache, it drives the HIT# pin. HIT# is asserted (low) two clocks after EADS# is sampled asserted[1] if the inquire cycle hit a line in the code or data cache. HIT# is deasserted (high) two clocks after EADS# is sampled asserted if the inquire cycle missed in both internal caches. The HIT# output changes its value only as a result of an inquire cycle. It retains its value between inquire cycles. In addition, the HITM# pin is asserted two clocks after EADS# if the inquire cycle hit a modified line in the data cache. HITM# is asserted to indicate to the external system that the processor contains the most current copy of the data and any device needing to read that data should wait for the processor to write it back. The HITM# output remains asserted until two clocks after the last BRDY# of the writeback cycle is asserted.

The external system must inhibit inquire cycles during BIST (initiated by INIT being sampled high on the falling edge of RESET), and during the Boundary Scan Instruction RUNBIST. When the model specific registers (test registers) are used to read or write lines directly to or from the cache

1. Since the EADS# input is ignored by the processor in certain clocks, the two clocks reference is from the clock in which EADS# is asserted and actually sampled by the processor at the end of this clock (i.e., rising edge of next clock) as shown in Figure 6-25.

it is important that external snoops (inquire cycles) are inhibited to guarantee predictable results when testing. This can be accomplished by inhibiting the snoops externally or by putting the processor in SRAM mode (CR0.CD=CR0.NW=1).

The EADS# input is ignored during external snoop writeback cycles (HITM# asserted), or during the clock after ADS# or EADS# is active. EADS# is also ignored when the processor is in SRAM mode, or when the processor is driving the address bus.

Note that the processor may drive the address bus in the clock after AHOLD is deasserted. It is the responsibility of the system designer to ensure that address bus contention does not occur. This can be accomplished by not deasserting AHOLD to the processor until all other bus masters have stopped driving the address bus.

Figure 6-24 shows an inquire cycle that misses both internal caches. Note that both the HIT# and HITM# signals are deasserted two clocks after EADS# is sampled asserted.

Figure 6-25 shows an inquire cycle that invalidates a non-modified line. Note that INV is asserted (high) in the clock that EADS# is returned. Note that two clocks after EADS# is sampled asserted, HIT# is asserted and HITM# is deasserted.

Figure 6-24 and Figure 6-25 both show that the AP pin is sampled/driven along with the address bus, and that the APCHK# pin is driven with the address parity status two clocks after EADS# is sampled asserted.

An inquire cycle that hits a M-state line is shown in Figure 6-26. Both the HIT# and HITM# outputs are asserted two clocks after EADS# is sampled asserted. ADS# for the writeback cycle occurs no earlier than two clocks after the assertion of HITM#.

Figure 6-24. Inquire Cycle that Misses the Processor Cache

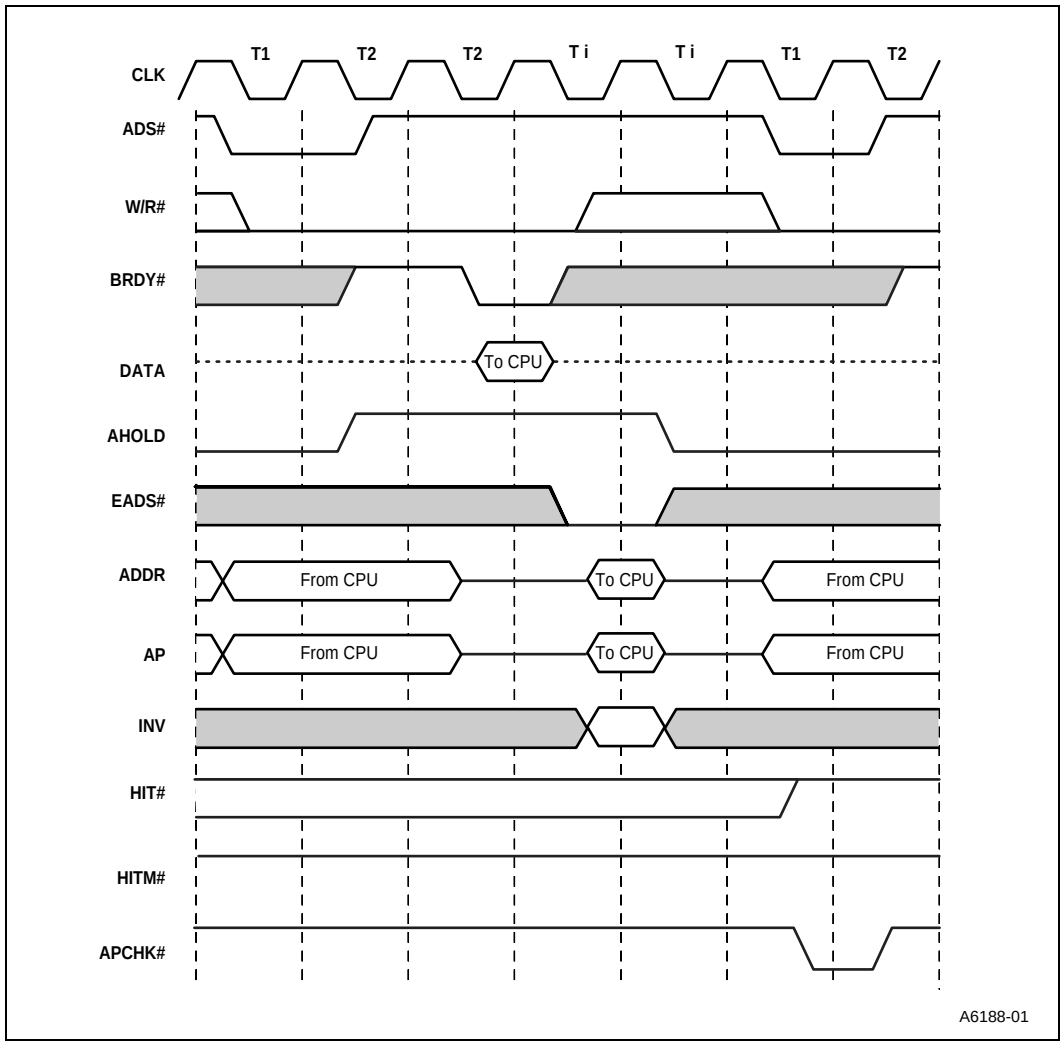
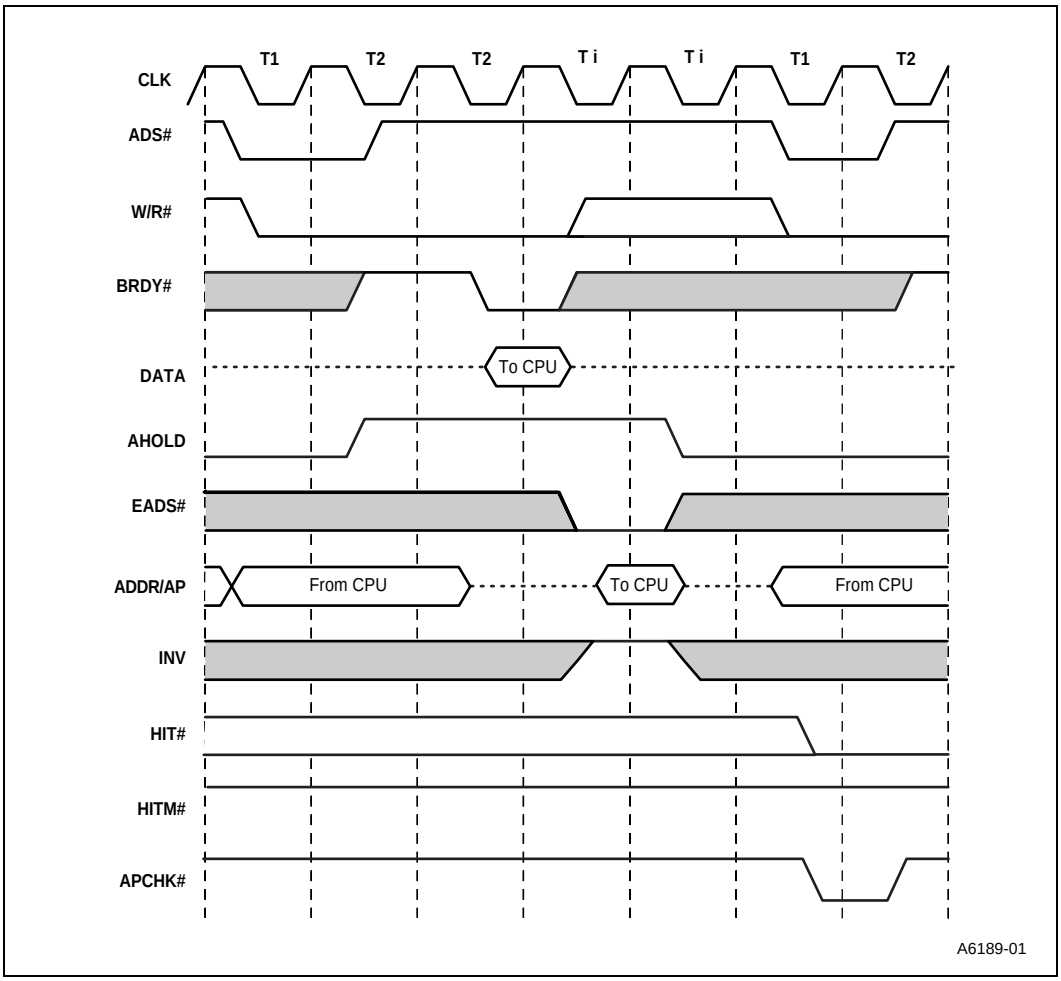


Figure 6-25. Inquire Cycle that Invalidates a Non-M-State Line



HITM# is asserted only if an inquire cycle (external snoop) hits a modified line in the processor data cache. HITM# is not asserted for internal snoop writeback cycles or cache replacement writeback cycles. HITM# informs the external system that the inquire cycle hit a modified line in the data cache and that line is written back. Any ADS# driven by the processor while HITM# is asserted will be the ADS# of the writeback cycle. The HITM# signal stays active until the last BRDY# is returned for the corresponding inquire cycle. Writeback cycles start at burst address 0.

Note that ADS# is asserted despite the AHOLD signal being active. This ADS# initiates a writeback cycle corresponding to the inquire hit. Such a cycle can be initiated while address lines are floating to support multiple inquires within a single AHOLD session. This functionality can be used during secondary cache replacement processing if its line is larger than the processor cache line (32 bytes). Although the cycle specification is driven properly by the processor, address pins are not driven because AHOLD forces the processor off the address bus. If AHOLD is cleared before the processor drives out the inquire writeback cycle, the processor drives the correct address for inquire writeback in the next clock. The ADS# to initiate a writeback cycle as a result of an inquire hit is the only time ADS# is asserted while AHOLD is also asserted.

Note that in the event of an address parity error during inquire cycles, the snoop cycle is not inhibited. If the inquire hits a modified line in this situation and an active AHOLD prevents the processor from driving the address bus, the processor potentially writes back a line at an address other than the one intended. If the processor is not driving the address bus during the writeback cycle, it is possible that memory will corrupted.

If BOFF# or HLDA were asserted to perform the inquire cycle, the writeback cycle would wait until BOFF# or HLDA was deasserted.

State machines should not depend on a writeback cycle to follow an assertion of HITM#. HITM# may be negated without a corresponding writeback cycle being run. This may occur as a result of the internal caches being invalidated due to the INVD instruction or by testability accesses. Note that inquire cycles occurring during testability accesses generate unpredictable results. In addition, a second writeback cycle is not generated for an inquire cycle that hits a line already being written back (see Figure 6-28). This can happen if an inquire cycle hits a line in one of the processor writeback buffers.

6.5.1 Restrictions on Deassertion of AHOLD

To prevent the address and data buses from switching simultaneously, the following restrictions are placed on the negation of AHOLD: (i) AHOLD must not be negated in the same clock as the assertion of BRDY# during a write cycle; (ii) AHOLD must not be negated in the dead clock between write cycles pipelined into read cycles; and (iii) AHOLD must not be negated in the same clock as the assertion of ADS# while HITM# is asserted. Note that there are two clocks between EADS# being sampled asserted and HITM# being asserted, and a further minimum of two clocks between an assertion of HITM# and ADS#.

These restrictions on the deassertion of AHOLD are the only considerations the system designer needs to make to prevent the simultaneous switching of the address and data buses. All other considerations are handled internally.

Figure 6-26 can be used to illustrate restrictions (i) and (iii). AHOLD may be deasserted in Clock 2, 3, or 4, but not in Clock 5, 6, 7, 8 or 9.

Figure 6-27 and Figure 6-28 depict restrictions (i) and (ii) respectively. Note that there are no restrictions on the assertion of AHOLD.

Figure 6-26. Inquire Cycle that Invalidates M-State Line

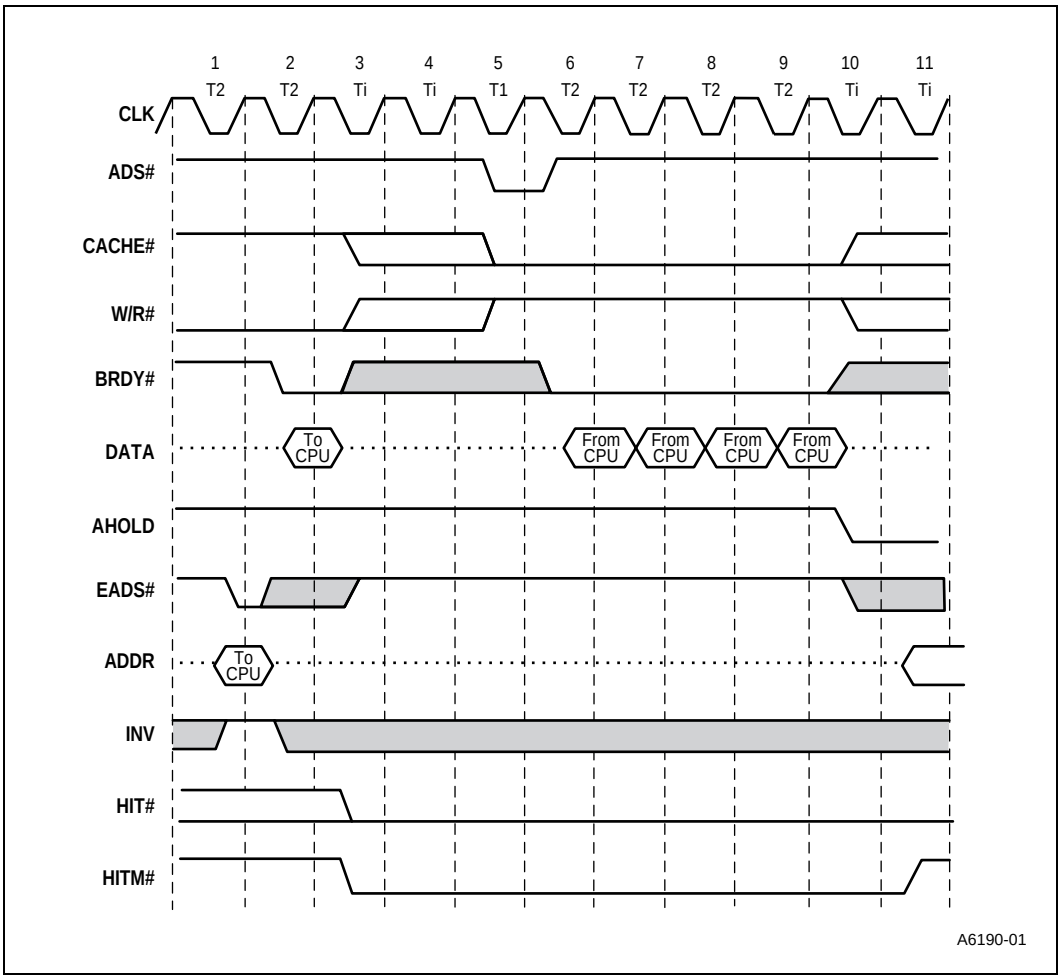


Figure 6-27 shows a writeback (due to a previous snoop that is not shown). ADS# for the writeback is asserted even though AHOLD is asserted. Note that AHOLD can be deasserted in Clock 2, 4, 7, or 9. AHOLD cannot be deasserted in Clock 1, 3, 5, 6, or 8.

Figure 6-27. AHOLD Restriction during Write Cycles

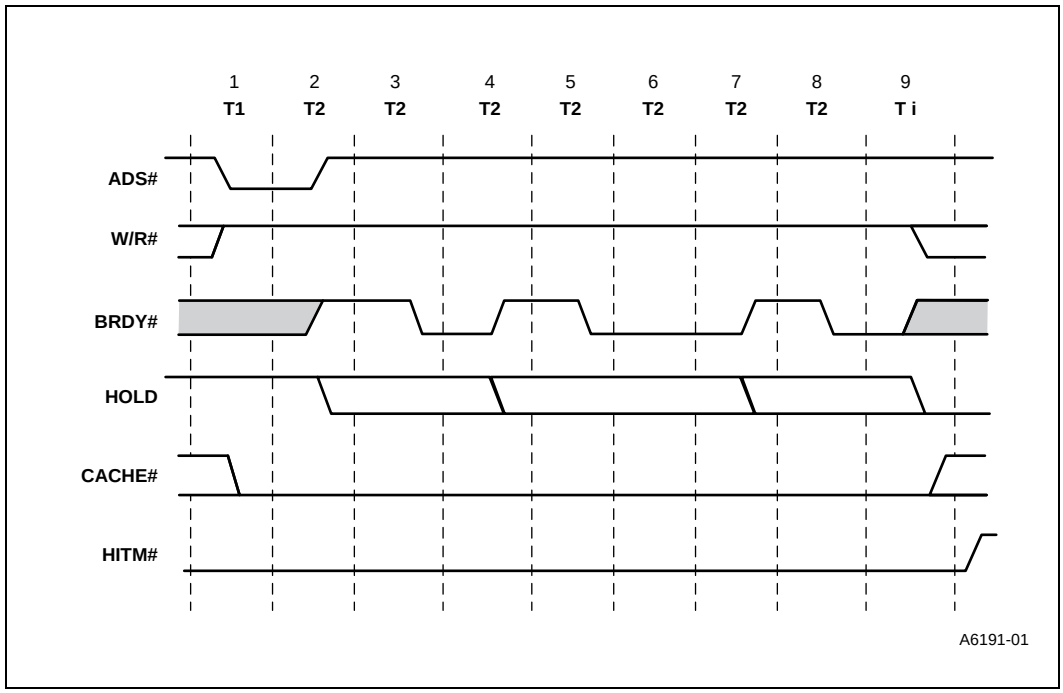
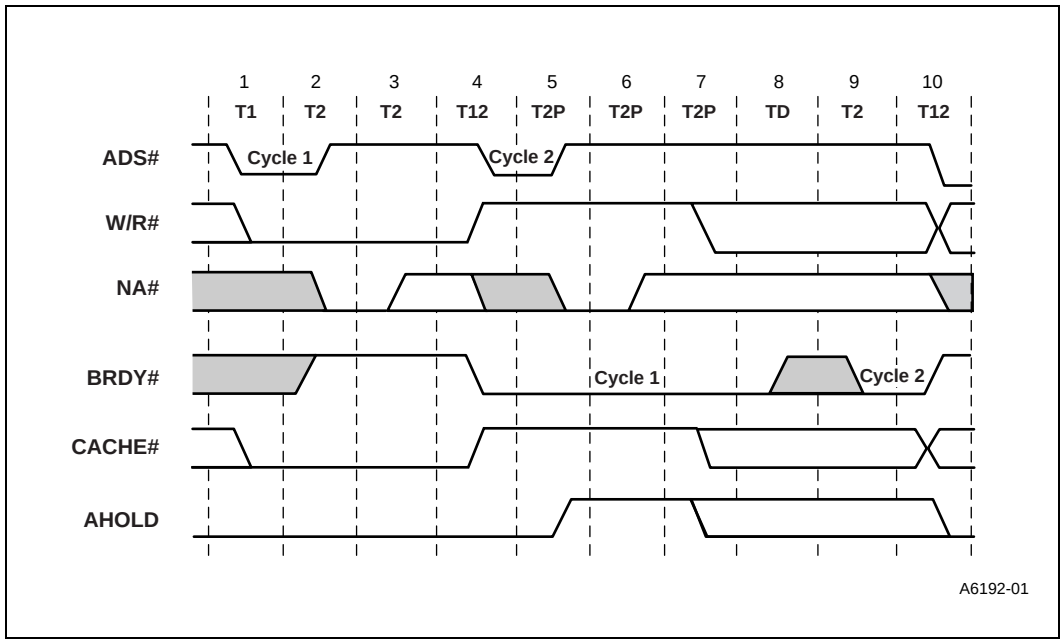


Figure 6-28 shows a write cycle being pipelined into a read cycle. Note that if AHOLD is asserted in Clock 5, it can be deasserted in Clock 7 before the TD, or in Clock 10 after the TD, but it cannot be deasserted in Clock 8 (the TD clock). AHOLD cannot be deasserted in Clock 9 because BRDY# for the write cycle is being returned.

Figure 6-28. AHOLD Restriction During TD



6.5.2 Rate of Inquire Cycles

The processor can accept inquire cycles at a maximum rate of one every other clock. However, if an inquire cycle hits an M-state line of the processor, subsequent inquire cycles will be ignored until the line is written back and HITM# is deasserted. EADS# is also ignored the clock after ADS# is asserted.

6.5.3 Internal Snooping

"Internal snoop" is the term used to describe the snooping of the internal code or data caches that is not initiated by the assertion of EADS# by the external system. Internal snooping occurs in the three cases described below. Note that neither HIT# nor HITM# are asserted as a result of an internal snoop.

1. An internal snoop occurs if an access is made to the code cache, and that access is a miss. In this case, if the accessed line is in the S or E state in the data cache, the line is invalidated. If the accessed line is in the M state in the data cache, the line is written back then invalidated.
2. An internal snoop occurs if an access is made to the data cache, and that access is a miss or a writethrough. In this case, if the accessed line is valid in the code cache, the line is invalidated.
3. An internal snoop occurs if there is a write to the accessed and/or dirty bits in the page table/directory entries. In this case, if the accessed line is valid in either the code or data cache, the line is invalidated. If the accessed line is in the M state in the data cache, the line is written back then invalidated.

6.5.4 Snooping Responsibility

In systems with external second level caches allowing concurrent activity of the memory bus and processor bus, it is desirable to run invalidate cycles concurrently with other processor bus activity. Writes on the memory bus can cause invalidations in the secondary cache at the same time that the processor fetches data from the secondary cache. Such cases can occur at any time relative to each other, and therefore the order in which the invalidation is requested, and data is returned to the processor becomes important.

The processor always snoops the instruction and data caches when it accepts an inquire cycle. If a snoop comes in during a linefill, the processor also snoops the line currently being filled. If more than one cacheable cycle is outstanding (through pipelining), the addresses of both outstanding cycles are snooped.

For example, during linefills, the processor starts snooping the address(es) associated with the line(s) being filled after KEN# has been sampled active for the line(s). Each line is snooped until it is put in the cache. If a snoop hits a line being currently filled, the processor asserts HIT# and the line ends up in the cache in the S or I state, depending on the value of the INV pin sampled during the inquire cycle. However, the processor uses the data returned for that line as a memory operand for the instruction that caused the data cache miss/line fill or execute an instruction contained in a code cache miss/line fill.

Figure 6-29 and Figure 6-30 illustrate the snoop responsibility pickup. Figure 6-29 shows a non-pipelined cycle, while Figure 6-30 illustrates a pipelined cycle. The figures show the earliest EADS# assertion that causes snooping of the line being cached relative to the first BRDY# or NA#.

Figure 6-29. Snoop Responsibility Pickup — Non-Pipelined Cycles

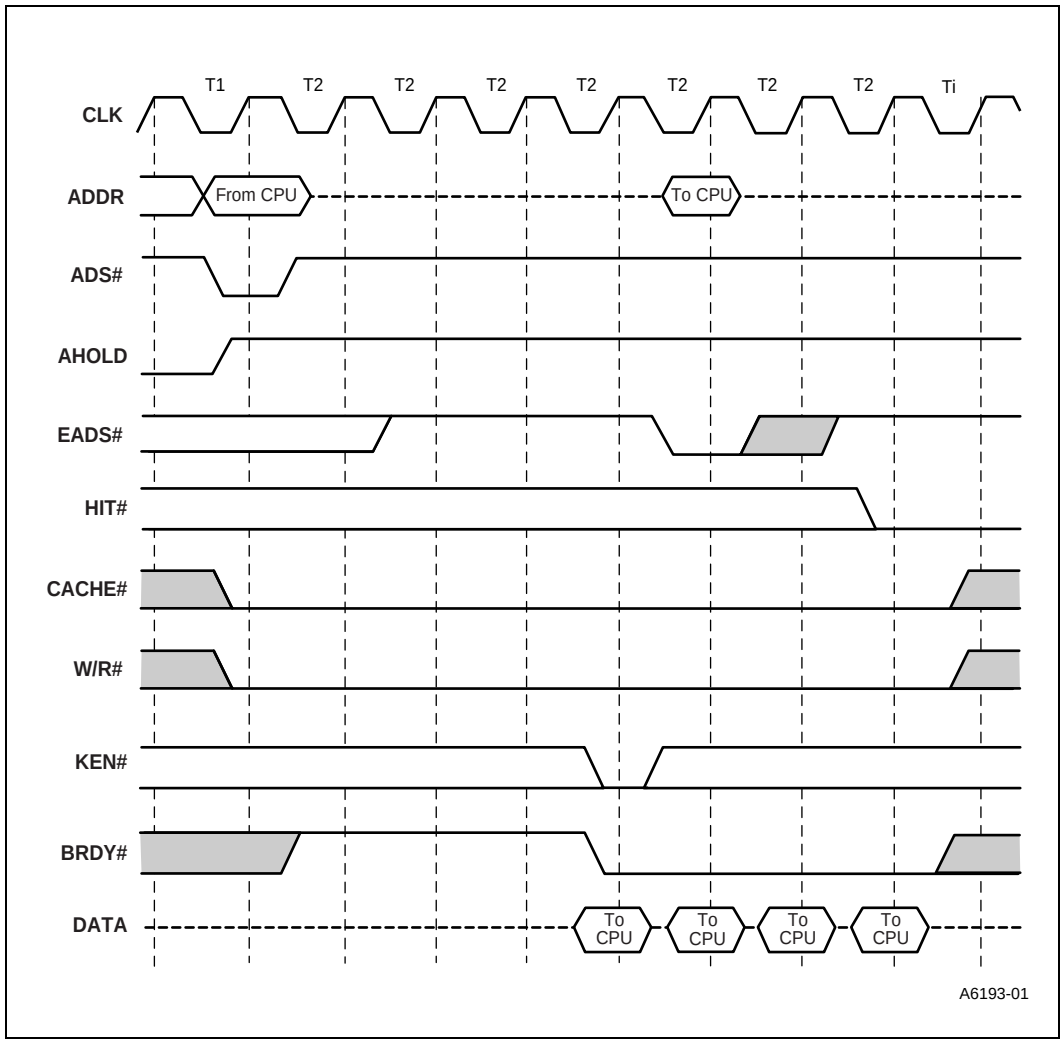
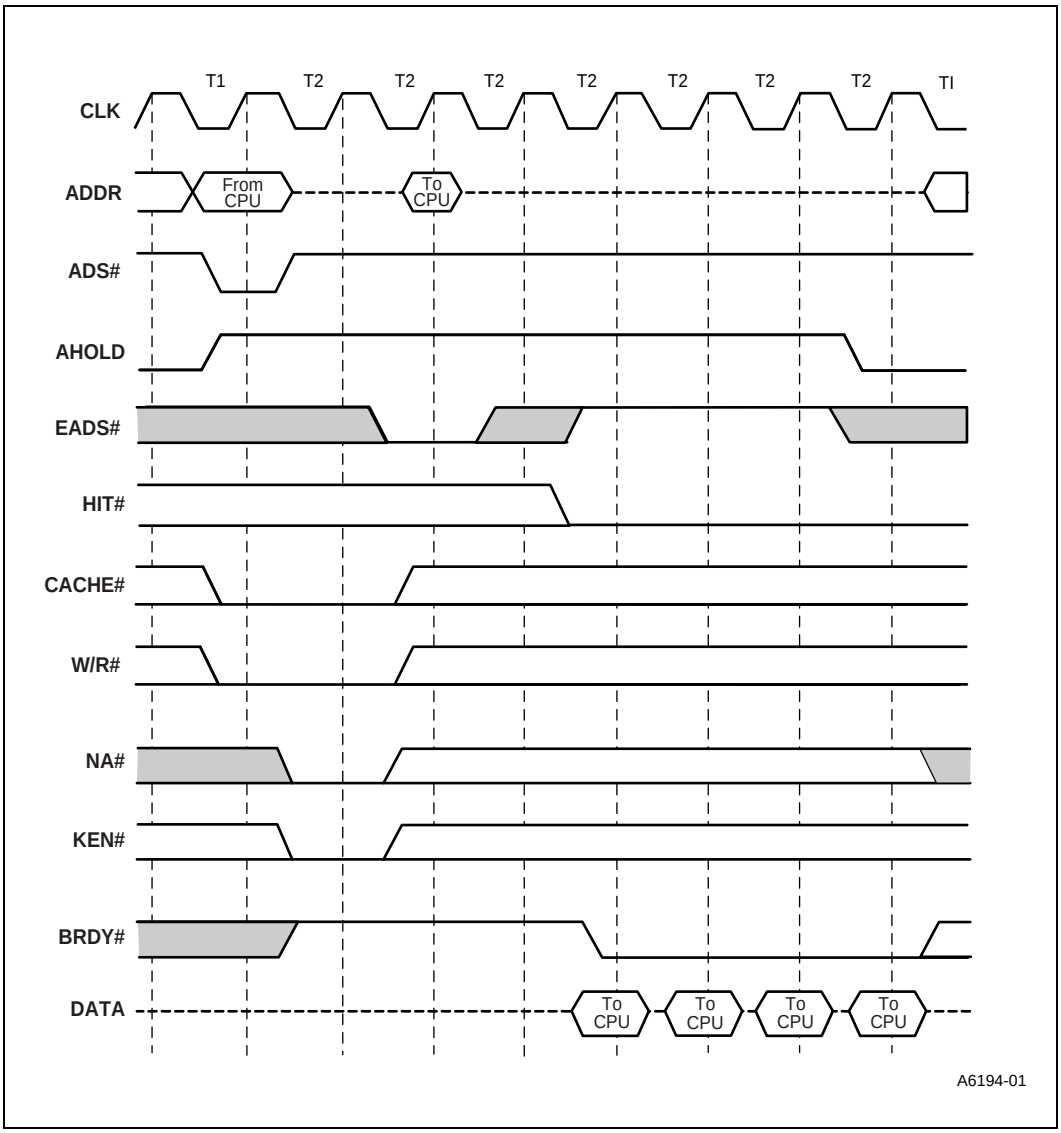


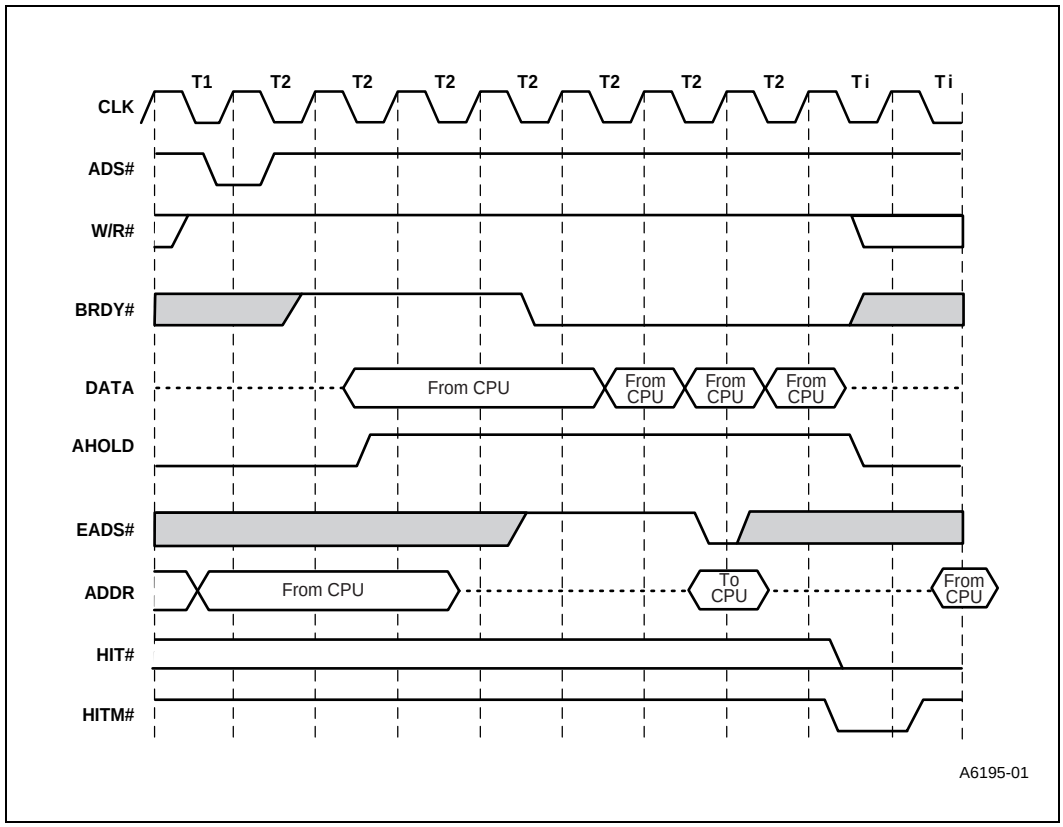
Figure 6-30. Snoop Responsibility Pickup — Pipelined Cycle



The processor also snoops M state lines in the writeback buffers until the writeback of the M state lines are complete. If a snoop hits an M state line in a writeback buffer, both HIT# and HITM# are asserted. Figure 6-31 illustrates snooping (snoop responsibility drop) of an M state line that is being written back because it has been replaced with a “new” line in the data cache. It shows the latest EADS# assertion, relative to the last BRDY# of the writeback cycle that results in a snoop hit to the line being written back. HITM# stays asserted until the writeback is complete. Note that an additional ADS# is not asserted during the writeback cycle.

The HIT# signal is a super set of the HITM# signal; it is always asserted with HITM#.

Figure 6-31. Latest Snooping of Writeback Buffer



6.6 Summary of Dual Processing Bus Cycles

The following is a list of bus cycles or bus cycle sequences which would not occur in embedded Pentium processor uni-processor systems, but may be seen in Dual processor systems.

- Locked Cycle Sequences
- Cycle Pipelining
- Cycle Ordering Due to BOFF#
- Cache Line State
- Back-to-Back Cycles
- Address Parity Checking
- Synchronous FLUSH# and RESET
- PCHK# Assertion
- Flush Cycles
- Floating-Point Error Handling

6.6.1 Locked Cycle Sequences

1. Locked read to address X
2. Locked write back to address X
3. Locked read to address X
4. Locked write to address X

May occur due to the inter-processor cache consistency mechanism. Refer to Chapter 4, “Microprocessor Initialization and Configuration.”

Implications

Processor bus hardware needs to handle this locked sequence. The only other time the system sees a locked write back is when an external snoop hits a modified line while a locked cycle is in progress (this can occur in a uni-processor or a dual-processor system).

6.6.2 Cycle Pipelining

Inter-processor (Primary/Dual processor) back-to-back write cycles are not pipelined even if NA# has been asserted. The purpose of this rule is to prevent data bus contention during bus arbitration from one processor to the other. In dual processor mode, the Primary processor may pipeline I/O cycles into I/O cycles from the Dual processor (and vice versa) for any I/O instruction combination (i.e., except I/O writes into writes).

Implications

System hardware designers should be aware of these bus changes.

6.6.3 Cycle Ordering Due to BOFF#

Cycle ordering following an assertion of BOFF# may be different between uni-processor and dual processor modes. This occurs when there are pipelined cycles from both processors, a BOFF# stalls both cycles, and an external snoop hits a modified line in the LRMs cache.

Implications

System hardware designers should be aware of these bus changes.

6.6.4 Cache Line State

In embedded Pentium processor family uni-processor systems, if a line is put into the E state by the system hardware using the WB/WT# signal during the line fill, then all subsequent writes to that line are handled internally via the on-chip cache. In dual-processor systems, under certain circumstances, even if the system puts a line into the E state using WB/WT#, the dual-processor protocol may force the line to be stored in the S state. Private snooping in dual processor systems can also cause a line to be placed into the S or I state.

Implications

There are no system implications. The system may be required to handle writes to a line which would not otherwise have been seen.

Note: In a dual processing system where NW=1 and CD=1 are set, (i.e., SRAM mode), an inquire cycle invalidates a cache line with INV on a HIT#.

6.6.5 Back-to-Back Cycles

Due to the dual-processor cache consistency protocol, the Primary and Dual processors may follow a write to address X with a write back to a 32-byte area which contains X. This does not occur in uni-processor systems. Also, a read to address X may be followed by a write back to a 32-byte area which contains X.

Implications

There are no system implications.

6.6.6 Address Parity Checking

Address parity is checked during every private snoop between the Primary and Dual processors. Therefore, APCHK# may be asserted due to an address parity error during this private snoop. If an error is detected, APCHK# is asserted two clocks after ADS# for one processor clock period. The system can choose to acknowledge this parity error indication at this time or do nothing.

Implications

There are no system implications. The system designers get extra address parity checking with dual processors due to the automatic private snooping.

6.6.7 Synchronous FLUSH# and RESET

When the Dual processor is present, the FLUSH# and RESET signals must be recognized by both processors at the same time.

Implications

FLUSH# and RESET must be asserted on the same clock to both the Primary and Dual processors.

6.6.8 PCHK# Assertion

In a dual-processor configuration, there is the possibility that the PCHK# signal can be asserted either two or three CLKs following incorrect parity being detected on the data bus (depending on the bus-to-core ratio).

Implications

Chip sets must account for this difference from the embedded Pentium processor in their logic or state machines.

6.6.9 Flush Cycles

The Primary and Dual processors incorporate a mechanism to present a unified view of the cache flush operation to the system when in dual processing mode. The Dual processor performs the cache flush operation first, then grants the bus to the Primary processor. The Primary processor flushes its internal caches, and then runs the cache flush special cycle.

Implications

The system hardware *must* not assert a subsequent FLUSH# to the processors until the flush acknowledge special cycle has completed on the processor bus. The assertion of FLUSH# to the processors prior to this point would result in a corruption of the dual processing bus arbitration state machines.

6.6.10 Floating-Point Error Handling

The embedded Pentium processor, when configured as a Dual processor, ignores the IGNNE# input. The FERR# output is also undefined in the Dual processor.

Implications

None.

Electrical Differences Between Family Members 7

This section describes the electrical differences between the embedded Pentium® processor family members.

7.1 Differences Between Processors

When designing an embedded Pentium® processor with MMX™ technology system from an existing embedded Pentium processor (at 100/133/166 MHz) system, there are a number of electrical differences that require attention. Designing a single motherboard that supports various members of the embedded Pentium processor family can be accomplished easily.

Refer to the *Embedded Pentium® Processor Flexible Motherboard Design Guidelines* (order number 273206) for more information and specific implementation examples.

The following sections highlight key electrical issues pertaining to power supplies, connection specifications, and buffer models.

7.1.1 Power Supplies

One key electrical difference between family processors is the operating voltage.

- The embedded Pentium processor (at 100/133/166 MHz) requires a single voltage supply for all V_{CC} pins. This single supply powers both the core and I/O pins.
- The embedded Pentium processor with Voltage Reduction Technology, the embedded Pentium processor with MMX technology, and the low-power embedded Pentium processor with MMX technology require two separate voltage inputs, V_{CC2} and V_{CC3}. The V_{CC2} pins supply power to the core, while the V_{CC3} pins supply power to the processor I/O pins.

By connecting all of the V_{CC2} pins together and all the V_{CC3} pins together on separate power islands, embedded Pentium processor (at 100/133/166 MHz) designs can easily be converted to support the embedded Pentium processor with MMX technology. In order to maintain compatibility with embedded Pentium processor-based platforms, embedded Pentium processors with MMX technology supports the standard 3.3 V specification on its V_{CC3} pins.

Refer to each processor's datasheet for complete electrical specifications.

7.1.1.1 Power Supply Sequencing

There is no specific power sequence required for powering up or powering down the separate V_{CC2} and V_{CC3} supplies of the embedded Pentium processor with MMX technology. It is recommended that the V_{CC2} and V_{CC3} supplies be either both ON or both OFF within 1 second of each other.

7.1.2 Connection Specifications

Connection specifications for the power and ground inputs, 3.3 V inputs and outputs, and the NC/INC and unused inputs are discussed in the following sections.

7.1.2.1 Power and Ground Connections

For clean on-chip power distribution, the embedded Pentium processor has 53 V_{CC} (power) and 53 V_{SS} (ground) inputs.

Power and ground connections must be made to all V_{CC} and V_{SS} pins of the embedded Pentium processor. On the circuit board, all V_{CC} pins must be connected to a V_{CC} plane. All V_{SS} pins must be connected to a V_{SS} plane.

It is imperative that the system decoupling be sufficient to maintain ALL V_{CC} pins of the processor within their specified operating range regardless of whether a unified-plane or split-plane processor is installed.

The unified-plane embedded Pentium processor packages have a single internal V_{CC} plane. This plane may be used as the means of conduction between the V_{CC2} and V_{CC3} motherboard power planes when a unified-plane processor is installed in the system. Should such an implementation be used, it must be ensured that the maximum current flowing through the processor package does not exceed 8 Amps, including the power required by the processor. (The embedded Pentium processor (100/133/166) is a unified plane processor.)

Given the above specifications, many different implementations of power distribution are possible for embedded Pentium processor-based motherboard designs. These can be broadly categorized into two groups:

1. Unified-plane processors receive power externally to all V_{CC2} and V_{CC3} pins, while split-plane processors receive power from independent sources for V_{CC2} and V_{CC3}.
2. Unified-plane processors receive power externally to either the V_{CC2} or V_{CC3} pins, while split-plane processors receive power from independent sources for V_{CC2} and V_{CC3}.

The second implementation discussed above implies that when a unified-plane processor is installed, either the V_{CC2} or V_{CC3} voltage regulator will shut down if the voltage from the other regulator is higher than its own output setpoint. This will leave one voltage regulator powering either the V_{CC2} or V_{CC3} pins directly. The remaining functioning voltage regulator must be capable of providing the total required current independently. In the case when a split-plane processor is installed, both voltage regulators must continue to function at the proper voltage levels.

7.1.2.2 V_{CC} Measurement Specification

The values of V_{CC} should be measured at the bottom side of the processor pins using an oscilloscope with a 3 dB bandwidth of at least 20 MHz (100 MS/s digital sampling rate). There should be a short isolation ground lead attached to a processor pin on the bottom side of the board.

The measurement should be taken at the following V_{CC}/V_{SS} pairs: AN13/AM10, AN21/AM18, AN29/AM26, AC37/Z36, U37/R36, L37/H36, A25/B28, A17/B20, A7/B10, G1/K2, S1/V2, AC1/Z2. Note that on the embedded Pentium processor with MMX technology, one-half of these pins are V_{CC2} while the others are V_{CC3}; the operating ranges for the V_{CC2} and V_{CC3} pins are specified at different voltages.

The display should show continuous sampling of the V_{CC} line, at 20 mV/div, and 500 ns/div with the trigger point set to the center point of the range. Slowly move the trigger to the high and low ends of the specification, and verify that excursions beyond these limits are not observed. There are no allowances for crossing the high and low limits of the voltage specification. For more information on measurement techniques, see the *Implementation Guidelines for 3.3 V Pentium® Processors with VR/VRE Specifications* (order number 242687) and *Voltage Guidelines for Pentium® Processors with MMX™ Technology* (order number 243186) application notes.

7.1.2.3 Decoupling Recommendations

Liberal decoupling capacitance should be placed near the embedded Pentium processor. The embedded Pentium processor driving its large address and data buses at high frequencies 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 embedded Pentium 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.

For the embedded Pentium processor, the power consumption can transition from a low level of power to a much higher level (or high to low power) very rapidly. A typical example would be entering or exiting the Stop Grant State. Another example would be executing a HALT instruction, causing the embedded Pentium processor to enter the AutoHALT Power Down State, or transitioning from HALT to the Normal State. All of these examples may cause abrupt changes in the power being consumed by the embedded Pentium processor. Note that the AutoHALT Power Down 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 Ω to 100 Ω 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 embedded Pentium processor on the power plane(s) to ensure that the supply voltage stays within specified limits during changes in the supply current during operation.

Detailed decoupling recommendations are provided in the *Embedded Pentium® Processor Flexible Motherboard Design Guidelines* (order number 273206).

7.1.2.4 3.3 V Inputs and Outputs

The inputs and outputs of the embedded Pentium processor comply with the 3.3 V JEDEC standard levels. Both inputs and outputs are also TTL-compatible, although the inputs cannot tolerate voltage swings above the $V_{IN}3$ (max.) specification.

System support components which use TTL-compatible inputs will interface to the embedded Pentium processor without extra logic. This is because the embedded Pentium processor drives according to the 5 V TTL specification (but not beyond 3.3 V).

For embedded Pentium processor inputs, the voltage must not exceed the 3.3 V $V_{IN}3$ (max.) specification. System support components can consist of 3.3 V devices or open-collector devices. In an open-collector configuration, the external resistor should be biased to V_{CC3}.

All pins, other than the CLK and PICCLK of the embedded Pentium processor (100/133/166), are 3.3 V-only. If an 8259A interrupt controller is used, for example, the system must provide level converters between the 8259A and the embedded Pentium processor.

The CLK and PICCLK inputs of the embedded Pentium processor (100/133/166) are 5 V tolerant. This allows a 5 V clock driver to be used for the embedded Pentium processor (100/133/166). These inputs, however, are not 5 V tolerant on the embedded Pentium processor with MMX technology. The embedded Pentium processor with MMX technology CLK and PICCLK inputs are 3.3 V tolerant only. A 3.3 V clock driver should be used in systems designed to support both the embedded Pentium processor with MMX technology and embedded Pentium processor (100/133/166).

7.1.2.5 NC/INC and Unused Inputs

All NC and INC pins must remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Unused active low inputs of the embedded Pentium processor with MMX technology should be connected to V_{CC3}, and unused active low inputs of the embedded Pentium processor (100/133/166) should be connected to V_{CC}. Unused active high inputs should be connected to V_{SS} (ground).

7.1.3 Buffer Models

The structure of the buffer models for the embedded Pentium processor with MMX technology and the embedded Pentium processor (100/133/166) are identical. Some of the values of the components have changed to reflect the minor manufacturing process and package differences between the processors. The system should see insignificant differences between the AC behavior of the Pentium Processor with MMX Technology and the embedded Pentium processor (100/133/166).

Simulation of AC timings using the embedded Pentium processor buffer models is recommended to ensure robust system designs. Pay specific attention to the signal quality restrictions imposed by 3.3 V buffers.

I/O Buffer Models 8

This chapter describes the 3.3 V I/O buffer models of the embedded Pentium® processor.

The first order I/O buffer model is a simplified representation of the complex input and output buffers used in the Pentium processor family. Figure 8-1 and Figure 8-2 show the structure of the input buffer model and Figure 8-3 shows the output buffer model. Table 8-1 and Table 8-2 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.

The following two models represent the input buffer models. The first model, Figure 8-1, represents all of the input buffers of the Pentium processor except for a special group of input buffers. The second model, Figure 8-2, represents these special buffers. These buffers are: AHOLD, EADS#, KEN#, WB/WT#, INV, NA#, EWBE#, BOFF#, CLK, and PICCLK.

The embedded Pentium processor supports 5 V tolerant buffers on the CLK and PICCLK pins. It is important to note that all inputs of the embedded Pentium processor with MMX™ technology are 3.3 V tolerant only. The CLK and PICCLK pins are not 5 V tolerant on the embedded Pentium processor with MMX technology.

Figure 8-1. Input Buffer Model, Except Special Group

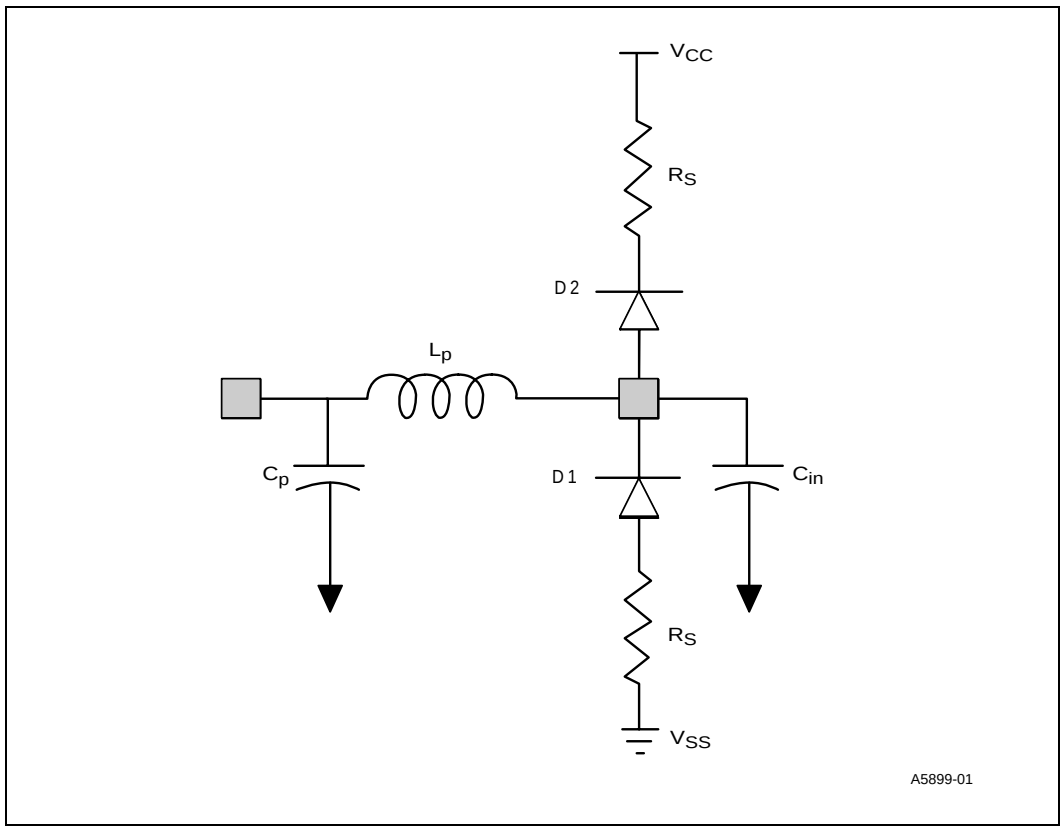


Figure 8-2. Input Buffer Model for Special Group

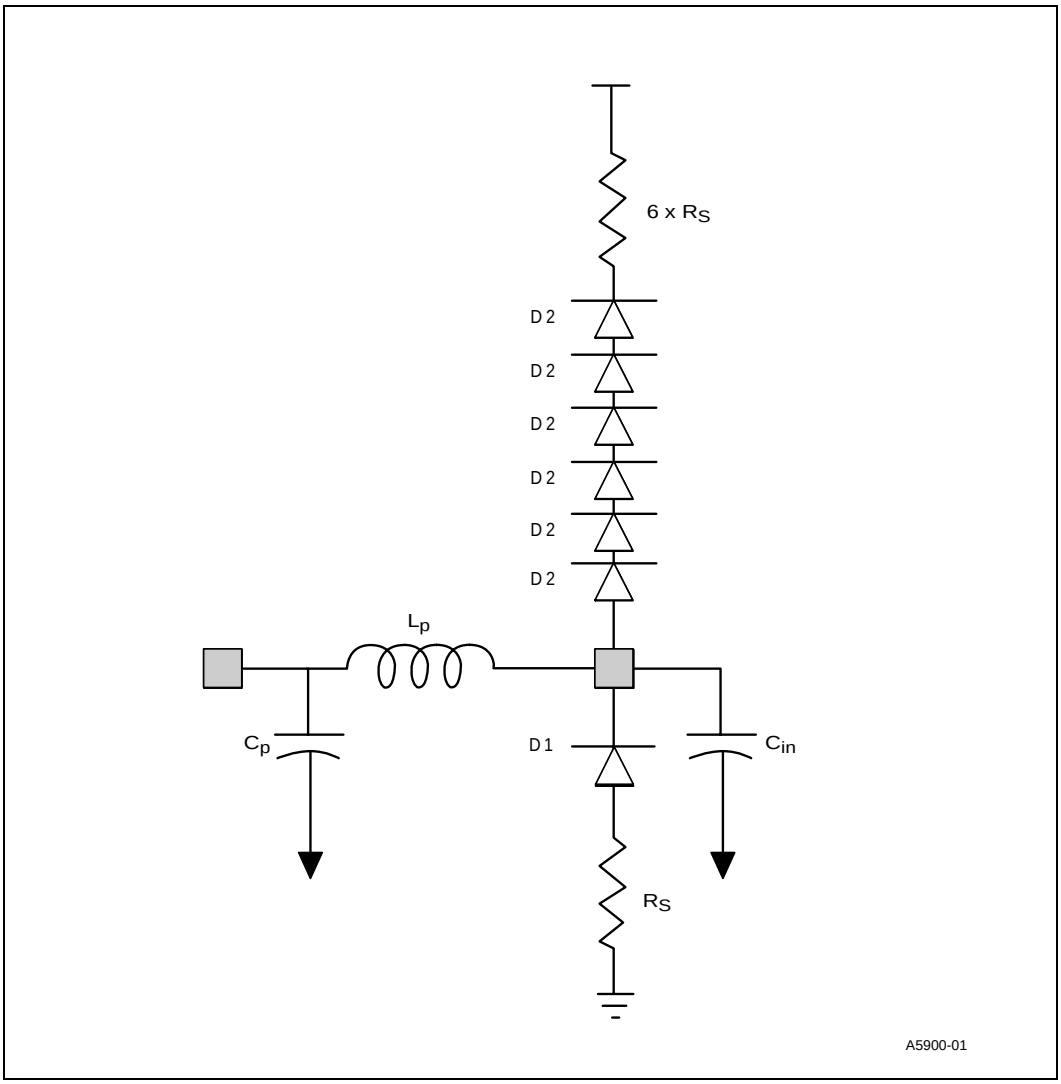


Table 8-1. Parameters Used in the Specification of the First Order Input Buffer Mode

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 8-3 shows the structure of the output buffer model. This model is used for all of the output buffers of the Pentium processor.

Figure 8-3. First Order Output Buffer Model

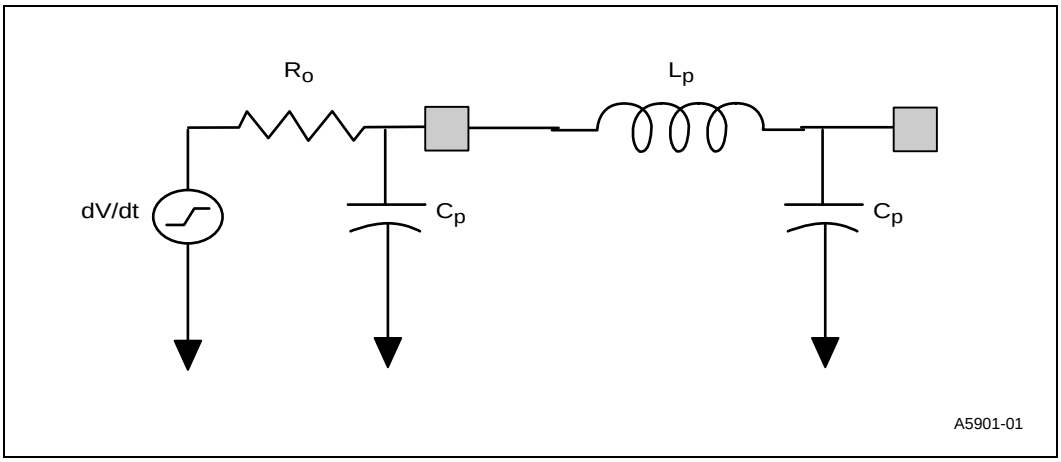


Table 8-2. Parameters Used in the Specification of the First Order Output Buffer Mode

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.

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, that 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.

8.1 Buffer Model Parameters

This section gives the parameters for each Pentium processor input, output, and bidirectional signals, and the settings for the configurable buffers.

In dual processor mode, a few signals change from output signals to I/O signals. These signals are: ADS#, M/IO#, D/C#, W/R#, LOCK#, CACHE#, SCYC, HLDA, HIT#, and HITM#. When simulating these signals use the correct operation of the buffer while in DP mode.

Some pins on the processor have selectable buffer sizes to allow for faster switching of the buffer in heavily loaded environments. The buffer selection is done through the setting of configuration pins at power on RESET. Once selected, these cannot be changed without a power on RESET. The

BUSCHK# and BRDYC# pins are used to select the different buffer size. All configurable pins get set to the selected buffer size. There is no selection for specific signal groups to get specific buffers. Keep in mind that the largest buffer size is not always the best selection especially in a lightly loaded environment. AC timing and signal quality simulations should be done to ensure that the buffers used meet required timing and signal quality specifications for the components that will be used in the specific board design.

The pins with selectable buffer sizes use the configurable output buffer EB2. Table 8-3 shows the drive level required at falling edge of RESET, to select the buffer strength. Once selected, the buffer size cannot be changed without a power on RESET. The buffer sizes selected should be the appropriate size required, otherwise AC timings might not be met, or too much overshoot and ringback may occur. There are no other selection choices, all the configurable buffers get set to the same size at the same time.

Table 8-3 shows the proper settings on BRDYC# and BUSCHK# for proper buffer size selection.

Table 8-3. Buffer Selection Chart

Environment	BRDYC#	BUSCHK#	Buffer Selection
Typical Stand Alone Component	1	X	EB2
Loaded Component	0	1	EB2A
Heavily Loaded Component	0	0	EB2B

NOTE: X is a “don’t care” (0 or 1). Please refer to Table 8-4 for the groupings of the buffers

Table 8-4. Signal to Buffer Type

Signals	Type	Driver Buffer Type	Receiver Buffer Type
CLK	I		ER0
A20M#, AHOLD, BF1–BF0, BOFF#, BRDY#, BRDYC#, BUSCHK#, EADS#, EWBE#, FLUSH#, FRCMC#[2], HOLD, IGNNE#, INIT, INTR, INV, KEN#, NA#, NMI, PEN#, PICCLK, R/S#, RESET, SMI#, STPCLK#, TCK, TDI, TMS, TRST#, WB/WT#	I		ER1
ADSC#, APCHK#, BE7#–BE5#, BP3–BP2, BREQ, FERR#, IERR#, PCD, PCHK#, PM0/BP0, PM1/BP1, PRDY, PWT, SMIACT#, TDO, D/P#	O	ED1	
A31–A21, AP, BE4#–BE0#, CACHE#, D/C#, D63–D0, DP7–DP0, HLDA, LOCK#, M/IO#, PBGNT#, PBREQ#, PHIT#, PHITM#, SCYC	I/O	EB1	EB1
A20–A3, ADS#, HITM#, W/R#	I/O	EB2/A/B	EB2
HIT#	I/O	EB3	EB3
PICD0, PICD1	I/O	EB4	EB4

NOTES:
1. VCC2DET# has no buffer model – it is simply a short to V_{SS} on the embedded Pentium® processor with MMX™ technology . This pin is an INC on the embedded Pentium processor.
2. FRCMC# is defined only for the embedded Pentium processor.

The input, output and bidirectional buffers values are listed in Table 8-5. Table 8-5 contains listings for all three types; do not get them confused during simulation. 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 8-5. Input, Output and Bidirectional Buffer Model Parameters

Buffer Type	Transition	dV/dt (V/nsec)		Ro (Ohms)		Cp (pF)		Lp (nH)		Co/Cin (pF)	
		min	max	min	max	min	max	min	max	min	max
ER0	Rising					3.0	5.0	4.0	7.2	0.8	1.2
(input)	Falling					3.0	5.0	4.0	7.2	0.8	1.2
ER1	Rising					1.1	6.1	4.7	15.3	0.8	1.2
(input)	Falling					1.1	6.1	4.7	15.3	0.8	1.2
ED1	Rising	3/3.0	3.7/0.9	21.6	53.1	1.1	8.2	4.0	17.7	2.0	2.6
(output)	Falling	3/2.8	3.7/0.8	17.5	50.7	1.1	8.2	4.0	17.7	2.0	2.6
EB1	Rising	3/3.0	3.7/0.9	21.6	53.1	1.3	8.7	4.0	18.7	2.0	2.6
(bidir)	Falling	3/2.8	3.7/0.8	17.5	50.7	1.3	8.7	4.0	18.7	2.0	2.6
EB2	Rising	3/3.0	3.7/0.9	21.6	53.1	1.3	8.3	4.4	16.7	9.1	9.7
(bidir)	Falling	3/2.8	3.7/0.8	17.5	50.7	1.3	8.3	4.4	16.7	9.1	9.7
EB2A	Rising	3/2.4	3.7/0.9	10.1	22.4	1.3	8.3	4.4	16.7	9.1	9.7
(bidir)	Falling	3/2.4	3.7/0.9	9.0	21.2	1.3	8.3	4.4	16.7	9.1	9.7
EB2B	Rising	3/1.8	3.7/0.7	5.5	12.9	1.3	8.3	4.4	16.7	9.1	9.7
(bidir)	Falling	3/1.8	3.7/0.7	4.6	12.3	1.3	8.3	4.4	16.7	9.1	9.7
EB3	Rising	3/3.0	3.7/0.9	21.6	53.1	1.9	7.5	9.9	14.3	3.3	3.9
(bidir)	Falling	3/2.8	3.7/0.8	17.5	50.7	1.9	7.5	9.9	14.3	3.3	3.9
EB4	Rising	3/3.0	3.7/0.9	100K†	100K†	2.0	6.9	5.8	14.6	5.0	7.0
(bidir)	Falling	3/2.8	3.7/0.8	17.5	50.7	2.0	6.9	5.8	14.6	5.0	7.0

† The buffer is an open drain. For simulation purposes it should be modeled by a very large internal resistor with an additional external pull-up.

Table 8-6. Input Buffer Model Parameters: D (Diodes)

Symbol	Parameter	D1	D2
IS	Saturation Current	1.4e–14 A	2.78e–16 A
N	Emission Coefficient	1.19	1.00
RS	Series Resistance	6.5 ohms	6.5 ohms
TT	Transit Time	3 ns	6 ns
VJ	PN Potential	0.983 V	0.967 V
CJ0	Zero Bias PN Capacitance	0.281 pF	0.365 pF
M	PN Grading Coefficient	0.385	0.376

8.2 Signal Quality Specifications

Signals driven by the system into the Pentium processor 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. There are two signal quality parameters: Ringback and Settling Time. For more information, see “CLK/PICCLK Signal Quality Specification for the Pentium® Processor with MMX™ Technology” on page 8-8.

8.2.1 Ringback

Excessive ringback can contribute to long-term reliability degradation of the Pentium processor, and can cause false signal detection. Ringback is simulated at the input pin of a component using the input buffer model. Ringback can be simulated with or without the diodes that are in the input buffer model.

Ringback is the absolute value of the maximum voltage at the receiving pin below V_{CC} (or above V_{SS}) relative to V_{CC} (or V_{SS}) level after the signal has reached its maximum voltage level. The input diodes are assumed present.

- Maximum Ringtrack on Inputs = 0.8 V (with diodes)

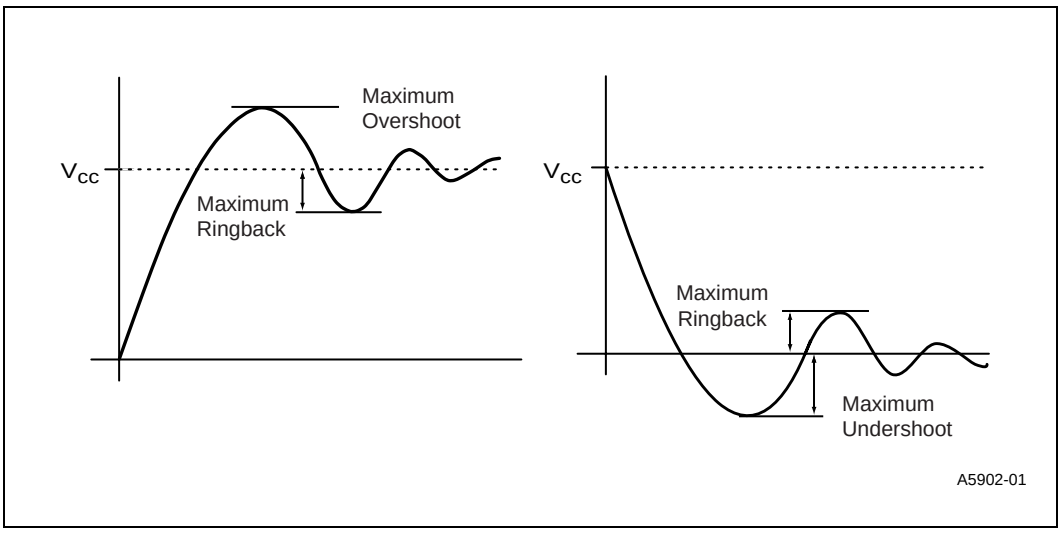
If simulated without the input diodes, follow the Maximum Overshoot/Undershoot specification. By meeting the overshoot/undershoot specification, the signal is guaranteed not to ringback excessively.

If simulated with the diodes present in the input model, follow the maximum ringback specification.

Overshoot (Undershoot) is the absolute value of the maximum voltage above V_{CC} (below V_{SS}). The guideline assumes the absence of diodes on the input.

- Maximum Overshoot/Undershoot on 3.3 V Pentium processor Inputs (including CLK and PICCLK) = 1.4 V above V_{CC3} (without diodes)

Figure 8-4. Overshoot/Undershoot and Ringback Guidelines



8.2.2 Settling Time

The settling time is defined as the time a signal requires at the receiver to settle within 10% of V_{CC} or V_{SS}. Settling time is the maximum time allowed for a signal to reach within 10% of its final value.

Most available simulation tools are unable to simulate settling time so that it accurately reflects silicon measurements. On a physical board, second-order effects and other effects serve to dampen the signal at the receiver. Because of all these concerns, settling time is a recommendation or a tool for layout tuning and not a specification.

Settling time is simulated at the slow corner, to make sure that there is no impact on the flight times of the signals if the waveform has not settled. Settling time may be simulated with the diodes included or excluded from the input buffer model. If diodes are included, settling time recommendation will be easier to meet.

Although simulated settling time has not shown good correlation with physical, measured settling time, settling time simulations can still be used as a tool to tune layouts.

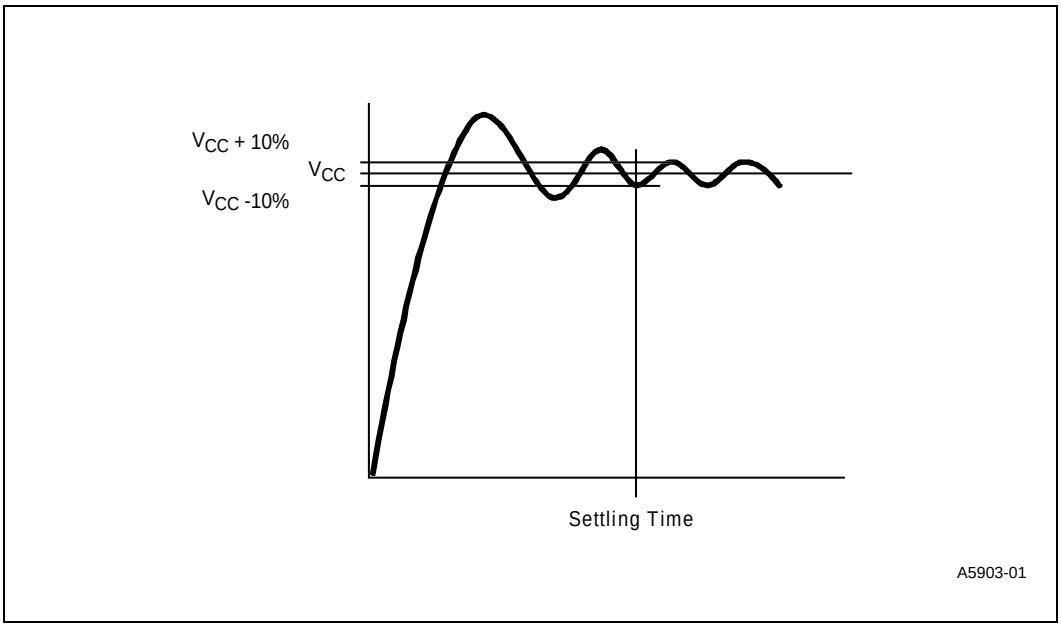
Use the following procedure to verify board simulation and tuning with concerns for settling time.

1. Simulate settling time at the slow corner for a particular signal.
2. If settling time violations occur, simulate signal trace with DC diodes in place at the receiver pin. The DC diode behaves almost identically to the actual (non-linear) diode on the part as long as excessive overshoot does not occur.
3. If settling time violations still occur, simulate flight times for five consecutive cycles for that particular signal.
4. If flight time values are consistent over the five simulations, settling time should not be a concern. If however, flight times are not consistent over the five simulations, tuning of the layout is required.

Note that, for signals that are allocated two cycles for flight time, the recommended settling time is doubled.

Maximum Settling Time to within 10% of V_{CC} is: 12.5 ns @66 MHz.

Figure 8-5. Settling Time



8.2.3 CLK/PICCLK Signal Quality Specification for the Pentium® Processor with MMX™ Technology

The maximum overshoot, maximum undershoot, overshoot threshold duration, undershoot threshold duration, and maximum ringback specifications for CLK/PICCLK are described below:

Maximum Overshoot And Maximum Undershoot Specification: The maximum overshoot of the CLK/PICCLK signals should not exceed V_{CC3}, nominal +0.9 V. The maximum undershoot of the CLK/PICCLK signals must not drop below –0.9 V.

Overshoot Threshold Duration Specification: The overshoot threshold duration is defined as the sum of all time during which the CLK/PICCLK signal is above V_{CC3}, nominal +0.5 V within a single clock period. The overshoot threshold duration must not exceed 20% of the period.

Undershoot Threshold Duration Specification: The undershoot threshold duration is defined as the sum of all time during which the CLK/PICCLK signal is below –0.5 V within a single clock period. The undershoot threshold duration must not exceed 20% of the period.

Maximum Ringback Specification: The maximum ringback of CLK/PICCLK associated with their high states (overshoot) must not drop below V_{CC3} –0.8 V as shown in Figure 8-7. Similarly, the maximum ringback of CLK/PICCLK associated with their low states (undershoot) must not exceed 0.8 V as shown in Figure 8-9.

Refer to Table 8-7 and Table 8-8 for a summary of the clock overshoot and undershoot specifications for the embedded Pentium processor with MMX technology.

Table 8-7. Overshoot Specification Summary

Specification Name	Value	Units	Notes
Threshold Level	V_{CC3}, nominal +0.5	V	(1) (2)
Maximum Overshoot Level	V_{CC3}, nominal +0.9	V	(1) (2)
Maximum Threshold Duration	20% of clock period above threshold voltage	ns	(2)
Maximum Ringback	V_{CC3}, nominal –0.8	V	(1) (2)

NOTES:
1. V_{CC3}, nominal refers to the voltage measured at the bottom side of the V_{CC3} pins.
2. See Figure 8-6 and Figure 8-7.

Table 8-8. Undershoot Specification Summary

Specification Name	Value	Units	Notes
Threshold Level	–0.5	V	(1)
Minimum Undershoot Level	–0.9	V	(1)
Maximum Threshold Duration	20% of clock period below threshold voltage	ns	(1)
Maximum Ringback	0.8	V	(1)

NOTE:
1. See Figure 8-8 and Figure 8-9.

8.2.3.1 Clock Signal Measurement Methodology

The waveform of the clock signals should be measured at the bottom side of the processor pins using an oscilloscope with a 3 dB bandwidth of at least 20 MHz (100 ms/s digital sampling rate). There should be a short isolation ground lead attached to a processor pin on the bottom side of the board. A 1 MOhm probe with loading of less than 1 pF (e.g., Tektronics 6243 or Tektronics 6245) is recommended. The measurement should be taken at the CLK (AK18) and PICCLK (H34) pins and their nearest V_{SS} pins (AM18 and H36, respectively).

Maximum Overshoot, Maximum Undershoot And Maximum Ringback Specifications: The display should show continuous sampling (e.g., infinite persistence) of the waveform at 500 mV/div and 5 ns/div (for CLK) or 20 ns/div (for PICCLK) for a recommended duration of approximately five seconds. Adjust the vertical position to measure the maximum overshoot and associated ringback with the largest possible granularity. Similarly, readjust the vertical position to measure the maximum undershoot and associated ringback. There is no allowance for crossing the maximum overshoot, maximum undershoot or maximum ringback specifications.

Overshoot Threshold Duration Specification: A snapshot of the clock signal should be taken at 500 mV/div and 500 ps/div (for CLK) or 2 ns/div (for PICCLK). Adjust the vertical position and horizontal offset position to view the threshold duration. The overshoot threshold duration is defined as the sum of all time during which the clock signal is above V_{CC3}, nominal +0.5 V within a single clock period. The overshoot threshold duration must not exceed 20% of the period.

Undershoot Threshold Duration Specification: A snapshot of the clock signal should be taken at 500 mV/div and 500 ps/div (for CLK) or 2 ns/div (for PICCLK). Adjust the vertical position and horizontal offset position to view the threshold duration. The undershoot threshold duration is defined as the sum of all time during which the clock signal is below –0.5 V within a single clock period. The undershoot threshold duration must not exceed 20% of the period.

These overshoot and undershoot specifications are illustrated in Figures 8-6 to 8-9.

Figure 8-6. Maximum Overshoot Level, Overshoot Threshold Level, and Overshoot Threshold Duration

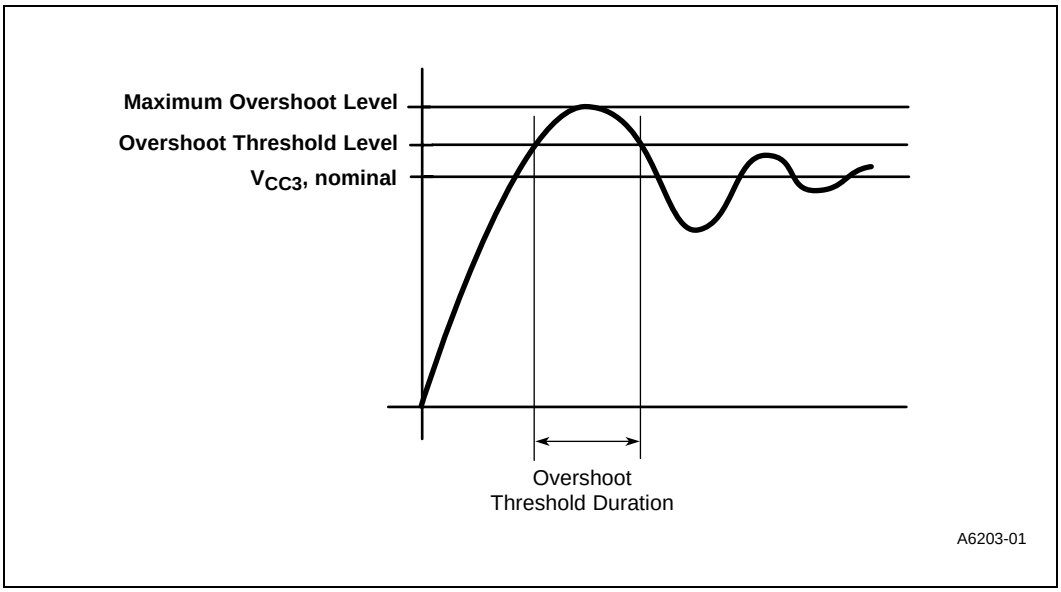


Figure 8-7. Maximum Ringback Associated with the Signal High State

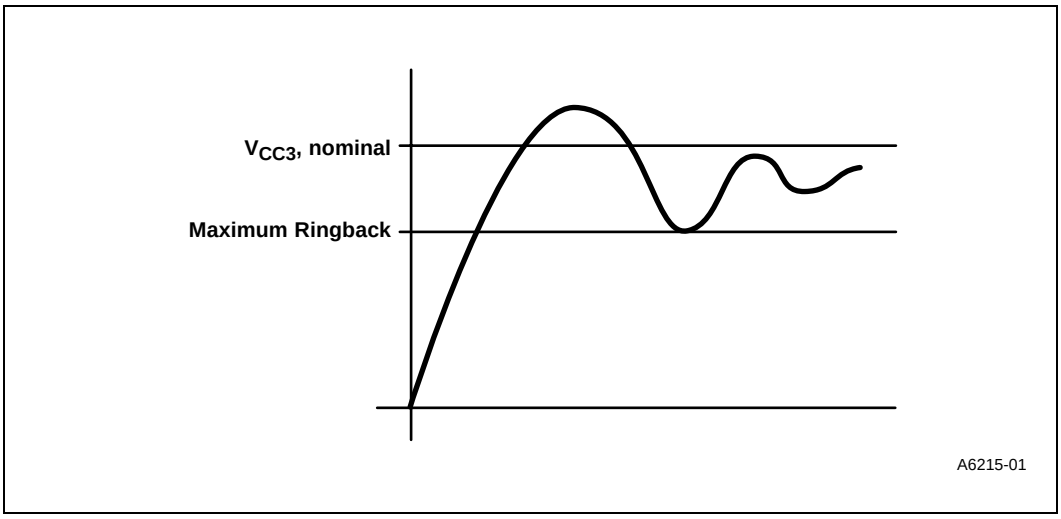


Figure 8-8. Maximum Undershoot Level, Undershoot Threshold Level, and Undershoot Threshold Duration

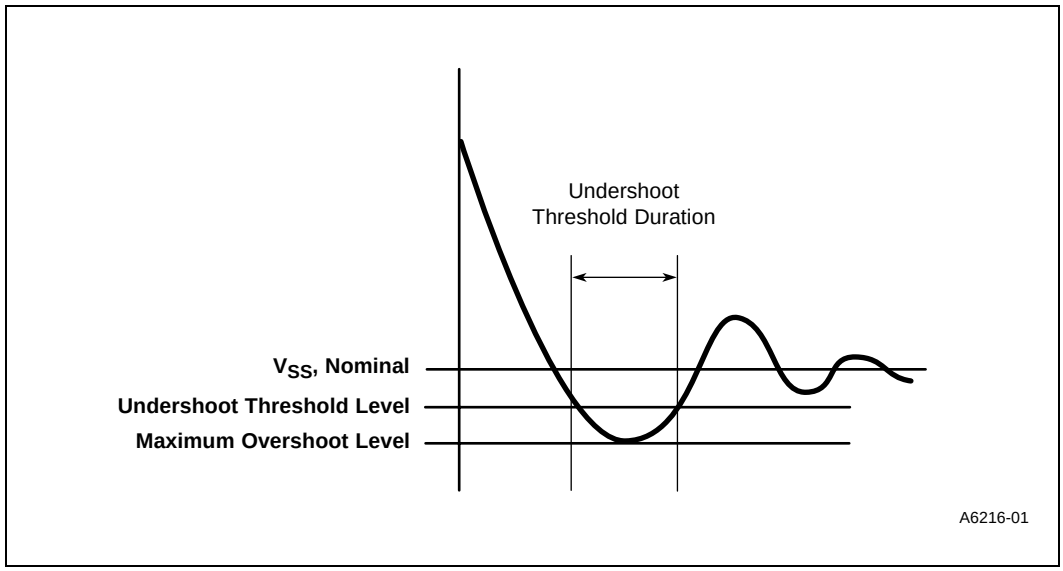
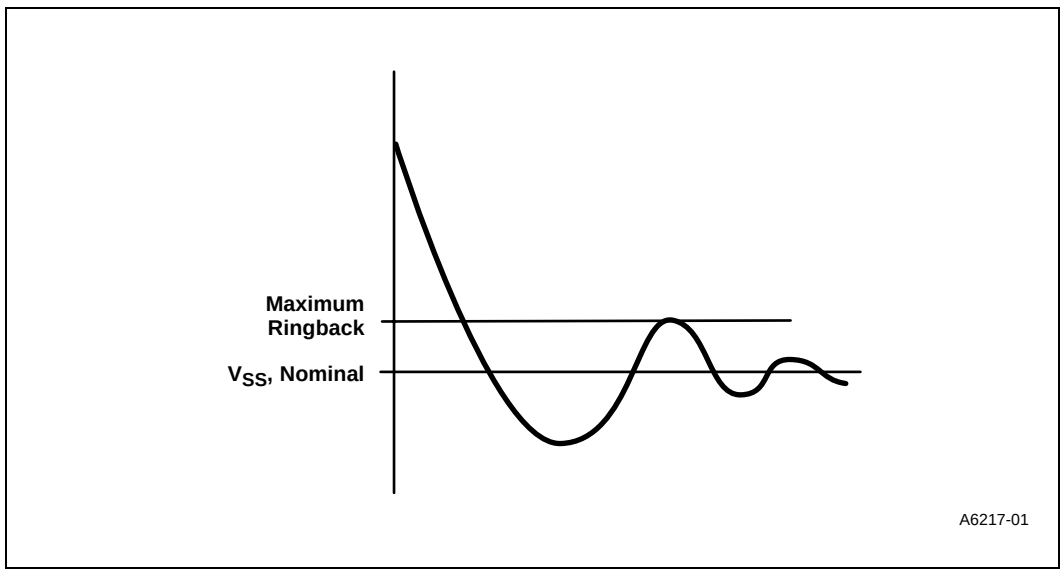


Figure 8-9. Maximum Ringback Associated with the Signal Low State



Testability 9

This chapter describes the features which are included in the embedded Pentium® processor for the purpose of enhancing testability. The capability of the Intel486™ processor test hooks are included in the embedded Pentium processor; however, some are implemented differently. In addition, new test features were added to assure timely testing and production of the system product.

Internal component testing through the Built-In Self-Test (BIST) feature provides 100% single stuck at fault coverage of the microcode ROM and large PLAs. Some testing of the instruction cache, data cache, Translation Lookaside Buffers (TLBs), and Branch Target Buffer (BTB) is also performed. In addition, the constant ROMs are checked.

Three-State Test Mode and the IEEE 1149.1 “Test Access Port and Boundary Scan” mechanism are included to facilitate testing of board connections.

See “Testability And Test Registers” on page 14-3 for more information regarding the testing of the on-chip caches, translation lookaside buffers, branch target buffer, second level caches, the superscalar architecture, and internal parity checking through the test registers.

9.1 Built-in Self-test (BIST)

Self-test is initiated by driving the INIT pin high when RESET transitions from high to low. No bus cycles are run by the embedded Pentium processor during self-test. The duration of self-test is approximately 2^{19} core clocks. Approximately 70% of the devices in the processor are tested by BIST. The BIST consists of two parts: hardware self-test and microcode self-test.

During the hardware portion of BIST, the microcode and all large PLAs are tested. All possible input combinations of the microcode ROM and PLAs are tested.

The constant ROMs, BTB, TLBs and all caches are tested by the microcode portion of BIST. The array tests (caches, TLBs and BTB) have two passes. On the first pass, data patterns are written to arrays, read back and checked for mismatches. The second pass writes the complement of the initial data pattern, reads it back and checks for mismatches. The constant ROMs are tested by using the microcode to add various constants and check the result against a stored value.

Upon completion of BIST, the cumulative result of all tests are stored in the EAX register. When EAX contains 0H, all checks passed; any non-zero result indicates a faulty unit. Note that if an internal parity error is detected during BIST, the processor will assert the IERR# pin and attempt to shutdown.

9.2 Three-state Test Mode

When the FLUSH# pin is sampled low in the clock prior to the RESET pin going from high to low, the processor enters three-state test mode. The processor floats all of its output pins and bidirectional pins including pins which are never floated during normal operation (except TDO). Three-state test mode can be initiated in order to facilitate testing of board connections. The processor remains in three-state test mode until the RESET pin is toggled again.

In a dual-processor system, the private interface pins are not floated in Three-state Test mode. These pins are PBREQ#, PBGNT#, PHIT#, and PHITM#.

Note: There are several pins that have internal pullups or pulldowns attached that show these pins going high or low, respectively, during Three-state Test mode. There is one pin, PICD1, that has an internal pulldown attached that shows this pin going low during Three-state Test mode. The five pins that have pullups are PHIT#, PHITM#, PBREQ#, PBGNT#, and PICD0. There are two other pins that have pullups attached during dual processor mode, HIT# and HITM#. The pullups on these pins (except HIT#) have a value of about 30 KOhms, HIT# is about 2 KOhms.

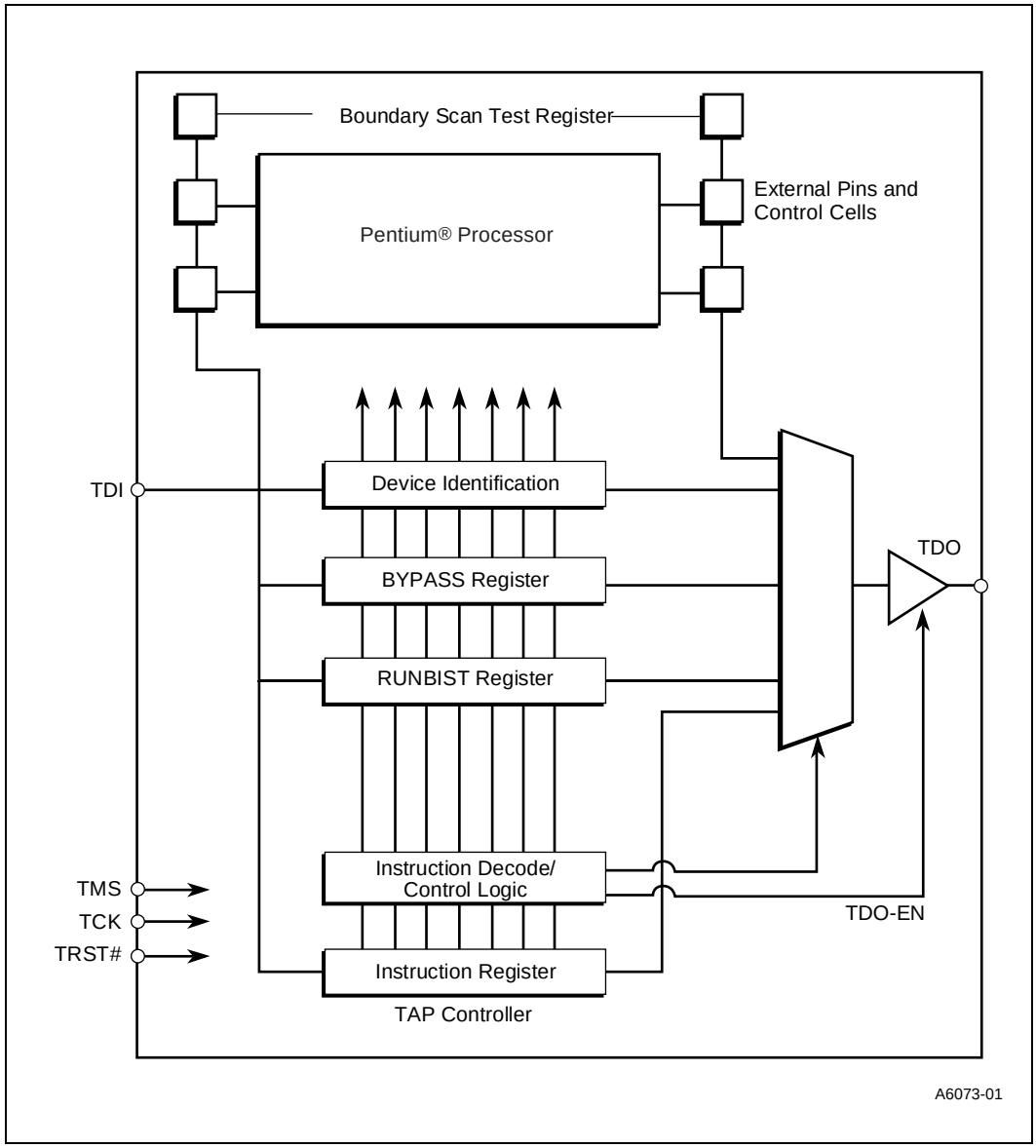
9.3 IEEE 1149.1 Test Access Port and Boundary Scan Mechanism

The IEEE Standard Test Access Port and Boundary Scan Architecture (Standard 1149.1) is implemented in the embedded Pentium processor. This feature allows board manufacturers to test board interconnects by using “boundary scan,” and to test the processor itself through BIST. All output pins are three-stateable through the IEEE 1149.1 mechanism.

9.3.1 Test Access Port (TAP)

The processor Test Access Port (TAP) contains a TAP controller, a Boundary Scan Register, four input pins (TDI, TCK, TMS, and TRST#) and one output pin (TDO). The TAP controller consists of an Instruction Register, a Device ID Register, a Bypass Register, a Runbist Register and control logic. See Figure 9-1 for the TAP Block Diagram.

Figure 9-1. Test Access Port Block Diagram



9.3.1.1 TAP Pins

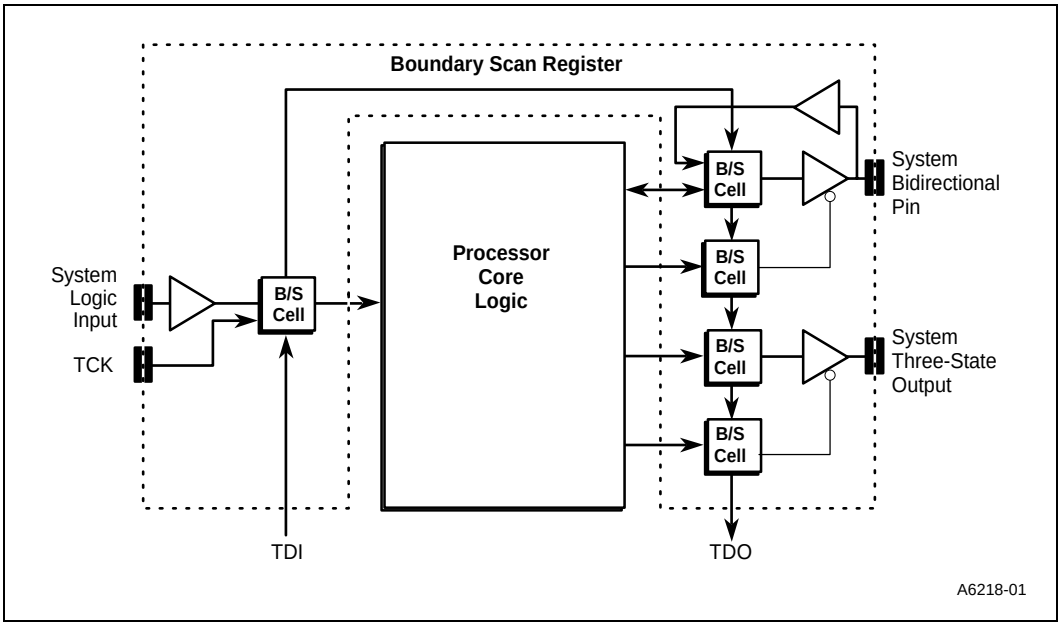
As mentioned in the previous section, the TAP includes four input pins and one output pin. TDI (test data in) is used to shift data or instructions into the TAP in a serial manner. TDO (test data out) shifts out the response data. TMS (test mode select) is used to control the state of the TAP controller. TCK is the test clock. The TDI and TMS inputs are sampled on the rising edge of this TCK. Asserting TRST# will force the TAP controller into the Test Logic Reset State (see the TAP controller state diagram, Figure 9-4). The input pins (TDI, TMS, TCK, and TRST#) have pullup resistors.

9.3.1.2 TAP Registers

Boundary Scan Register

The IEEE standard requires that an extra single bit shift register be inserted at each pin on the processor. These single bit shift registers are connected into a long shift register, the Boundary Scan Register. Therefore, the Boundary Scan Register is a single shift register path containing the boundary scan cells that are connected to all input and output pins of the processor. Figure 9-2 shows the logical structure of the Boundary Scan Register. While output cells determine the value of the signal driven on the corresponding pin, input cells only capture data; they do not affect the normal operation of the device (the INTEST instruction is not supported by the embedded Pentium processor). Data is transferred without inversion from TDI to TDO through the Boundary Scan Register during scanning. The Boundary Scan Register can be operated by the EXTEST and SAMPLE/PRELOAD instructions. The Boundary Scan Register order is defined later in this chapter.

Figure 9-2. Boundary Scan Register



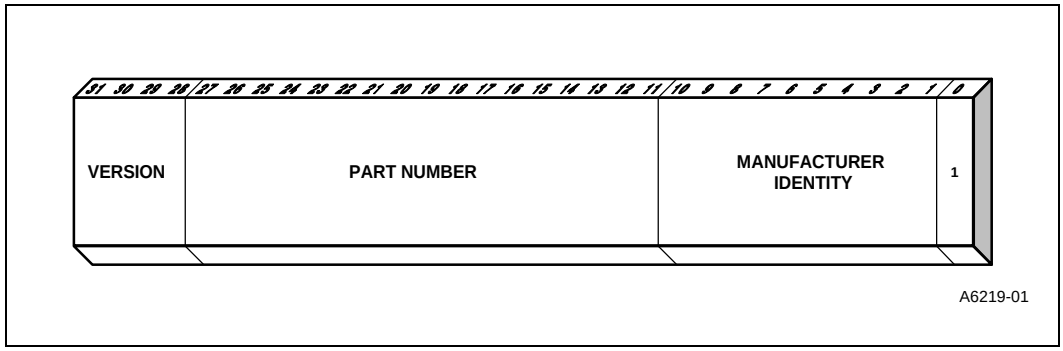
Bypass Register

The Bypass Register is a one-bit shift register that provides the minimal length path between TDI and TDO. This path can be selected when no test operation is being performed by the component to allow rapid movement of test data to and from other components on the board. While the bypass register is selected data is transferred from TDI to TDO without inversion. The Bypass Register loads a logic 0 at the start of a scan cycle.

Device ID Register

The Device Identification Register contains the manufacturer's identification code, part number code, and version code in the format shown in Figure 9-3. It is selected to be connected between TDI and TDO by using the IDCODE instruction.

Figure 9-3. Format of the Device ID Register



The processor has divided the 16-bit part number into three fields. The upper 7 bits are used to define the product type (examples: Cache, processor architecture). The middle 4 bits are used to represent the generation or family (examples: Intel486 processor, embedded Pentium processor). The lower 5 bits are used to represent the model (examples: SX, DX). Using this definition, the embedded Pentium processor ID code is shown in Table 9-1.

The version field is used to indicate the stepping ID.

Table 9-1. Device ID Register Values

Processor	Stepping	Version	Part Number			Manufacturing ID	"1"	Entire Code
			Product Type	Generation	Model			
Pentium processor (100/133/166)	x	xH	01H	05H	04H	09H	1	x82A4013H
Pentium processor with MMX™ technology	x	xH	01H	05H	03H	09H	1	x82A3013H

Runbist Register

The Runbist Register is a one bit register used to report the results of the embedded Pentium processor BIST when it is initiated by the RUNBIST instruction. This register is loaded with "0" upon successful completion of BIST.

Instruction Register

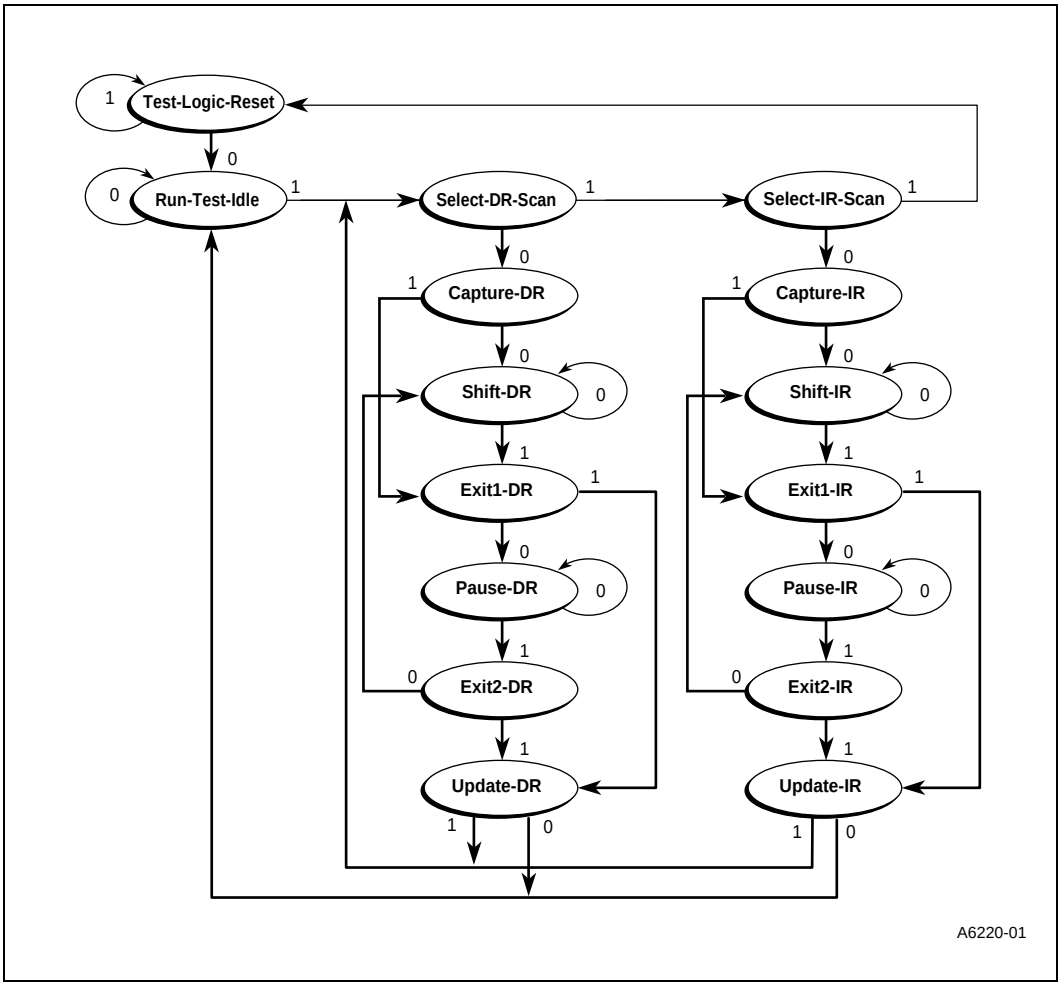
This register is 13 bits wide. The command field (the lower 4 bits of instruction) is used to indicate one of the following instructions: EXTEST, IDCODE, RUNBIST, SAMPLE/PRELOAD and BYPASS. The upper 9 bits are reserved.

The most significant bit of the Instruction Register is connected to TDI, the least significant to TDO.

9.3.1.3 TAP Controller State Diagram

Figure 9-4 shows the 16-state TAP controller state diagram. A description of each state follows. Note that the state machine contains two main branches to access either data or instruction registers.

Figure 9-4. TAP Controller State Diagram



Test-Logic-Reset State

In this state, the test logic is disabled so that normal operation of the device can continue unhindered. During initialization, the processor initializes the instruction register such that the IDCODE instruction is loaded.

No matter what the original state of the controller, the controller enters Test-Logic-Reset state when the TMS input is held high (logic 1) for at least five rising edges of TCK. The controller remains in this state while TMS is high. The TAP controller is forced to enter this state when the TRST# pin is asserted (with TCK toggling or TCK at a high logic value). The processor automatically enters this state at power-up.

Run-Test/Idle State

This is a controller state between scan operations. Once in this state, the controller remains in this state as long as TMS is held low. In devices supporting the RUNBIST instruction, the BIST is performed during this state and the result is reported in the Runbist Register. For instructions not causing functions to execute during this state, no activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is high and a rising edge is applied to TCK, the controller moves to the Select-DR state.

Select-DR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-DR state, and a scan sequence for the selected test data register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Select-IR-Scan state.

The instruction does not change in this state.

Capture-DR State

In this state, the Boundary Scan Register captures input pin data if the current instruction is EXTEST or SAMPLE/PRELOAD. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or the Shift-DR state if TMS is low.

Shift-DR State

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or remains in the Shift-DR state if TMS is low.

Exit1-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

Pause-DR State

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. An example use of this state could be to allow a tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-DR state.

Exit2-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

Update-DR State

The Boundary Scan Register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE/PRELOAD instructions. When the TAP controller is in this state and the Boundary Scan Register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in the test data register selected by the current instruction retains their previous value during this state. The instruction does not change in this state.

Select-IR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-IR state, and a scan sequence for the instruction register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Test-Logic-Reset state. The instruction does not change in this state.

Capture-IR State

In this controller state the shift register contained in the instruction register loads a fixed value on the rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or the Shift-IR state if TMS is held low.

Shift-IR State

In this state the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or remains in the Shift-IR state if TMS is held low.

Exit1-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

Pause-IR State

The pause state allows the test controller to temporarily halt the shifting of data through the instruction register.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-IR state.

Exit2-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain their previous value.

9.3.2 Boundary Scan

The IEEE Standard 1149.1 Boundary Scan is implemented using the Test Access Port and TAP Controller as described above. The embedded Pentium processor implements all of the required boundary scan features as well as some additional features. The required pins (all 3.3 V) are: TDI, TDO, TCK and TMS. The required registers are: Boundary Scan, Bypass, and the Instruction Register. Required instructions include: BYPASS, SAMPLE/PRELOAD and EXTEST. The additional pin, registers, and instructions are implemented to add additional test features.

On the board level, the TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes. The testing is controlled through the TAP Controller State machine that can be implemented with automatic test equipment or a PLD.

On power up the TAP controller is automatically initialized to the test logic reset state (test logic disabled), so normal processor behavior is the default. The Test Logic Reset State is also entered when TRST# is asserted, or when TMS is high for five or more consecutive TCK clocks.

To implement boundary scan, the TDO of one device is connected to TDI of the next in a daisy-chain fashion. This allows all of the I/O of the devices on this chain to be accessed through a long shift register. TMS and TCK are common to all devices.

The Boundary Scan Register for the embedded Pentium processor contains a cell for each pin.

The following is the bit order of the embedded Pentium processor with MMX technology Boundary Scan Register (left to right, top to bottom):

TDO→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, M/IO#, CACHE#, EWBE#, INV, AHOLD, KEN#, BRDYC#, BRDY#, BOFF#, NA#, Disbus†, Dismisch†, Disbusl†, Dismisc†, Disua2bus†, Disua1bus†, Dismisca†, Dismiscf†, WB/WT#, HOLD, PHITM#, PHIT#, PBREQ#, PBGNT#, SMIACT#, PRDY, PCHK#, APCHK#, BREQ, HLDA, AP, LOCK#, ADSC#, PCD, PWT, D/C#, EADS#, ADS#, HITM#, HIT#, W/R#, 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, D/P#, NMI, RS#, INTR, SMI#, IGNNE#, INIT, PEN#, FRCMC#, Reserved, Reserved, Reserved, Reserved, Reserved, STPCLK#, Reserved, Reserved, Reserved, Reserved, Reserved, Reserved, Reserved →TDI

The following is the bit order of the embedded Pentium processor (100/133/166) 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#, PM0/BP0, PM1/BP1, BP2, BP3, M/IO#, CACHE#, EWBE#, INV, AHOLD, KEN#, BRDYC#, BRDY#, BOFF#, NA#, Disbus†, Dismisch†, Disbus1†, Dismisc†, Disua2bus†, Disua1bus†, Dismisca†, Dismiscfa†, WB/WT#, HOLD, PHITM#, PHIT#, PBREQ#, PBGNT#, SMIACT#, PRDY, PCHK#, APCHK#, BREQ, HLDA, AP, LOCK#, ADSC#, PCD, PWT, D/C#, EADS#, ADS#, HITM#, HIT#, W/R#, 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, D/P#, NMI, R/S#, INTR, SMI#, IGNNE#, INIT, PEN#, FRCMC#, Reserved, Reserved, BF0, BF1, Reserved, STPCLK#, Reserved, Reserved, Reserved, Reserved, Reserved, Reserved, CPUTYP → TDO

“Reserved” includes the no connect “NC” signals on the embedded Pentium processor.

The cells marked with an “†” are control cells that are used to select the direction of bidirectional 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:

For the embedded Pentium processor with MMX technology:

Disabus:	A31–A3, AP.
Disbus:	BE7–BE0#, CACHE#, SCYC, M/IO#, D/C#, W/R#, PWT, PCD.
Disbusl:	ADS#, ADSC#, LOCK#.
Dismisc:	APCHK#, PCHK#, PRDY, BP3, BP2, PM1/BP1, PM0/BP0.
Dismiscf:	D/P#.
Dismisch:	FERR#, SMIACT#, BREQ, HLDA, HIT#, HITM#.
Dismisca:	IERR#.
Disua1bus:	PBREQ#, PHIT#, PHITM#.
Disua2bus:	PBGNT#.
Diswr:	D63–D0, DP7–DP0.
Disapsba:	PICD1–PICD0

For the embedded Pentium processor (at 100/133/166 MHz):

Disabus:	A31–A3, AP
Dismiscfa:	D/P#, FERR#
Dismisca:	IERR#
Disua1buS:	PBREQ#, PHIT#, PHITM#
Disua2bus:	PBGNT#
Dismisc:	APCHK#, PHCK#, PRDY#, BP3, BP2, PM1/BP1, PM0/BP0
Disbus1:	ADS#, ADSC#, LOCK#
Dismisch:	HIT#, HITM#, HLDA, BREQ#, SMIACT#
Disbus:	SCYC, BE7#–BE0#, W/R#, D/C#, PWT, PCD, CACHE#, M/IO#
Diswr:	DP7–DP0, D63–D0
Disapsba:	PICD0, PICD1

9.3.2.1 Boundary Scan TAP Instruction Set

Table 9-2 shows the Boundary Scan TAP instructions and their instruction register encoding. A description of each instruction follows. The IDCODE and BYPASS instructions may also be executed concurrent with processor execution. The following instructions are not affected by the assertion of RESET: EXTEST, SAMPLE/PRELOAD, BYPASS, and IDCODE.

The instructions should be scanned in to the TAP port least significant bit first (bit 0 of the TAP Command field is the first bit to be scanned in).

Table 9-2. TAP Instruction Set and Instruction Register Encoding

Instruction Name	Instruction Register [Bits 12:4]	TAP Command Field [Bits 3:0]
EXTEST	XXXXXXXXX	0000
Sample/Preload	XXXXXXXXX	0001
IDCODE	XXXXXXXXX	0010
Private Instruction	XXXXXXXXX	0011
Private Instruction	XXXXXXXXX	0100
Private Instruction	XXXXXXXXX	0101
Private Instruction	XXXXXXXXX	0110
RUNBIST	XXXXXXXXX	0111
Private Instruction	XXXXXXXXX	1000
Private Instruction	XXXXXXXXX	1001
Private Instruction	XXXXXXXXX	1010
HI-Z	XXXXXXXXX	1011
Private Instruction	XXXXXXXXX	1100
BYPASS	XXXXXXXXX	1111

The TAP Command field encodings not listed in Table 9-2 (1101, 1110) are unimplemented and will be interpreted as Bypass instructions.

EXTEST The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the processor's Boundary Scan Register out on the output pins corresponding to each boundary scan cell and capturing the values on the processor input pins to be loaded into their corresponding Boundary Scan Register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the Boundary Scan Register. Values shifted into input latches in the Boundary Scan Register are never used by the internal logic of the processor. Note: after using the EXTEST instruction, the processor must be reset before normal (non-boundary scan) use.

SAMPLE/PRELOAD The SAMPLE/PRELOAD performs two functions. When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a "snap-shot" of the normal operation of the component without interfering with that normal operation. The instruction causes Boundary Scan Register cells associated with outputs to sample the value being driven by the processor. It causes the cells associated with inputs to sample the value being driven into the processor. On both outputs and inputs the sampling occurs on the rising edge of TCK. When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the Boundary Scan Register on the falling edge of TCK.

IDCODE The IDCODE instruction selects the device identification register to be connected to TDI and TDO. This allows the device identification code to be shifted out of the device on TDO.

RUNBIST The RUNBIST instruction selects the one (1) bit Runbist Register, loads a value of “1” into the Runbist Register, and connects it to TDO. It also initiates the built-in self test (BIST) feature of the embedded Pentium processor. After loading the RUNBIST instruction code in the instruction register, the TAP controller must be placed in the Run-Test/Idle state. BIST begins on the first rising edge of TCK after entering the Run-Test/Idle state. The TAP controller must remain in the Run-Test/Idle state until BIST is completed. It requires 2^{19} core clock cycles to complete BIST and report the result to the Runbist Register. After completing BIST, the value in the Runbist Register should be shifted out on TDO during the Shift-DR state. A value of “0” being shifted out on TDO indicates BIST successfully completed. A value of “1” indicates a failure occurred. The CLK clock must be running in order to execute RUNBIST. After executing the RUNBIST instruction, the processor must be reset prior to normal (non-boundary scan) operation.

HI-Z The TAP Hi-Z instruction causes all outputs and I/Os of the embedded Pentium processor to go to a high-impedance state (float) immediately. The Hi-Z state is terminated by either resetting the TAP with the TRST# pin, by issuing another TAP instruction, or by entering the Test_Logic_Reset state. The Hi-Z state is enabled or disabled on the first TCK clock after the TAP instruction has entered the UPDATE-IR state of the TAP control state machine. This instruction overrides all other bus cycles. Resetting the processor will not disable this instruction since CPU RESET does not reset the TAP.

BYPASS The BYPASS instruction selects the Bypass Register to be connected to TDI and TDO. This effectively bypasses the test logic on the processor by reducing the shift length of the device to one bit. Note that an open circuit fault in the board level test data path will cause the Bypass Register to be selected following an instruction scan cycle due to a pull-up resistor on the TDI input. This was implemented to prevent any unwanted interference with the proper operation of the system logic.

Error Detection 10

The embedded Pentium processor incorporates a number of data integrity features that are focused on the detection and limited recovery of errors. The data integrity features provide capabilities for error detection of the internal devices and the external interface. The processor also provides the capability to obtain maximum levels of error detection by incorporating Functional Redundancy Checking (FRC) support. Error detecting circuits in the embedded Pentium processor do not limit the operating frequency of the chip.

The data integrity features can be categorized as (1) internal error detection, (2) error detection at the bus interface, and (3) FRC support.

10.1 Internal Error Detection

Detection of errors of a majority of the devices in the processor is accomplished by employing parity checking in the large memory arrays of the chip. The data and instruction caches (both storage and tag arrays), translation lookaside buffers, and microcode ROM are all parity protected. The following describes the parity checking employed in the major memory arrays in the processor (MESI status bits are not parity protected):

- Parity bit per byte in the data cache storage array.
- Parity bit per entry in the data cache tag array.
- Four Parity bits: One for each of the even upper, even lower, odd upper, odd lower bits of an instruction cache line.
- Parity bit per entry in the instruction cache tag array.
- Parity bit per entry in both the data and instruction TLBs storage arrays.
- Parity bit per entry in both the data and instruction TLBs tag arrays.
- Parity bit per entry in the microcode ROM.

Parity checking as described above provides error detection coverage of 53% of the on-chip devices. This error detection coverage number also includes the devices in the branch target buffer since branch predictions are always verified.

If a parity error has occurred internally, processor operation can no longer be trusted. Normally, a parity error on a read from an internal array will cause the processor to assert the IERR# pin and then shutdown. (Shutdown will be entered assuming it is not prevented from doing so by the error.); however, if TR1.NS is set, IERR# will not result in processor shutdown. Execution will continue, but operation will not be reliable. Parity errors on reads during normal instruction execution, reads during a flush operation, reads during BIST and testability cycles, and reads during inquire cycles will cause IERR# to be asserted. The IERR# pin will be asserted for one clock for each clock a parity error is detected and may be latched by the system. The IERR# pin is a glitch free signal, so no spurious assertions of IERR# will occur.

In general, internal timing constraints of the processor do not allow the inhibition of writeback cycles caused by inquire cycles, FLUSH# assertion or the WBINVD instruction when a parity error is encountered. In those cases where an internal parity error occurred during the generation of a writeback cycle, and that cycle was not able to be inhibited, the IERR# pin can be used to

recognize that the writeback should be ignored. If an internal parity error occurs during a flush operation, the processor will assert the IERR# pin as stated above, and the internal caches will be left in a partially flushed state. The flush, flush acknowledge, or writeback special cycles will not be run.

10.2 Error Detection at the Processor Interface

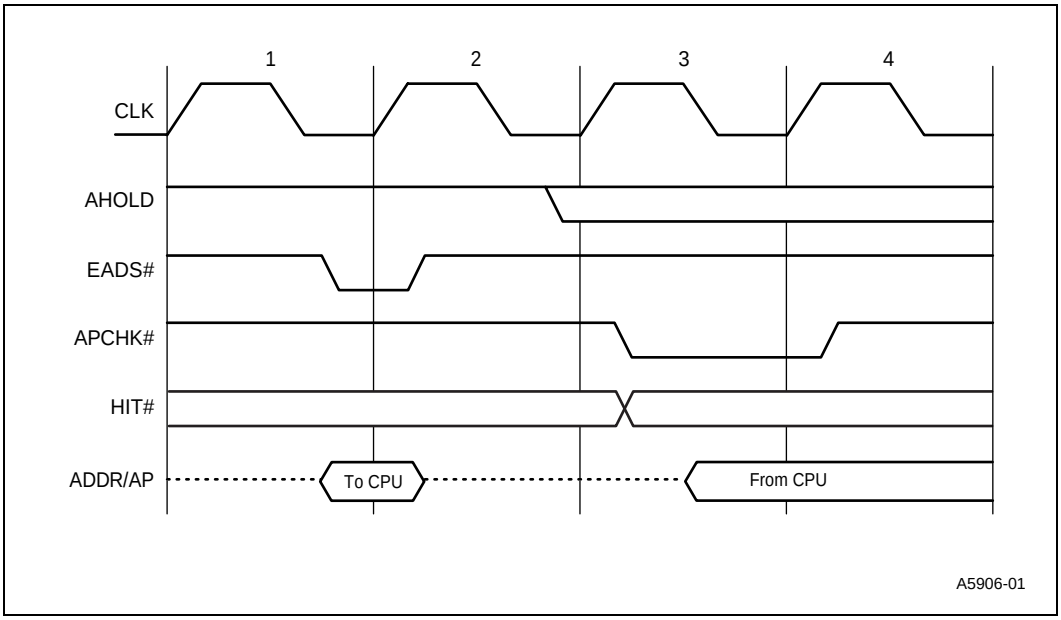
The processor provides parity checking on the external address and data buses. There is one parity bit for each byte of the data bus and one parity bit for bits A31–A5 of the address bus.

10.2.1 Address Parity

A separate and independent mechanism is used for parity checking on the address bus during inquire cycles. Even address parity is driven along with the address bus during all processor initiated bus cycles and checked during inquire cycles. When the processor is driving the address bus, even parity is driven on the AP pin. When the address bus is being driven into the processor during an inquire cycle, this pin is sampled in any clock in which EADS# is sampled asserted. APCHK# is driven with the parity status two clocks after EADS# is sampled active. The APCHK# output (when active) indicates that a parity error has occurred on the address bus during an inquire. Figure 10-1 depicts an address parity error during an inquire cycle. For additional timing diagrams which show address parity, see Chapter 6, “Bus Functional Description.” The APCHK# pin will be asserted for one clock for each clock a parity error is detected and may be latched by the system. The APCHK# pin is a glitch free signal, so no spurious assertions of APCHK# will occur.

In the event of an address parity error during inquire cycles, the internal snoop will not be inhibited. If the inquire hits a modified line in this situation and an active AHOLD prevents the processor from driving the address bus, the processor will potentially writeback a line at an address other than the one intended. If the processor is not driving the address bus during the writeback cycle, it is possible that memory will be corrupted.

Figure 10-1. Inquire Cycle Address Parity Checking

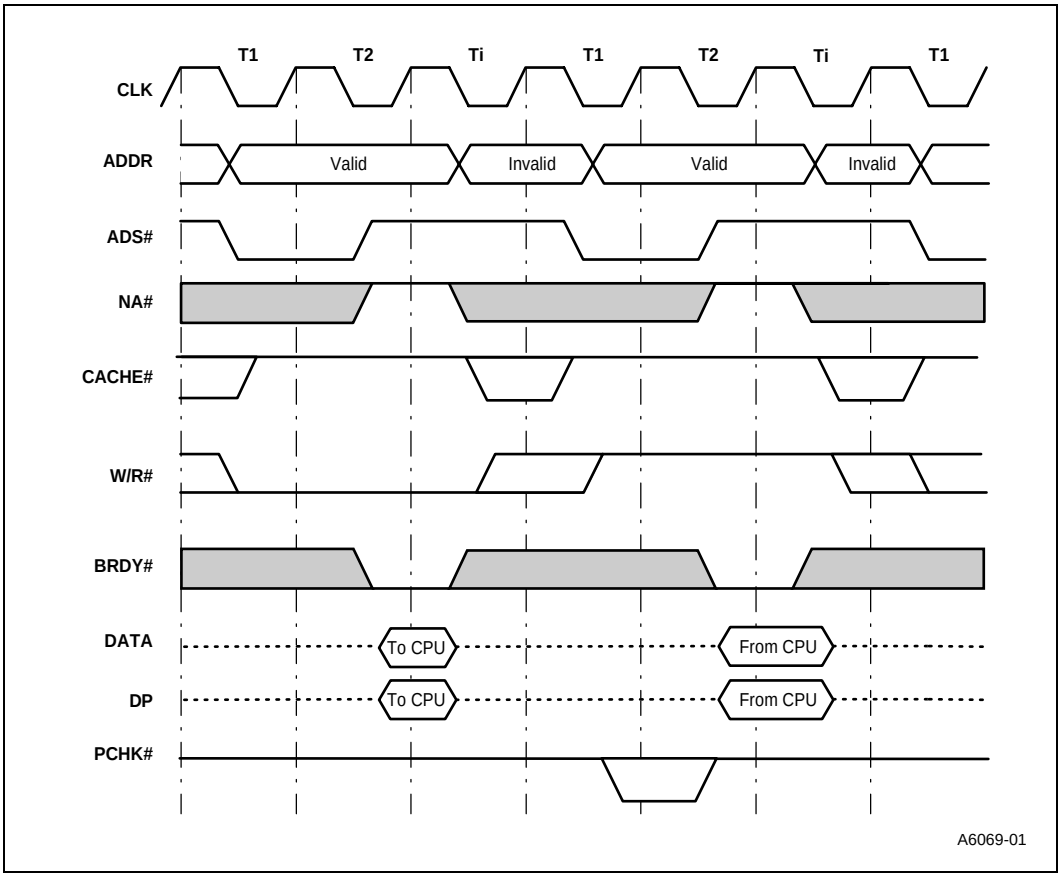


Driving APCHK# is the only effect that bad address parity has on the processor. It is the responsibility of the system to take appropriate action if a parity error occurs. If parity checks are not implemented in the system, the APCHK# pin may be ignored.

10.2.2 Data Parity

Even data parity is driven on the DP7–DP0 pins in the same clock as the data bus is driven during all processor initiated data write cycles. During reads, even parity information may be driven back to the processor on the data parity pins along with the data being returned. Parity status for data sampled is driven on the PCHK# pin two clocks after the data is returned. PCHK# is driven low if a data parity error was detected, otherwise it is driven high. The PCHK# pin will be asserted for one clock for each clock a parity error is detected and may be latched by the system. The PCHK# pin is a glitch free signal, so no spurious assertions of PCHK# will occur. Figure 10-2 shows when the data parity (DP) pins are driven/sampled and when the PCHK# pin is driven. For additional timing diagrams that show data parity, see Chapter 6, “Bus Functional Description.”

Figure 10-2. Data Parity During a Read and Write Cycle



Driving PCHK# is the only effect that bad data parity has on the processor. It is the responsibility of the system to take appropriate action if a parity error occurs. If parity checks are not implemented in the system, the PCHK# pin may be ignored.

10.2.2.1 Machine Check Exception as a Result of a Data Parity Error

The PEN# input determines whether a machine check interrupt will be taken as a result of a data parity error. If a data parity error occurs on a read for which PEN# was asserted, the physical address and cycle information of the cycle causing the parity error will be saved in the Machine Check Address Register and the Machine Check Type Register. If in addition, the CR4.MCE is set to 1, the machine check exception is taken. See “Machine Check Exception” on page 10-4 for more information.

The parity check pin, PCHK#, is driven as a result of read cycles regardless of the state of the PEN# input.

10.2.3 Machine Check Exception

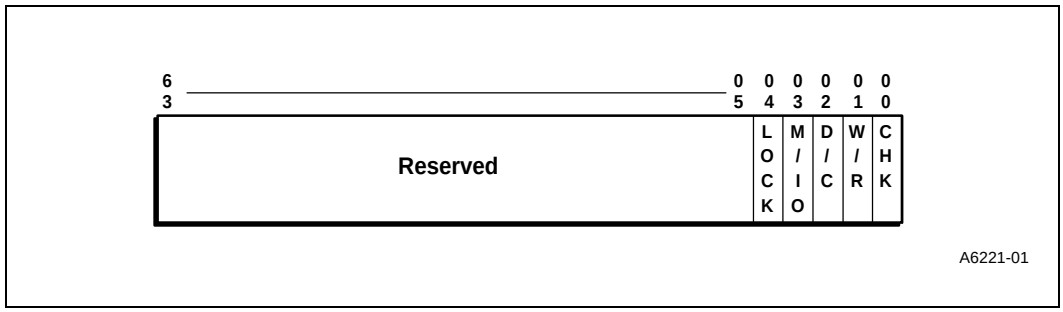
As mentioned in the earlier section, a new exception has been added to the processor. This is the machine check exception which resides at interrupt vector 18 (decimal). In processors previous to the Pentium processor, interrupt vector 18 was reserved and, therefore, there should be no interrupt routine located at vector 18. For compatibility, the MCE bit of the CR4 register will act as the machine check enable bit. When set to “1,” this bit will enable the generation of the machine check exception. When reset to “0,” the processor will inhibit generation of the machine check exception. CR4.MCE will be cleared on processor reset. In the event that a system is using the machine check interrupt vector for another purpose and the Machine Check Exception is enabled, the interrupt routine at vector 18 must examine the state of the CHK bit in the Machine Check Type register to determine the cause of its activation. Note that at the time the system software sets CR4.MCE to 1, it must read the Machine Check Type register in order to clear the CHK bit.

The Machine Check Exception is an abort; that is, it is not possible to reliably restart the instruction stream or identify the instruction causing the exception. Therefore, the exception does not allow the restart of the program that caused the exception. The processor does not generate an error code for this exception. Since the machine check exception is synchronous to a bus cycle and not an instruction, the IP pushed on to the stack may not be pointing to the instruction which caused the failing bus cycle.

The Machine Check Exception can be caused by one of two events: 1) Detection of data parity error during a read when the PEN# input is active, or 2) The BUSHCK# input being sampled active. When either of these events occur, the cycle address and type will be latched into the Machine Check Address (MCA) and Machine Check Type (MCT) registers (independent of the state of the CR4.MCE bit). If in addition, the CR4.MCE is “1,” a machine check exception will occur. When the MCA and MCT registers are latched, the MCT.CHK bit is set to “1” indicating that their contents are valid (Figure 10-3).

The Machine Check Address register, and the Machine Check Type register are model specific, read only registers. The Machine Check Address register is a 64-bit register containing the physical address for the cycle causing the error. The Machine Check Type register is a 64-bit register containing the cycle specification information, as defined in Figure 10-3. These registers are accessed using the RDMSR instruction. When the MCT.CHK is zero, the contents of the MCT and MCA registers are undefined. When the MCT register is read (using the RDMSR instruction), the CHK bit is reset to zero. Therefore, software must read the MCA register before reading the MCT register.

Figure 10-3. Machine Check Type Register



The bits in the Machine Check Type Register are defined as follows:

CHK: This bit is set to 1 when the Machine Check Type register is latched and is reset to 0 after the Machine Check Type register is read via the RDMSR instruction. In the event that the Machine Check Type register is latched in the same clock in which it is read, the CHK bit will be set. The CHK bit is reset to "0" on assertion of RESET. When the CHK bit is "0," the contents of the MCT and MCA registers are undefined.

M/IO#, D/C#, W/R#: These cycle definition pins can be decoded to determine if the cycle in error was a memory or I/O cycle, a data or code fetch, and a read or a write cycle. (See the embedded Pentium processor datasheets for detailed pin definitions.)

LOCK: Set to "1" if LOCK# is asserted for the cycle

10.2.4 Bus Error

The BUSCHK# input provides the system a means to signal an unsuccessful completion of a bus cycle. This signal is sampled on any edge in which BRDY# is sampled, for reads and writes. If this signal is sampled active, then the cycle address and type will be latched into the Machine Check Address and Machine Check Type registers. If in addition, the CR4.MCE bit is set to 1, the processor will be vectored to the machine check exception.

Even if BUSCHK# is asserted in the middle of a cycle, BRDY# must be asserted the appropriate number of clocks required to complete the bus cycle. The purpose of BUSCHK# is to act as an indication of an error that is synchronous to bus cycles. If the machine check interrupt is not enabled, i.e., the MCE bit in the CR4 register is zero, then an assertion of BUSCHK# will not cause the processor to vector to the machine check exception.

The embedded Pentium processor can remember only one machine check exception at a time. This exception is recognized on an instruction boundary. If BUSCHK# is sampled active while servicing the machine check exception for a previous BUSCHK#, it will be remembered by the processor until the original machine check exception is completed. It is then that the processor will service the machine check exception for the second BUSCHK#. Note that only one BUSCHK# will be remembered by the processor while the machine exception for the previous one is being serviced.

When the BUSCHK# is sampled active by the processor, the cycle address and cycle type information for the failing bus cycle is latched upon assertion of the last BRDY# of the bus cycle. The information is latched into the Machine Check Address and Machine Check Type registers respectively. However, if the BUSCHK# input is not deasserted before the first BRDY# of the next

bus cycle, and the machine check exception for the first bus cycle has not occurred, then new information will be latched into the MCA and MCT registers, over-writing the previous information at the completion of this new bus cycle. Therefore, in order for the MCA and MCT registers to report the correct information for the failing bus cycle when the machine check exception for this cycle is taken at the next instruction boundary, the system must deassert the BUSCHK# input immediately after the completion of the failing bus cycle and before the first BRDY# of the next bus cycle is returned.

10.2.5 Functional Redundancy Checking

Functional Redundancy Checking (FRC) in the embedded Pentium processor will provide maximum error detection (>99%) of on-chip devices and the processor's interface. A "checker" processor that executes in lock step with the "master" processor is used to compare output signals every clock.

Note: The embedded Pentium processor with MMX technology does not support FRC. Also, FRC is not supported in Dual processor designs.

Two embedded Pentium processors are required to support FRC. Both the master and checker must be of the same stepping and same bus fraction. The processor configured as a master operates according to bus protocol described in this document. The outputs of the checker processor are three-stated (except IERR#, TDO, PICD0, PICD1—however, these signals are not part of FRC) so the outputs of the master can be sampled. If the sampled value differs from the value computed internally by the checker, the checker asserts the IERR# output to indicate an error. A master-checker pair should have all pins except FRCMC#, IERR#, PICD0, PICD1 and TDO tied together.

The processors are configured either as a master or a checker by driving the FRCMC# input to the appropriate level while RESET is asserted. If sampled low during reset, the processor enters checker mode and three-states all outputs except IERR# and TDO (IERR# is driven inactive during reset). This feature is provided to prevent bus contention before reset is completed. The final master/checker configuration is determined when RESET transitions from high to low. The final master/checker configuration may not be changed other than by a subsequent RESET.

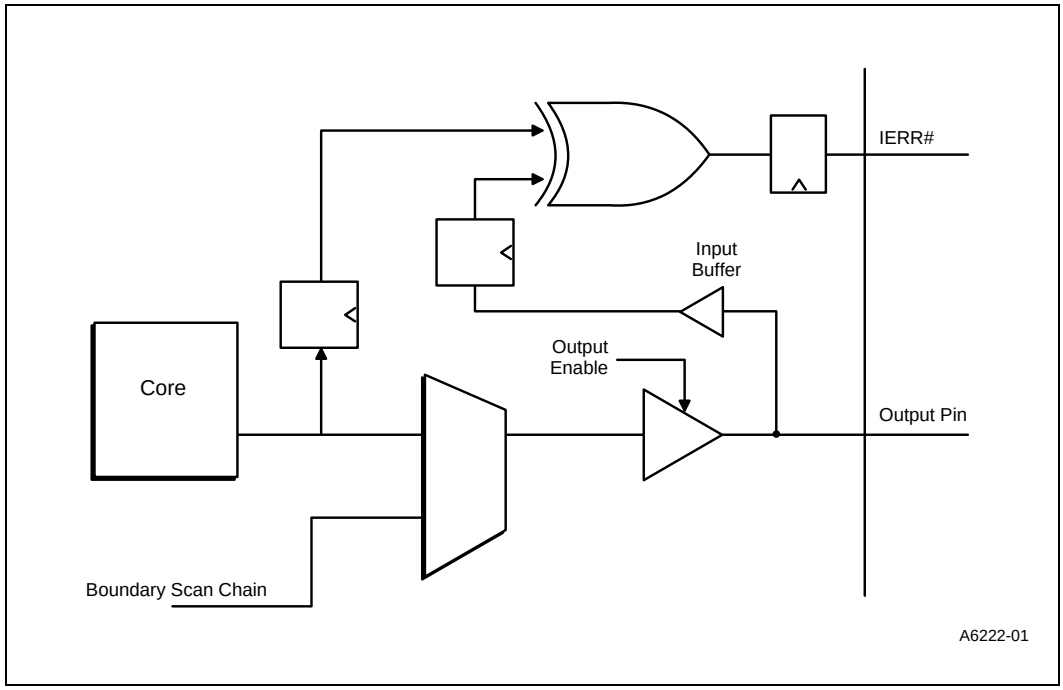
The IERR# pin reflects the result of the master-checker comparison. It is asserted for one clock, two clocks after the mismatch. It is asserted for each detected mismatch, so IERR# may be low for more than one consecutive clock. During the assertion of RESET, IERR# will be driven inactive. After RESET is deasserted, IERR# will not be asserted due to a mismatch until two clocks after the ADS# of the first bus cycle (i.e., in the third clock of the first bus cycle). IERR# will reflect pin comparisons thereafter. Note that IERR# may be asserted due to an internal parity error prior to the first bus cycle. It is possible for FRC mismatches to occur in the event that an undefined processor state is driven off-chip, therefore no processor state should be stored without having been previously initialized.

In order for the master-checker pair to operate correctly, the system must be designed such that the master and the checker sample identical input states in the same clock. All asynchronous inputs should change state in such a manner that both the master and checker sample them in the same state in the same clock. The simplest way to do this is to design all asynchronous inputs to be synchronously controlled.

The TDO pin is not tested by FRC since it operates on a separate clock. Note that it is possible to use boundary scan to verify the connection between the master and checker by scanning into one, latching the outputs of the other and then scanning out. The Stop Clock state feature cannot be used in dual processing or functional redundancy checking modes because there is no way to re-synchronize the internal clocks of the two processors.

Figure 10-4 illustrates the configuration of output pins with respect to FRC. The comparators at each output compare the value of the package pin with the value being driven from the core to that pin, not the value driven by boundary scan to that pin. Therefore, during the use of boundary scan, FRC mismatches (IERR# assertion) can be expected to occur.

Figure 10-4. Conceptual IERR# Implementation for FRC



Execution Tracing 11

The embedded Pentium® processor family uses special bus cycles to support execution tracing. These bus cycles, which are optional, have a significant impact on overall performance. Execution tracing allows the external hardware to track the flow of instructions as they execute inside the processor.

The special bus cycles generated by the processor are Branch Trace Messages (BTM). Due to physical limitations, the maximum number of outstanding taken branches allowed is two. Once the second taken branch reaches the last stage of the pipeline, execution is stalled until the first branch message is sent on the bus.

Branch trace messages may be enabled by setting the Execution Tracing bit, TR, of TR12 (bit 1) to a 1. Once enabled, there are two forms of branch trace messages: normal and fast. Normal messages produce two cycles, one for the branch target linear address, and one for the linear address of the instruction causing the taken branch. Fast messages only produce the second of these two cycles. The second message will always contain the linear address of the instruction executed in the u pipe even if the instruction that caused the branch was executed in the v pipe. For serializing instructions and segment descriptor loads the address field of the first cycle will contain the address of the next sequential instruction after the instruction that caused the BTM. Fast execution tracing is enabled by setting bit 8 of TR12 to 1. Note that switching between the normal and fast formats by using the WRMSR instruction to change bit 8 of TR12, the WRMSR instruction causes a branch trace message when they are enabled. The format for this branch trace message will be the format that was programmed *before* the WRMSR instruction was executed.

Normal and fast branch trace messages may be delayed by 0 or more clocks after the cycle in which the branch was taken depending on the bus activity. Also, higher priority cycles may be run between the first and second cycles of a normal branch trace message. In dual-processor mode, branch trace message cycles may be interleaved with cycles from the other processor. Branch trace message cycles are buffered so they do not normally stall the processor.

Branch trace messages, normal and fast, may be identified by the following special cycle:

M/IO#	= 0
D/C#	= 0
W/R#	= 1
BE7#–BE0#	= 0DFH

The address and data bus fields for the two bus cycles associated with a branch trace message are defined below:

First Cycle (Normal)

A31–A4	Bits 31 – 4 of the branch target linear address
A3	“1” if the default operand size is 32 bits “0” if the default operand size is 16 bits
D63–D60	Bits 3 – 0 of the branch target linear address
D59	“0” - indicating the first of the two cycles
D58–D0	Reserved. Driven to a valid state, but must be ignored

Second Cycle (Normal)

A31–A4	Bits 31 – 4 of the linear address of the instruction causing the taken branch
A3	“1” if the default operand size is 32 bits
	“0” if the default operand size is 16 bits
D63–D60	Bits 3 – 0 of the linear address of the instruction causing the taken branch
D59	“1” - indicating the second of the two cycles
D58–D0	Reserved. Driven to a valid state, but must be ignored

Fast Cycle

A31–A4	Bits 31 – 4 of the linear address of the instruction causing the taken branch
A3	“1” if the default operand size is 32 bits
	“0” if the default operand size is 16 bits
D63–D60	Bits 3 – 0 of the linear address of the instruction causing the taken branch
D59	Driven to a “1”
D58–D0	Reserved. Driven to a valid state, but must be ignored

In addition to conditional branches, jumps, calls, returns, software interrupts, and interrupt returns, the processor treats the following operations as causing taken branches:

- Serializing instructions
- Some segment descriptor loads
- Hardware interrupts
- Certain floating-point exceptions (both masked and unmasked) and all other exceptions that invoke a trap or fault handler
- Exiting the HALT state

With execution tracing enabled, these operations will also cause a corresponding branch trace message cycle. The processor data bus is valid during branch trace message special cycles. Instructions which cause masked floating point exceptions may cause one or more branch trace special cycles. This is because execution of an instruction may be aborted and restarted several times due to the exception.

Also note that the WRMSR instruction to enable branch trace messages will cause a BTM to be generated (WRMSR is a serializing instruction and serializing instructions cause BTMs). A WRMSR to disable BTMs will not generate a BTM. Conditions which cause the VERR, VERW, LAR and LSL instruction to clear the ZF bit in EFLAGS will also cause these instructions to be treated as taken branches. However, if these instructions fail the protection checks, no branch trace message will be generated.

Note that if an instruction faults, it does not complete execution but instead is flushed from the pipeline and an exception handler is invoked. This faulting instruction effectively causes a branch; a branch trace message is generated accordingly.

Power Management 12

The embedded Pentium® processor family implements Intel's System Management Mode (SMM) architecture. This chapter describes the hardware interface to SMM and Clock Control.

12.1 Power Management Features

- System Management Interrupt can be delivered through the SMI# signal or through the local APIC using the SMI# message, which enhances the SMI interface, and provides for SMI delivery in APIC-based Pentium processor dual processing systems.
- In dual processing systems, SMIACT# from the bus master (MRM) behaves differently than in uniprocessor systems. If the LRM processor is the processor in SMM mode, SMIACT# will be inactive and remain so until that processor becomes the MRM.
- The Pentium processor is capable of supporting an SMM I/O instruction restart. This feature is automatically disabled following RESET. To enable the I/O instruction restart feature, set bit 9 of the TR12 register to "1".
- The Pentium processor default SMM revision identifier has a value of 2 when the SMM I/O instruction restart feature is enabled.
- SMI# is NOT recognized by the processor in the shutdown state.

12.2 System Management Interrupt Processing

The system interrupts the normal program execution and invokes SMM by generating a System Management Interrupt (SMI#) to the processor. The processor will service the SMI# by executing the following sequence. See Figure 12-1.

1. Wait for all pending bus cycles to complete and EWBE# to go active.
2. The processor asserts the SMIACT# signal while in SMM indicating to the system that it should enable the SMRAM.
3. The processor saves its state (context) to SMRAM, starting at address location SMBASE + 0FFFFH, proceeding downward in a stack-like fashion.
4. The processor switches to the System Management Mode processor environment (a pseudo-real mode).
5. The processor will then jump to the absolute address of SMBASE + 8000H in SMRAM to execute the SMI handler. This SMI handler performs the system management activities.
6. The SMI handler will then execute the RSM instruction which restores the processor's context from SMRAM, deasserts the SMIACT# signal, and then returns control to the previously interrupted program execution.

Note: The default SMBASE value following RESET is 30000H.

Figure 12-1. Basic SMI# Interrupt Service

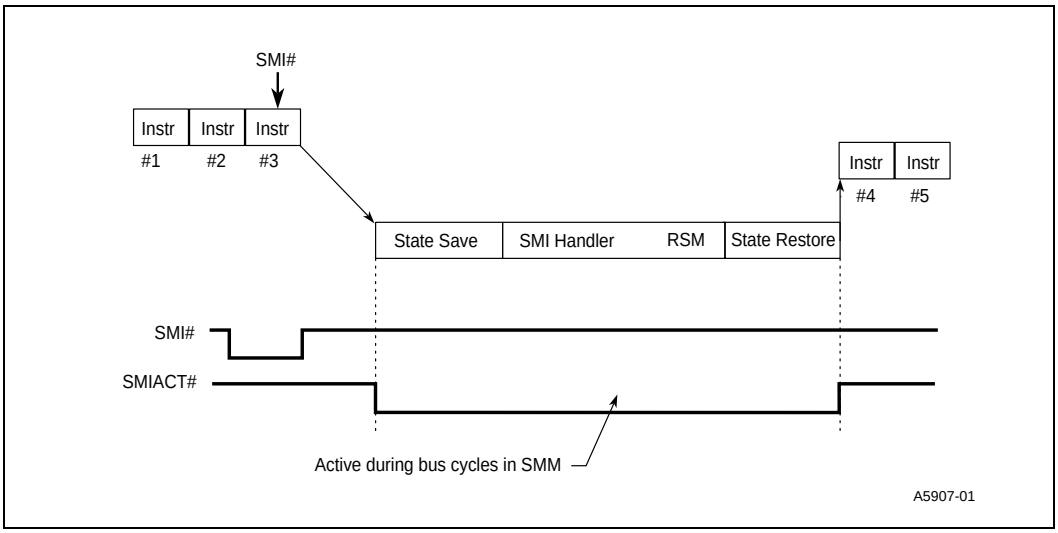
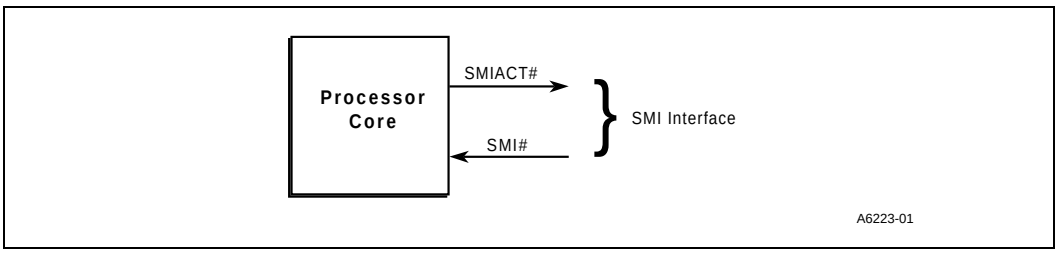


Figure 12-2 describes the System Management Interrupt hardware interface which consists of the SMI# interrupt request input and the SMIACT# output used by the system to decode the SMRAM.

Figure 12-2. Basic SMI# Hardware Interface



12.2.1 System Management Interrupt (SMI#)

SMI# is a falling-edge triggered, non-maskable interrupt request signal. SMI# is an asynchronous signal, but setup and hold times, t28 and t29, must be met in order to guarantee recognition on a specific clock. The SMI# input need not remain active until the interrupt is actually serviced. The SMI# input only needs to remain active for a single clock if the required setup and hold times are met. SMI# will also work correctly if it is held active for an arbitrary number of clocks.

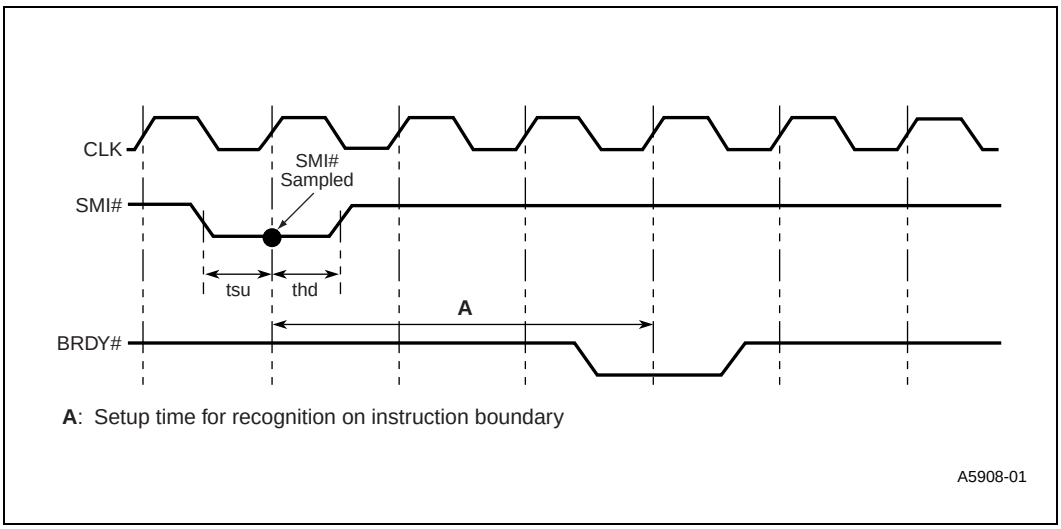
The SMI# signal is synchronized internally and must be asserted at least three CLK periods prior to asserting the BRDY# signal in order to guarantee recognition on a specific instruction boundary. See Figure 12-3.

The SMI# input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent SMI# might not be recognized if the SMI# input is not held inactive for at least four clocks after being asserted.

SMI#, like NMI, is not affected by the IF bit in the EFLAGS register and is recognized on an instruction boundary. An SMI# will not break locked bus cycles. The SMI# has a higher priority than NMI and is not masked during an NMI.

After the SMI# interrupt is recognized, the SMI# signal will be masked internally until the RSM instruction is executed and the interrupt service routine is complete. Masking the SMI# prevents recursive SMI# calls. If another SMI# occurs while the SMI# is masked, the pending SMI# will be recognized and executed on the next instruction boundary after the current SMI# completes. This instruction boundary occurs before execution of the next instruction in the interrupted application code, resulting in back to back SMM handlers. Only one SMI# can be pending while SMI# is masked.

Figure 12-3. SMI# Timing



12.2.1.1 SMI# Synchronization for I/O Instruction Restart

The SMI# signal is synchronized internally and must be asserted at least three CLK periods prior to asserting the BRDY# signal in order to guarantee recognition on a specific I/O instruction boundary. This is important for servicing an I/O trap with an SMI# handler. Due to the asynchronous nature of SMI# delivery with the APIC, it is impossible to synchronize the assertion of BRDY#. As a result, the SMM I/O instruction restart feature cannot be used when an SMI is delivered via the local APIC.

12.2.1.2 Dual Processing Considerations For SMI# Delivery

Although the SMM functions the same when the dual processor is inserted into Socket 7, the dual processor operation of the system must be carefully considered. Table 12-1 shows the four possible options for SMI# delivery depending on the SMM applications (mainly power management) the system has to support. There are implications to system design and the SMM handler. Note that for operation with the Dual processor and upgradability with a future upgrade processor, Option #3 is strongly recommended.

Table 12-1. Dual Processing SMI# Delivery Options

	SMI# Pins Tied Together	SMI# Pins NOT Tied Together
SMI# pins delivering SMI	Option #1 Both processors enter SMM.	Option #2 One processor enters SMM.
APIC delivering SMI	Option #3 One or Both processors enter SMM.	Option #4 One or Both processors enter SMM

Note: The I/O Instruction Restart Power Management feature should not be used when delivering the system management interrupt via the local APIC. Refer to the *Intel Architecture Software Developer's Manual, Volume 3* for additional details on I/O instruction restart.

Implications

1. SMI# pin delivery of SMI with the SMI# pins tied together: Any assertion of the SMI# pin will cause both the Primary and Dual processors to interrupt normal processing, enter SMM mode and start executing SMM code in their respective SMRAM spaces. In this case, using the I/O Instruction restart feature in Dual Processor mode will require additional system hardware (D/P# pin) and software (detection of which processor was the MRM when the SMI# pin was asserted) considerations.
2. SMI# pin delivery of SMI with the SMI# pins NOT tied together: Only the processor whose SMI# pin is asserted will handle SMM processing. It is possible that both the Primary and Dual processor will be doing SMM processing at the same time, especially if the I/O Instruction restart feature is being used. If I/O instruction restart is not supported, then it is possible to dedicate only one processor for SMM handling at any time.
3. APIC SMI# delivery of SMI with the SMI# pins tied together: This option is strongly recommended for operation with the Dual processor and upgradability with the Pentium OverDrive® processor. System Management Interrupts should be delivered via the APIC for DP systems, and may be delivered either via the APIC or the SMI# pin for turbo-upgraded systems. Either the Primary or Dual processor can be the assigned target for SMI# delivery and hence SMM handling. The SMM I/O instruction restart feature may be used in a uniprocessor system or in a system with a (with SMI# pin delivery of the interrupt), but the system must not use this feature when operating in dual processing mode (with APIC delivery of the interrupt).
4. APIC SMI# delivery of SMI with the SMI# pins NOT tied together: I/O Instruction Restart feature is not recommended when delivering SMI via the local APIC. Either the Primary or Dual processor can be the assigned target for SMI# delivery and hence SMM handling.

12.2.2 System Management Interrupt Via APIC

When SMI# is asserted (SMI# pin asserted low or APIC SMI# message) it causes the processor to invoke SMM.

12.2.3 SMI Active (SMIACT#)

SMIACT# indicates that the processor is operating in System Management Mode. The processor asserts SMIACT# in response to an SMI interrupt request on the SMI# pin or through the APIC message. SMIACT# is driven active for accesses only after the processor has completed all pending write cycles (including emptying the write buffers — EWBE# returned active by the system). SMIACT# will be asserted for all accesses in SMM beginning with the first access to SMRAM when the processor saves (writes) its state (or context) to SMRAM. SMIACT# is driven active for every access until the last access to SMRAM when the processor restores (reads) its state from SMRAM. The SMIACT# signal is used by the system logic to decode SMRAM.

The number of CLKs required to complete the SMM state save and restore is very dependent on system memory performance and the processor bus frequency.

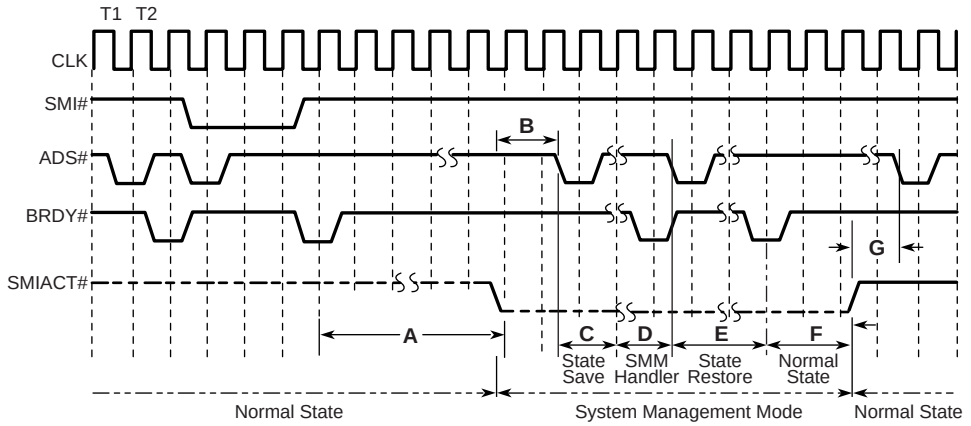
As shown in Figure 12-4, the approximate time required to enter an SMI handler routine for the Pentium processor (from the completion of the interrupted instruction) is given by:

Latency to beginning of SMI handler = A + B + C = ~184 CLKs

The approximate time required to return to the interrupted application (following the final SMM instruction before RSM) is given by:

Latency to continue interrupted application = E + F + G = ~207 CLKs

Figure 12-4. SMIACT# Timing



A: Last BRDY# from non-SMM transfer to SMIACT# assertion:	2 CLKs minimum
B: SMIACT# assertion to first ADS# for SMM state save:	2 CLKs minimum
C: SMM state save (dependent on memory performance):	Approximately 180 CLKs
D: SMM handler:	User determined
E: SMM state restore (dependent on memory performance):	Approximately 200 CLKs
F: Last RDY# from SMM transfer to deassertion of SMIACT#:	2 CLKs minimum
G: SMIACT# deassertion to first non-SMM ADS#:	5 CLKs minimum

A5909-01

12.2.3.1 Dual Processing Considerations for SMIACT#

When the processor is the only processor present, then it always drives the D/P# signal low. SMIACT# is asserted when the processor enters SMM and is deasserted only when the processor exits SMM.

When the Dual processor is also present, the D/P# signal toggles depending upon whether the Primary or Dual processor owns the bus (MRM). The SMIACT# pins may be tied together or be used separately to ensure SMRAM access by the correct processor.

Caution: If SMIACT# is used separately: the SMIACT# signal is only driven by the Primary or Dual processor when it is the MRM, so this signal must be qualified with the D/P# signal.

In a dual socket system, connecting the SMIACT# signals together on the Primary and Dual processor sockets is strongly recommended for dual processing operation.

In dual processing systems, SMIACT# may not remain low (e.g., may toggle) if both processors are not in SMM mode. he SMIACT# signal is asserted by either the Primary or Dual processor based on two conditions: the processor is in SMM mode and is the bus master (MRM). If one processor is executing in normal address space, the SMIACT# signal will go inactive when that processor is MRM. The LRM processor, even if in SMM mode, will not drive the SMIACT# signal low.

12.3 SMM — System Design Considerations

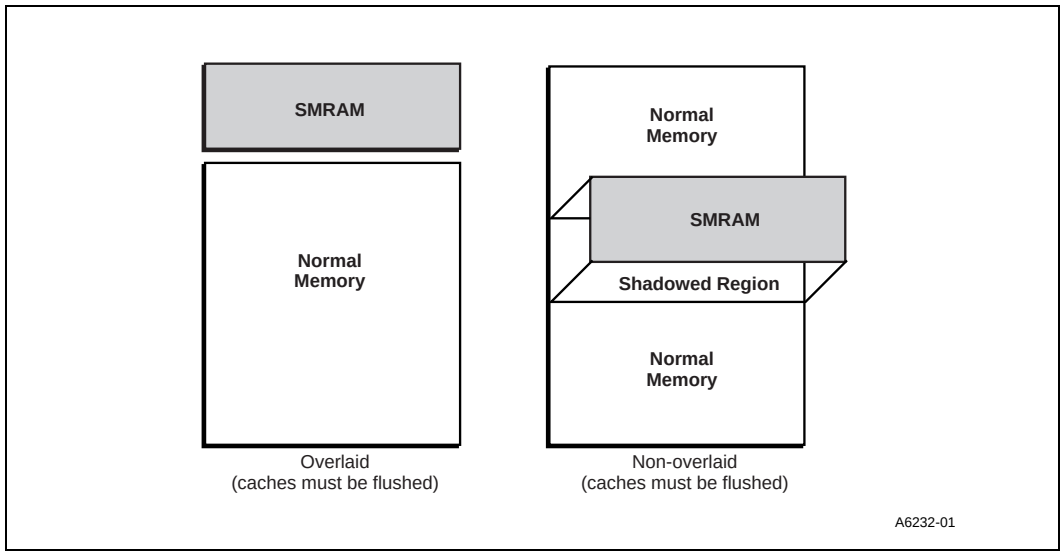
12.3.1 SMRAM Interface

The hardware designed to control the SMRAM space must follow these guidelines:

1. A provision should be made to allow for initialization of SMRAM space during system boot up. This initialization of SMRAM space must happen before the first occurrence of an SMI# interrupt. Initializing the SMRAM space must include installation of an SMM handler, and may include installation of related data structures necessary for particular SMM applications. The memory controller providing the interface to the SMRAM should provide a means for the initialization code to manually open the SMRAM space.
2. A minimum initial SMRAM address space of SMBASE + 8000H to SMBASE + 0FFFFH should be decoded by the memory controller.
3. Alternate bus masters (such as DMA controllers) should not be allowed to access SMRAM space. Only the processor, either through SMI or during initialization, should be allowed access to SMRAM.
4. In order to implement a zero-volt suspend function, the system must have access to all of normal system memory from within an SMM handler routine. If the SMRAM is going to overlay normal system memory, there must be a method of accessing any system memory that is located underneath SMRAM.
5. Inquire cycles are permitted during SMM, but it is the responsibility of the system to ensure that any snoop writeback completes to the correct memory space, irrespective of the state of the SMIACT# pin. Specifically, if SMM is overlaid, and SMM space is non cacheable, then any snoop writeback cycle occurring during SMM must complete to system memory, even though SMIACT# will remain active.

   If an inquire cycle occurs after assertion of SMI# to the processor, but before SMIACT# is returned, note that SMIACT# could be returned at any point during the snoop writeback cycle. Depending on the timing of SMI# and the inquire cycle, SMIACT# could change states during the writeback cycle. Again, it is the responsibility of the system, if it supports snooping during SMM, to ensure that the snoop writeback cycle completes to the correct memory space, irrespective of the state of the SMIACT# pin.
6. It should also be noted that upon entering SMM, the branch target buffer (BTB) is not flushed and thus it is possible to get a speculative prefetch to an address outside of SMRAM address space due to branch predictions based on code executed prior to entering SMM. If this occurs, the system must still return BRDY# for each code fetch cycle.

Figure 12-5. SMRAM Location



12.3.2 Cache Flushes

The processor does not unconditionally writeback and invalidate its cache before entering SMM (this option is left to the system designer). If the SMRAM is in an area that is cacheable and overlaid on top of normal memory that is visible to the application or operating system (default), then it is necessary for the system to flush both the processor cache and any second level cache upon entering SMM. This may be accomplished by asserting flush the same time as the request to enter SMM (i.e., cache flushing during SMM entry is accomplished by asserting the FLUSH# pin at the same time as the request to enter SMM through SMI#). The priorities of FLUSH# and SMI# are such that the FLUSH# will be serviced first. To guarantee this behavior, the constraints on setup and hold timings on the interaction of FLUSH# and SMI# as specified for a processor should be obeyed. When the default SMRAM location is used, SMRAM is overlaid on top of system main memory (at SMBASE + 8000H to SMBASE + 0FFFFH).

In a system where FLUSH# and SMI# pins are synchronous and setup/hold times are met, then the FLUSH# and SMI# pins may be asserted in the same clock. In asynchronous systems, the FLUSH# pin must be asserted at least one clock before the SMI# pin to guarantee that the FLUSH# pin is serviced first. Note that in systems that use the FLUSH# pin to write back and invalidate the cache contents before entering SMM, the processor prefetches at least one cache line in between the time the Flush Acknowledge special cycle is run and the recognition of SMI# and the driving of SMIACT# for SMRAM accesses. It is the obligation of the system to ensure that these lines are not cached by returning KEN# inactive.

If SMRAM is located in its own distinct memory space, which can be completely decoded with only the processor address signals, it is said to be non-overlaid. In this case, there is one new requirement for maintaining cache coherency. Refer to Table 12-2.

Table 12-2. Scenarios for Cache Flushes with Writeback Caches

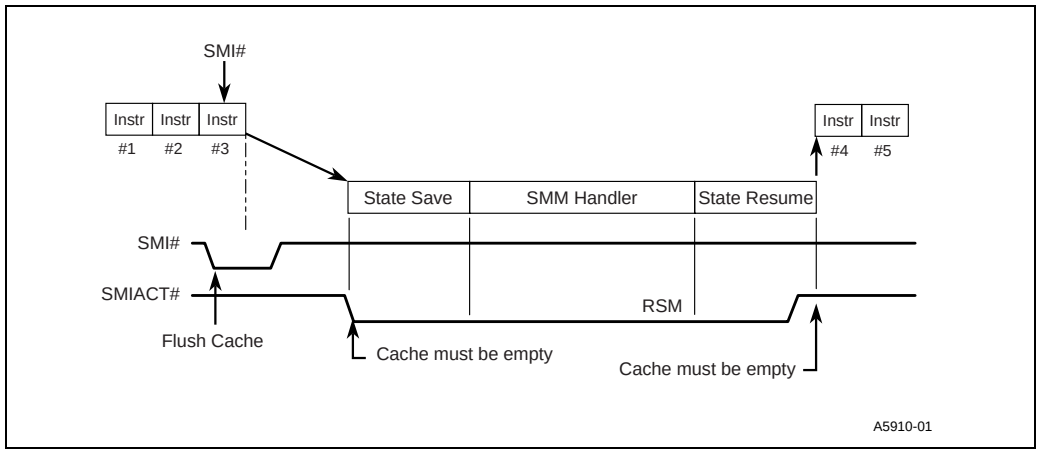
Is SMRAM overlapped with normal memory?	Is Normal Memory cacheable?	Is SMRAM cacheable?	Flush required during SMM entry?	Flush required during SMM exit?	Comments
No	No	No	No	No	
	No	WT	No	No	
	WT	No	No	No	
	WB	No	No†	No	†Snoop WBs must always go to normal memory space
	WT	WT	No	No	
	WB	WT	No†	No	†Snoop and Replacement WBs must go to normal memory space.
Yes	No	No	No	No	
	No	WT	No	Yes	
	WT	No	Yes	No	
	WB	No	Yes	No	
	WT	WT	Yes	Yes	
	WB	WT	Yes	Yes	

Note: Writeback cacheable SMRAM is not recommended. When flushing upon SMM exit, SMIACT# will be deasserted and may cause regular memory to be overwritten.

The processor implements writeback caches. Hence the performance hit due to flushing the cache for SMM execution can be more significant. Due to the writeback nature of the cache, flushing the cache has the following penalties:

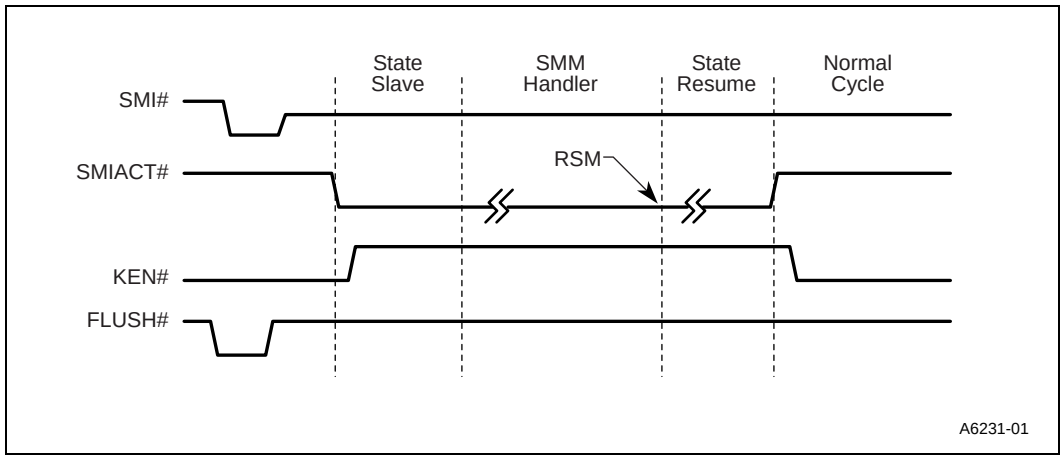
1. Before entry into SMM (when SMRAM is cacheable), the cache has to be flushed. Hence, all dirty lines need to be written back. This may cause a large number of bus cycles and increase SMM entry latency.
2. If the cache had to be flushed upon SMM exit, execution starts with cache miss 100%. The cache fill cycles reduce performance.

Figure 12-6. FLUSH# Mechanism During SMM with Overlay



The method suggested is shown in Figure 12-7.

Figure 12-7. Flush with Non-Cached SMM with Overlay



12.3.2.1 Dual Processing Considerations for Cache Flushes

Cache flushing during SMM exit is not possible while both the Primary and Dual processors are present due to the fact that it is not possible to clearly predict when the processor in SMM has exited. This is because the SMIACT# is not a static status indicator but only a bus cycle indicator for SMRAM accesses.

12.3.3 A20M# Signal

Systems based on the MS-DOS* operating system contain a feature that enables the processor address bit A20 to be forced to 0. This limits physical memory to a maximum of 1 Mbyte, and is provided to ensure compatibility with those programs that relied on the physical address wrap around functionality of the original IBM PC. The A20M# pin on the processor provides this function. When A20M# is active, all external bus cycles will drive A20 low, and all internal cache accesses will be performed with A20 low.

The A20M# pin is recognized while the processor is in SMM. The functionality of the A20M# input must be recognized in two instances:

1. If the SMM handler needs to access system memory space above 1 Mbyte (for example, when saving memory to disk for a zero-volt suspend), the A20M# pin must be deasserted before the memory above 1 Mbyte is addressed.
2. If SMRAM has been relocated to address space above 1 Mbyte, and A20M# is active upon entering SMM, the processor will attempt to access SMRAM at the relocated address, but with A20 low. This could cause the system to crash, since there would be no valid SMM interrupt handler at the accessed location.

In order to account for the above two situations, the system designer must ensure that A20M# is deasserted on entry to 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.

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:

- 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.)
- 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.

12.3.4 SMM and Second Level Write Buffers

Before the processor enters SMM, it empties its internal write buffers. This is necessary so that the data in the write buffers is written to normal memory space, not SMM space. Once the processor is ready to begin writing an SMM state save to SMRAM, it asserts the SMIACT# signal for SMRAM references. SMIACT# may be driven active by the processor before the system memory controller has had an opportunity to empty the second level write buffers.

To prevent the data from these second level write buffers from being written to the wrong location, the system memory controller needs to direct the memory write cycles to either SMM space or normal memory space. This can be accomplished by saving the status of SMIACT# along with the address for each word in the write buffers.

EWBE# can also be used to prevent the processor from asserting SMIACT# before write buffers are empty. The processor will wait for an active EWBE# before asserting SMIACT#.

12.4 Clock Control

12.4.1 Clock Generation

To understand the additional power management fears of the Pentium processor and how it manipulates the clock to conserve power, it is necessary to understand how the clock operates. The processor is capable of running internally at frequencies much higher than the bus speed via the various bus frequency settings. This allows simpler system design by lowering the clock speeds required in the external system. The high frequency internal clock relies on an internal Phase Lock Loop (PLL) to generate the two internal clock phases, “phase one” and “phase two.” Most external timing parameters are specified with respect to the rising edge of CLK. The PLL requires a constant frequency CLK input, and therefore the CLK input cannot be changed dynamically.

On the embedded Pentium processor, CLK provides the fundamental timing reference for the bus interface unit. The internal clock converter enhances all operations functioning out of the internal cache and/or operations not blocked by external bus accesses.

12.4.2 Stop Clock

The processor provides an interrupt mechanism, STPCLK#, that allows system hardware to control the power consumption of the processor by stopping the internal clock (output of the PLL) to the processor core in a controlled manner. This low-power state is called the Stop Grant state. The target for low-power mode supply current in the Stop Grant state is ~15% of normal I_{CC}.

When the processor recognizes a STPCLK# interrupt, the processor will stop execution on the next instruction boundary (unless superseded by a higher priority interrupt), stop the pre-fetch unit, complete all outstanding writes, generate a Stop Grant bus cycle, and then stop the internal clock. At this point, the processor is in the Stop Grant state.

Note: If STPCLK# is asserted during RESET and continues to be held active after RESET is deasserted, the processor will execute one instruction before the STPCLK# interrupt is recognized. Execution of instructions will therefore stop on the second instruction boundary after the falling edge of RESET.

The processor cannot respond to a STPCLK# request from a HLDA state because it cannot generate a Stop Grant cycle.

The rising edge of STPCLK# will tell the processor that it can return to program execution at the instruction following the interrupted instruction.

Unlike the normal interrupts, INTR and NMI, the STPCLK# interrupt does not initiate interrupt table reads. Among external interrupts, STPCLK# is the lowest priority.

12.4.2.1 STPCLK# Signal

STPCLK# is treated as a level triggered interrupt to the processor. This interrupt may be asserted asynchronously and is prioritized below all of the external interrupts. If asserted, the processor will recognize STPCLK# on the next instruction boundary, and then do the following:

1. Flush the instruction pipeline of any instructions waiting to be executed.
2. Wait for all pending bus cycles to complete and EWBE# to go active.

3. Drive a special bus cycle (Stop Grant bus cycle) to indicate that the clock is being stopped.
4. Enter low power mode.

STPCLK# is active low. To ensure STPCLK# recognition, the system must keep this signal active until the appropriate special cycle has been issued by the processor. To guarantee that every STPCLK# assertion, and subsequent deassertion and re-assertion, is recognized and thus will get a Stop Grant bus cycle response (which will also ensure that each deassertion of STPCLK# allows execution of at least one instruction), the system must meet the following requirements:

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

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

Note that if a system is not relying on either a Stop Grant bus cycle response for every STPCLK# assertion, or for each deassertion of STPCLK# to allow execution of at least one instruction, these detailed requirements can be ignored. Though STPCLK# is asynchronous, setup and hold times may be met to ensure recognition on a specific clock.

The STPCLK# input must be driven high (not floated) in order to exit the Stop Grant state. Once STPCLK# is deasserted and the processor resumes execution, the processor is guaranteed to execute at least one instruction before STPCLK# is recognized again. To return to normal state, external hardware must deassert STPCLK#.

12.4.2.2 Dual Processing Considerations

The Primary and Dual processors may or may not tie their STPCLK# signals together. The decision is dependent on system specific processor power conservation needs. Connecting the STPCLK# signals on the Primary and Dual processors together is strongly recommended for operation with the Dual processor.

Tying the STPCLK# signals together causes both the Primary and Dual processors to eventually enter the Stop Grant state on assertion of STPCLK#. The system ceases processing until the STPCLK# signal is deasserted. In Dual Processor mode with the STPCLK# pins tied together, independent STPCLK# control of each processor is not possible. Both the Primary processor and Dual processor will go into the Stop Grant state independently, and will each generate a Stop Grant special bus cycle.

Note: In a dual processing system where STPCLK# is tied to both the primary and dual processors, the system expects to see two Stop Grant Bus Cycles after STPCLK# is asserted. FLUSH# should not be asserted between the time STPCLK# is asserted and the completion of the second Stop Grant

Bus Cycle. If FLUSH# is asserted during this interval, the system may not see the second Stop Grant Bus Cycle until after STPCLK# is deasserted.

Not tying the STPCLK# signals together gives the flexibility to control either or both the processors' power consumption based on the system performance required. External logic would be required to control this signal to each processor in a DP system.

12.4.3 Stop Grant Bus Cycle

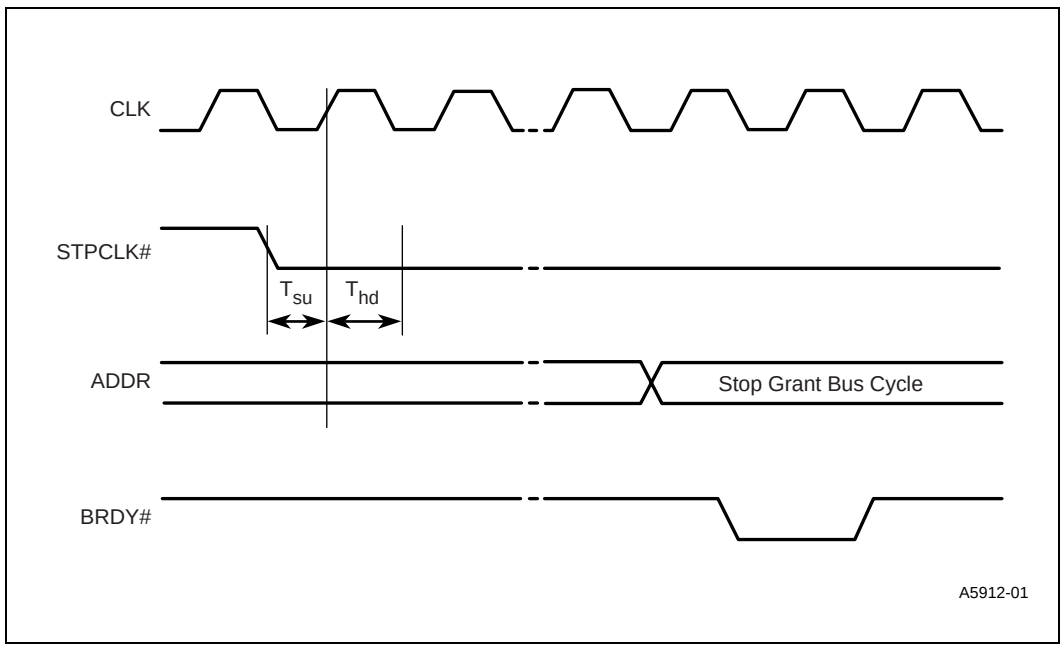
A special Stop Grant bus cycle will be driven to the bus after the processor recognizes the STPCLK# interrupt. The definition of this bus cycle is the same as the HALT cycle definition for the standard Intel486 microprocessor architecture, with the exception that the Stop Grant bus cycle drives the value 0000 0010H on the address pins. In a Dual Processor system, with both STPCLK# signals tied together, two stop grant cycles will occur in a row. The system hardware must acknowledge the Stop Grant cycle by returning BRDY#. The processor will not enter the Stop Grant state until BRDY# has been returned.

The Stop Grant Bus cycle consists of the following signal states: M/IO# = 0, D/C# = 0, W/R# = 1, Address Bus = 0000 0010H (A4 = 1), BE7#–BE0# = 1111 1011, Data bus = undefined.

Note: When operating in dual processing mode, and the STPCLK# signals are tied together, both the Primary processor and Dual processor will go into the Stop Grant state independently, and will each generate a Stop Grant special bus cycle. The system must return BRDY# for both of the special bus cycles.

The latency between a STPCLK# request and the Stop Grant bus cycle is dependent on the current instruction, the amount of data in the processor write buffers, and the system memory performance. Refer to Figure 12-8.

Figure 12-8. Entering Stop Grant State



12.4.4 Pin State During Stop Grant

During the Stop Grant state, most output and input/output signals of the microprocessor will be held at their previous states (the level they held when entering the Stop Grant state). See Table 12-3. However, the data bus and data parity pins will be floated. In response to HOLD being driven active during the Stop Grant state (when the CLK input is running), the processor will generate HLDA and three-state all output and input/output signals that are three-stated during the HOLD/HLDA state. After HOLD is deasserted, all signals will return to their states prior to the HOLD/HLDA sequence.

Table 12-3. Pin State During Stop Grant Bus State

Signal	Type	State
A31–A3	I/O	Previous State
D63–D0	I/O	Floated
BE7#–BE0#	O	Previous State
DP7–DP0	I/O	Floated
W/R#, D/C#, M/IO#	O	Previous State
ADS#, ADSC#	O	Inactive
LOCK#	O	Inactive
BREQ	O	Previous State
HLDA	O	As per HOLD
FERR#	O	Previous State
PCHK#	O	Previous State
PWT, PCD	O	Previous State
SMIACT#	O	Previous State

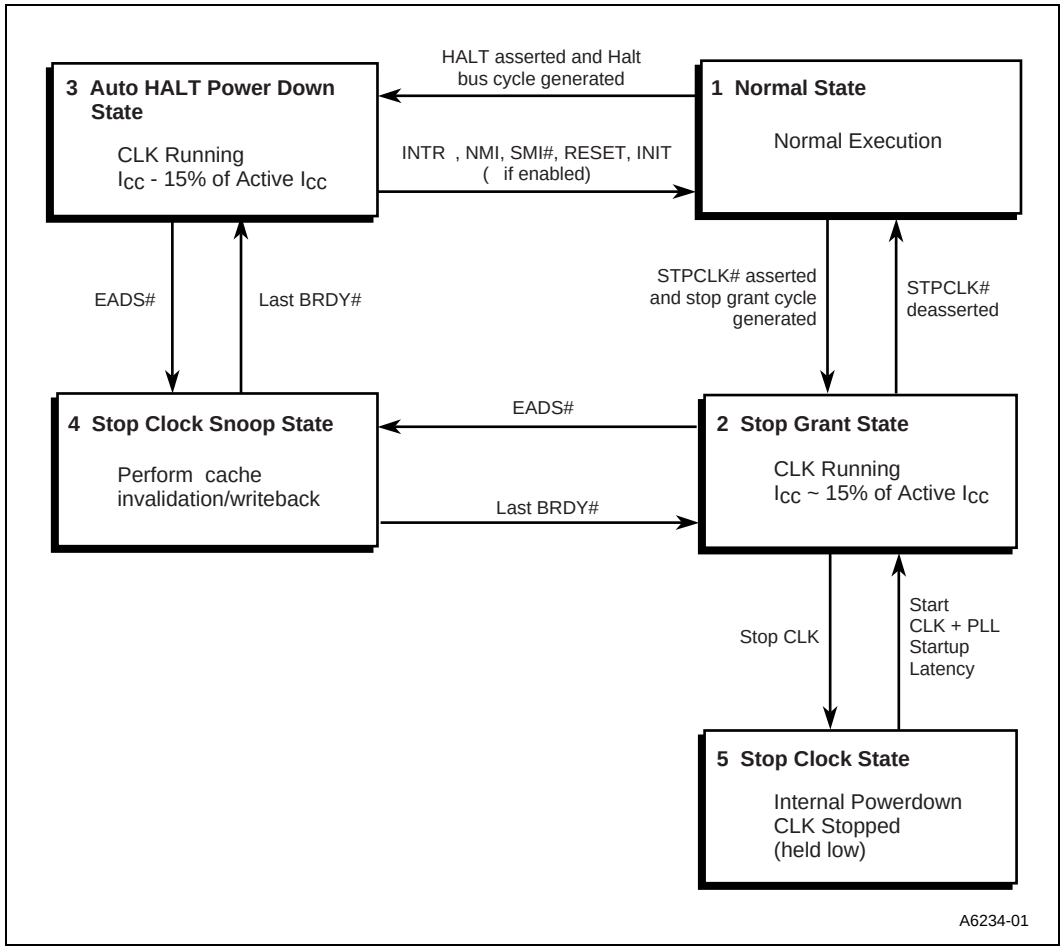
In order to achieve the lowest possible power consumption during the Stop Grant state, the system designer must ensure the input signals with pull-up resistors are not driven low and the input signals with pull-down resistors are not driven high.

All inputs, except data bus pins, must be driven to the power supply rails to ensure the lowest possible current consumption during Stop Grant or Stop Clock modes. Data pins should be driven low to achieve the lowest power consumption. Pull down resistors or bus keepers are needed to minimize the leakage current.

12.4.5 Clock Control State Diagram

Figure 12-9 shows the state descriptions and the state transitions for the clock control architecture.

Figure 12-9. Stop Clock State Machine



A Flush State can be entered from states 1, 2 and 3 by asserting the FLUSH# input signal. The flush state is exited (e.g., the processor returns to the state from which it came) when the Flush Acknowledge Special Bus Cycle is issued by the processor.

12.4.5.1 Normal State — State 1

This is the normal operating state of the processor.

12.4.5.2 Stop Grant State — State 2

The Stop Grant state (~15% of normal state I_{CC}) provides a fast wake-up state that can be entered by simply asserting the external STPCLK# interrupt pin. Once the Stop Grant bus cycle has been placed on the bus, and BRDY# is returned, the processor is in this state. The processor returns to the normal execution state in approximately 10 clock periods after STPCLK# has been deasserted.

For minimum processor power consumption, all other input pins should be driven to their inactive level while the processor is in the Stop Grant state. A RESET will bring the processor from the Stop Grant state to the normal state (note: unless STPCLK# is also deasserted, an active RESET will only bring the processor out of the Stop Grant state for a few cycles). The processor will recognize the inputs required for maintaining cache coherency (e.g., HOLD, AHOLD, BOFF#, and EADS# for cache invalidations and snoops) as explained later in this section. The processor will not recognize any other inputs while in the Stop Grant state. Input signals to the processor will not be recognized until 1 CLK after STPCLK# is deasserted.

While in the Stop Grant state, the processor will latch transitions on the external interrupt signals (SMI#, NMI, INTR, FLUSH#, R/S#, and INIT). All of these interrupts are taken after the deassertion of STPCLK# (e.g., upon re-entering the normal state). The Pentium processor requires INTR to be held active until the processor issues an interrupt acknowledge cycle in order to guarantee recognition.

The processor will generate a Stop Grant bus cycle only when entering that state from the normal state. When the processor enters the Stop Grant state from the Stop Clock Snoop state, the processor will not generate a Stop Grant bus cycle.

12.4.5.3 Auto Halt Powerdown State — State 3

The execution of a HLT instruction will also cause the Pentium processor to automatically enter the Auto HALT Power Down state where I_{CC} will be ~15% of I_{CC} in the Normal state. The processor will issue a normal HALT bus cycle when entering this state. The processor will transition to the normal state upon the occurrence of INTR, NMI, SMI#, RESET, or INIT.

A FLUSH# event during the Auto HALT power down state will be latched and acted upon while in this state.

STPCLK# is not recognized by the processor while in the Auto HALT Powerdown state. The system can generate a STPCLK# while the processor is in the Auto HALT Powerdown state, but the processor will only service this interrupt if the STPCLK# pin is still asserted when the Pentium returns to the normal state.

While in Auto HALT Powerdown state, the processor will only recognize the inputs required for maintaining cache coherency (e.g., HOLD, AHOLD, BOFF#, and EADS# for cache invalidations and snoops) as explained later in this section.

12.4.5.4 Stop Clock Snoop State (Cache Invalidations) — State 4

When the processor is in the Stop Grant state or the Auto HALT Powerdown state, the processor will recognize HOLD, AHOLD, BOFF# and EADS# for cache invalidation/writebacks. When the system asserts HOLD, AHOLD, or BOFF#, the processor will float the bus accordingly. When the system then asserts EADS#, the processor will transparently enter the Stop Clock Snoop state and perform the required cache snoop cycle. It will then re-freeze the clock to the processor core and return to the previous state. The processor does not generate the Stop Grant bus cycle or HALT special cycle when it returns to the previous state.

12.4.5.5 Stop Clock State — State 5

Stop Clock state (~ 1% of normal state I_{CC}) is entered from the Stop Grant state by stopping the CLK input. Note: the CLK must be held at a logic low while stopped. None of the processor input signals should change state while the CLK input is stopped. Any transition on an input signal (with the exception of INTR) before the processor has returned to the Stop Grant state will result in

unpredictable behavior. If INTR is driven active while the CLK input is stopped, and held active until the processor issues an interrupt acknowledge bus cycle, it will be serviced in the normal manner once the clock has been restarted. The system design must ensure the processor is in the correct state prior to asserting cache invalidation or interrupt signals to the processor.

While the processor is in the Stop Clock state, all pins with static pullups or pulldowns must be driven to their appropriate values as specified in the datasheet.

During the Stop Clock state the processor input frequency may be changed to any frequency between the minimum and maximum frequency listed in the AC timing specifications found in the datasheet. To exit out of the Stop Clock state, the CLK input must be restarted and remain at a constant frequency for a minimum of 1 ms. The PLL requires this amount of time to properly stabilize. After the PLL stabilizes, the processor will return to Stop Grant state and the STPCLK# signal may be deasserted to take the processor out of Stop Grant state and back to the Normal state.

In order to realize the maximum power reduction while in the Stop Clock state, PICCLK and TCK should also be stopped. These clock inputs have the same restarting restrictions as CLK. The local APIC cannot be used while in the Stop Clock state since it also uses the system clock, CLK.

Warning: The Stop Clock state feature cannot be used in dual processing or functional redundancy checking modes because there is no way to re-synchronize the internal clocks of the two processors.

Debugging 13

13.1 Introduction

Embedded Pentium® processor-based system designers intending to use integration tools to debug their prototype systems can interface to the processor using two methods:

- Insert an emulator probe into the processor socket.
- Include simple logic on their board that implements a debug port connection.

Inserting an emulator probe into the processor socket allows access to all bus signals, but capacitive loading issues may affect high speed operation. In contrast, the debug port connection allows debugger access to the processor's registers and signals without affecting high speed operation. This allows the system to operate at full speed with the debugger attached. Therefore, Intel recommends that all embedded Pentium processor-based system designs include a debug port.

13.2 Two Levels of Support

Two levels of support are defined for the debug port, the second level being a superset of first. The system designer should choose the level of support that is appropriate for the particular system design and implement that level. Samples of each level of implementation are given in "Implementation Examples" on page 13-4.

13.2.1 Level 1 Debug Port (L1)

The Level 1 debug port supports systems with a single processor. L1 uses a 20-pin connector to allow a debugger access to the processor's registers and signals.

13.2.2 Level 2 Debug Port (L2)

L2 extends the 20-pin debug port connector to 30 pins. The extra ten pins include a second set of boundary scan signals as well as additional R/S# and PRDY signals. The additional R/S# and PRDY signals are added to support the a dual-processor configuration. This enables a debugger to provide separate control over the two processors during debug.

Signals on pins 1 through 20 of the L2 debug port are identical to the signals on the L1 debug port.

13.3 Debug Port Connector Descriptions

A debugger can have a 30-pin connector on its probe that supports both levels of the debug port (as described previously, L1 or L2). Two cables can be provided, each cable having a 30-pin connector at one end (to mate with the debugger's probe connector) and the appropriate size connector at the

other end to mate with the debug port in the system under debug. (For example, the L1 debug port Cable can be a 20-conductor cable with a 20-pin connector at one end and a 30-pin connector at the other end, leaving pins 21 to 30 unconnected.)

Intel-recommended connectors for mating with debug port cables are available in either a vertical or right-angle configuration. Use the one that fits best in your design. The connectors are manufactured by AMP Incorporated and are in their AMPMODU System 50 line. Table 13-1 shows the AMP part numbers for the various connectors:

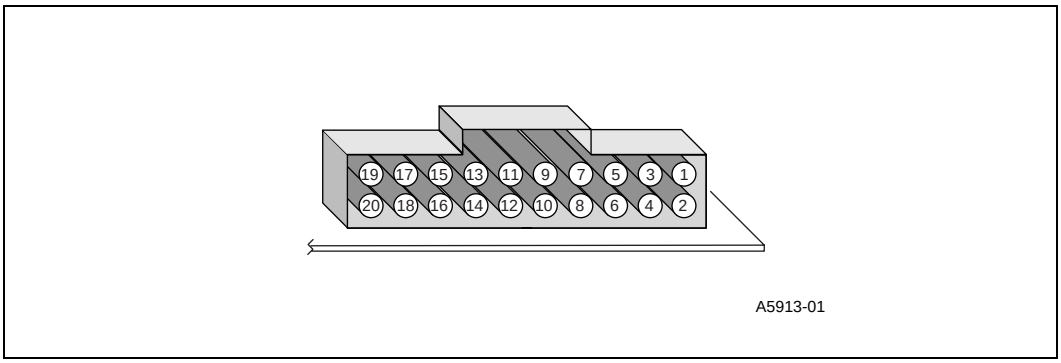
Table 13-1. Recommended Connectors

Connector	Vertical	Right-Angle
20-pin shrouded header	104068-1	104069-1
30-pin shrouded header	104068-3	104069-5

Note: These are high density through hole connectors with pins on 0.050” by 0.100” centers. Do not confuse these with the more common 0.100” by 0.100” center headers.

Figure 13-1 is an example of the pinout of the connector footprint as viewed from the connector side of the circuit board. This is just an example. Contact your third-party tools vendor to determine the correct implementation for the tool you will use. Note that the 30-pin connector is a logical extension of the 20-pin connector with the key aligned with pin 15.

Figure 13-1. Debug Port Connector



13.4 Signal Descriptions

Table 13-2 shows the debug port signals. Direction is given as follows: O = output from the Pentium processor-based board to a debugger; I = input to the Pentium processor-based board from a debugger. These are either 2.5 V or 3.3 V signals, depending on the Pentium processor used in the system. For the L1 debug port, ignore signals on pins 21 through 30.

Note: Target systems should be sure to provide a way for debugging tools like emulators, in-target probes and logic analyzers to reset the entire system, including upgrade processor, chip sets, etc. For example, if you follow the debug port implementation described below, the DBRESET signal provides this functionality. If you are not implementing the debug port, make sure that your system has a test point connected into the system reset logic to which a debug tool can connect.

Table 13-2. Debug Port Signals (Sheet 1 of 2)

Signal Name	Dir	Pin	Description
INIT	O	1	(Pentium® processor signal). A debugger may use INIT to support emulating through the processor INIT sequence while maintaining breakpoints or breaking on INIT.
DBRESET	I	2	Debugger Reset output. A debugger may assert DBRESET (high) while performing the “RESET ALL” and “RESET TARGET” debugger commands. DBRESET should be connected to the system reset circuitry such that the system and processor(s) are reset when DBRESET is asserted. This is useful for recovering from conditions like a “ready hang.” This signal is asynchronous.
RESET	O	3	(Pentium processor signal). A debugger may use RESET to support emulating through the reset while maintaining breaking on RESET.
GND		4	Signal ground.
SMIACT#	0	5	(Pentium processor signal) System Management mode interrupt active.
V_{CC}		6	V_{CC} from the system. A debugger uses this signal to sense that system power is on and to determine signal I/O voltage levels. Connect this signal to V_{CC3} through a 1 KOhm (or smaller) resistor.
R/S#	I	7	Connect to the R/S# pin of the.
GND		8	Signal ground.
NC		9	No connect. Leave this pin unconnected.
GND		10	Signal ground.
PRDY	O	11	From the PRDY pin of the.
TDI	I	12	Boundary scan data input (signal). This signal connects to TDI of the. For dual processor operation, TDI of the Dual would connect to TDO of the.
TDO	O	13	Boundary scan data output (signal). This signal connects to TDO from the for a single processor design, or to TDO from the Dual Pentium for dual processor operation.
TMS	I	14	Boundary scan mode select (signal).
GND		15	Signal ground.
TCK	I	16	Boundary scan clock (signal).
GND		17	Signal ground.
TRST#	I	18	Boundary scan reset (signal).
DBINST#	I	19	DBINST# is asserted (connected to GND) while the debugger is connected to the debug port. DBINST# can be used to control the isolation of signals while the debugger is installed.
BSEN#	I	20	Boundary scan enable. This signal can be used by the system to control multiplexing of the boundary scan input pins (TDI, TMS, TCK, and TRST# signals) between the debugger and other boundary scan circuitry in the system. The debugger asserts (low) BSEN# when it is driving the boundary scan input pins. Otherwise, the debugger drivers are high impedance. If the boundary scan pins are actively driven by the system, then BSEN# should control the system drivers/multiplexers on the boundary scan input pins. See “Example 2: Single Processor, Boundary Scan Used by System” on page 13-6.
PRDY2	O	21	From the PRDY pin of the Dual (for dual processor operation).

Table 13-2. Debug Port Signals (Sheet 2 of 2)

Signal Name	Dir	Pin	Description
GND		22	Signal ground.
R/S#2	I	23	Connect to the R/S# pin of the Dual (for dual processor operation).
NC		24	
NC		25	
NC		26	
NC		27	
NC		28	
GND		29	Signal ground.
NC		30	

13.5 Signal Quality Notes

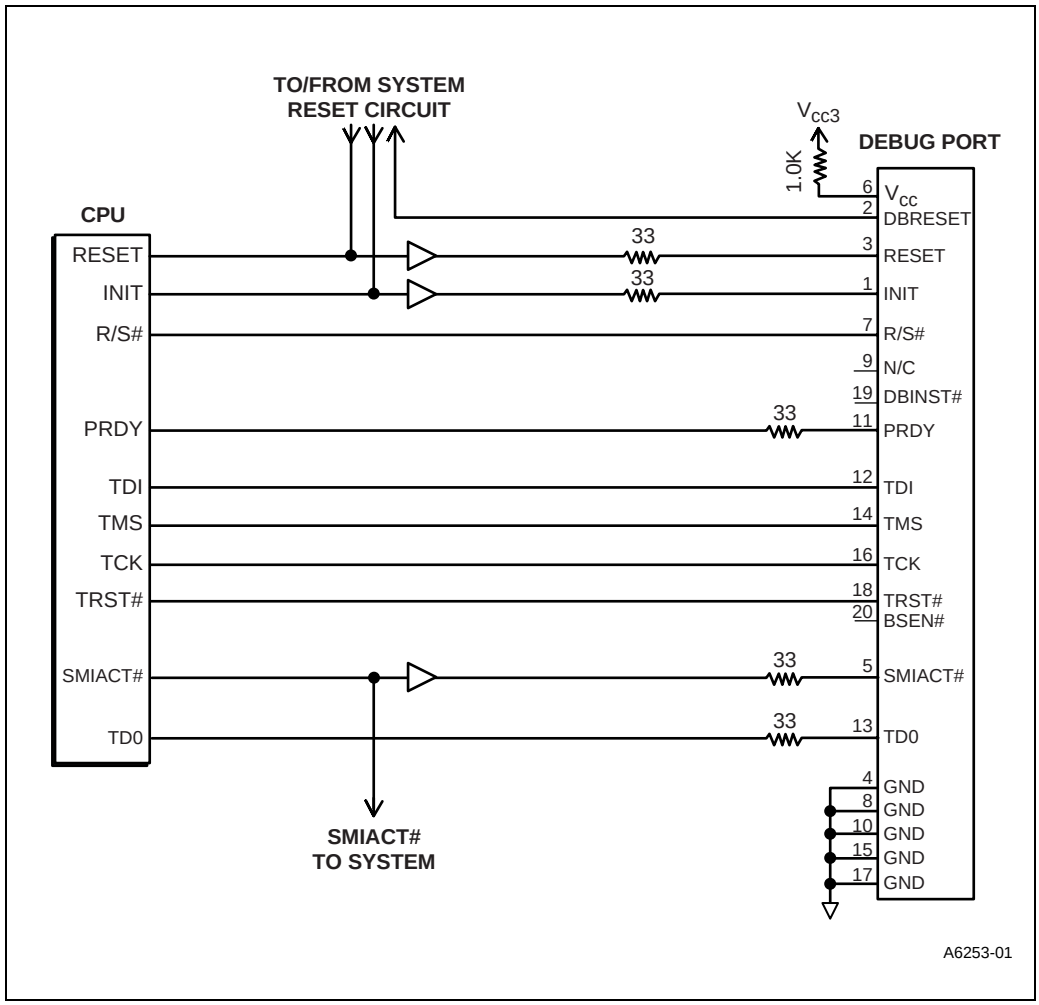
Since debuggers can connect to the system via cables of significant length (e.g., 18 inches), care must be taken in Pentium processor-based system design with regard to the signals going to the debug port. If system outputs to the debug port (i.e., TDO, PRDY, INIT and RESET) are used elsewhere in the system they should have dedicated drivers to the debug port. This will isolate them from the reflections from the end of the debugger cable. Series termination is recommended at the driver output. If the boundary scan signals are used elsewhere in the system, then the TDI, TMS, TCK, and TRST# signals from the debug port should be isolated from the system signals with multiplexers.

13.6 Implementation Examples

13.6.1 Example 1: Single Processor, Boundary Scan Not Used by System

Figure 13-2 shows a schematic of a minimal Level 1 debug port implementation for a Pentium processor, single-processor system in which the boundary scan pins of the are not used in the system.

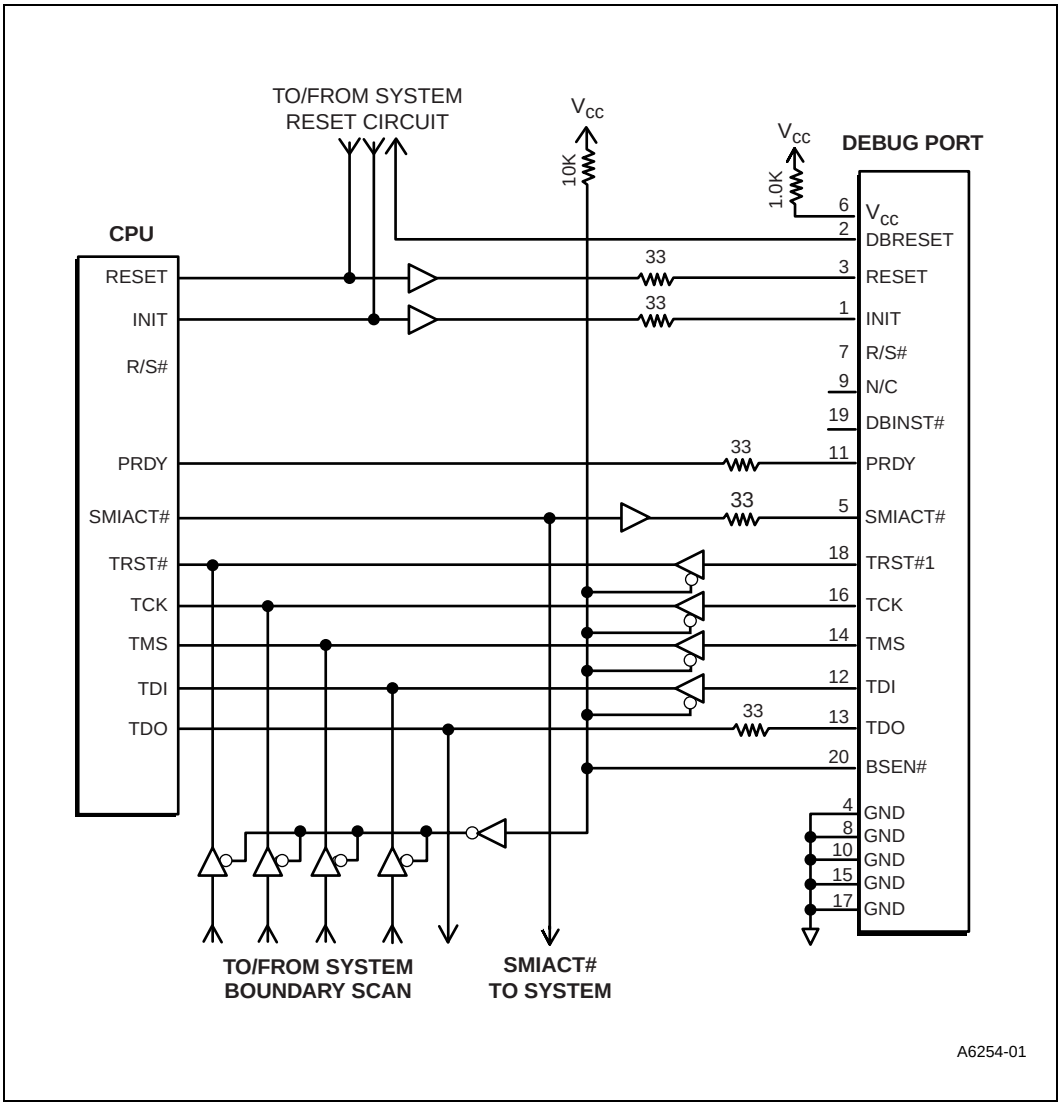
Figure 13-2. Single Processor – Boundary Scan Not Used



13.6.2 Example 2: Single Processor, Boundary Scan Used by System

Figure 13-3 shows a schematic of a Level 1 debug port implementation for a single-processor system in which the boundary scan pins are used. Note that the BSEN# signal controls the multiplexing of the boundary scan signals. With this implementation, the system could use the boundary scan (through the) while the debugger is “emulating,” but could not while the debugger is “halted” (because the chain is broken).

Figure 13-3. Single Processor – Boundary Scan Used

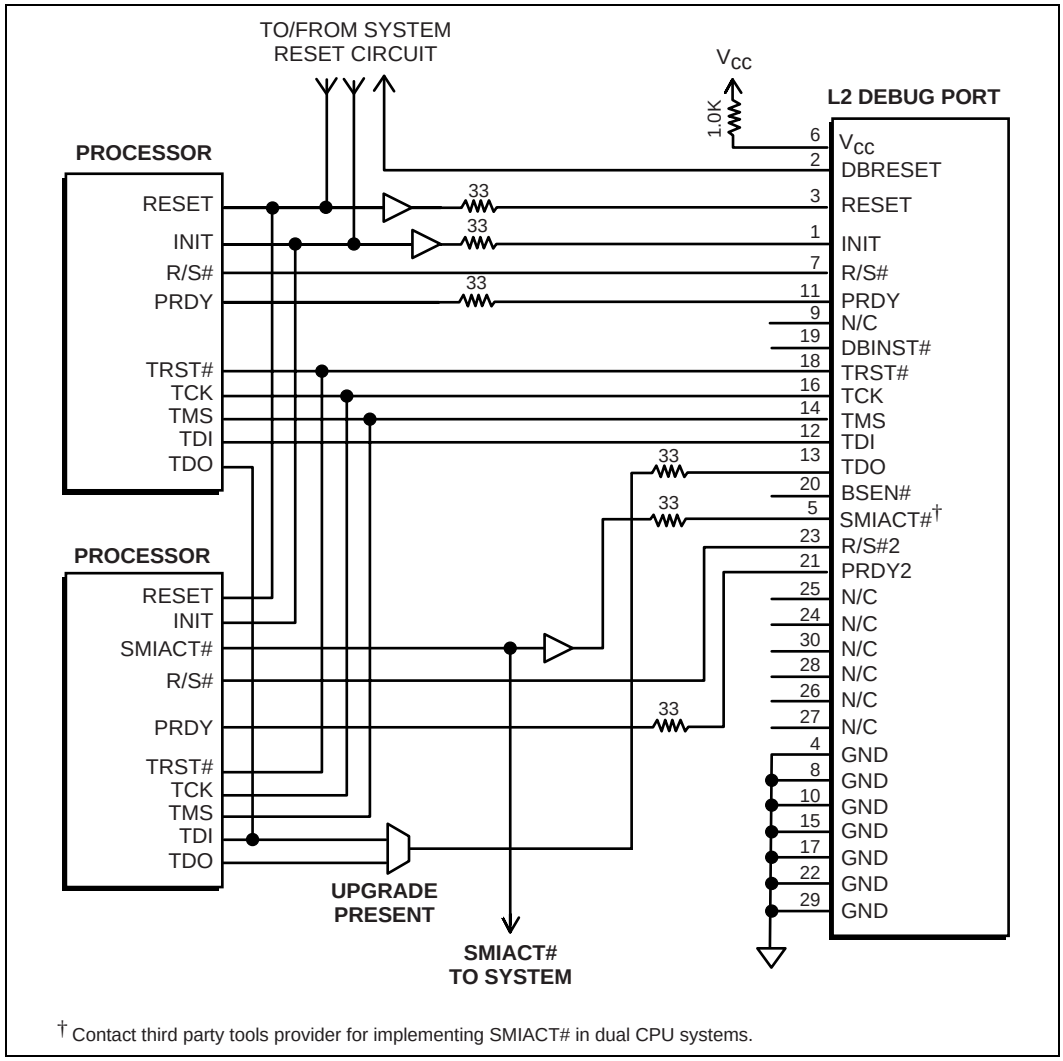


13.6.3 Example 3: Dual Processors, Boundary Scan Not Used by System

Figure 13-4 shows a schematic of a typical Level 2 debug port implementation for a Pentium processor, dual-processor system in which the boundary scan pins are not used. The multiplexer circuit for use with the “upgrade socket” concept is shown, but could be replaced with a jumper.

Note: Contact your third-party tools vendor for information on implementing SMIACT# in a dual processor system.

Figure 13-4. Dual Processor – Boundary Scan Not Used

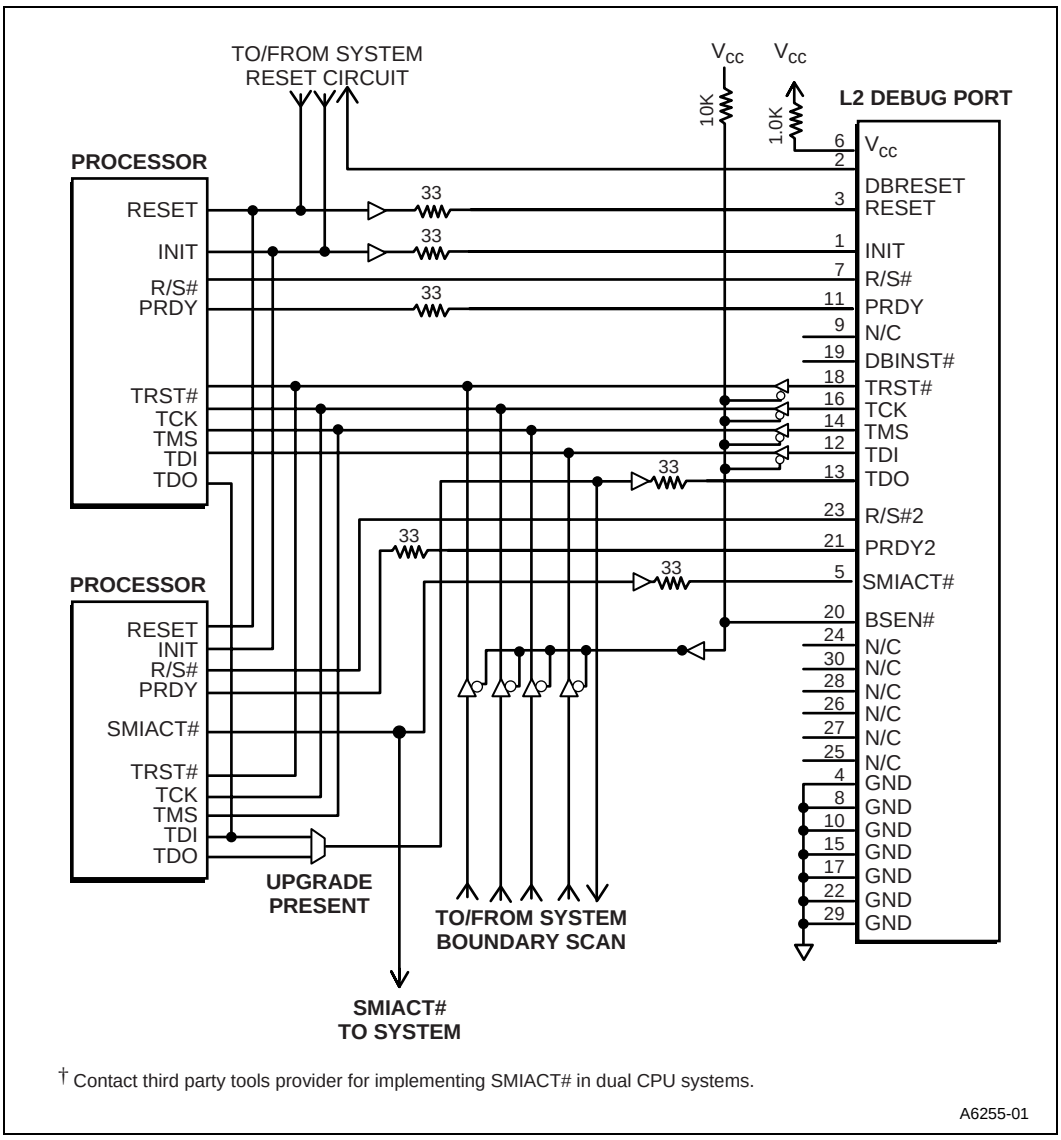


13.6.4 Example 4: Dual Processors, Boundary Scan Used by System

Figure 13-5 shows a schematic of a Level 2 debug port implementation for a dual-processor system that uses boundary scan. Note that the BSEN# signal controls the multiplexing of the boundary scan signals. With this implementation, the system could use the boundary scan (through the) while the debugger is "emulating," but could not while the debugger is "halted" (because the chain is broken).

Note: Contact your third-party tools vendor for information on implementing SMIACT# in a dual processor system.

Figure 13-5. Dual Processor – Boundary Scan Used



13.7 Implementation Details

13.7.1 Signal Routing Note

The debugger software communicates with the processor through the debug port using the boundary scan signals listed above. Typically, the debugger expects the processor to be the first and only component in the scan chain (from the perspective of the debug port). That is, it expects TDI to go directly from the debug port to the TDI pin of the processor, and the TDO pin to go directly from the processor to the debug port (see Figure 13-6). If you have designed your system so that this is not the case (for instance, see Figure 13-7), you will need to provide the debugger software with the following information: (1) position of the processor in the scan chain, (2) the length of the scan chain, (3) instruction register length of each device in the scan chain. Without this information the debugger will not be able to establish communication with the processor.

Figure 13-6. Example of Processor Only in Scan Chain

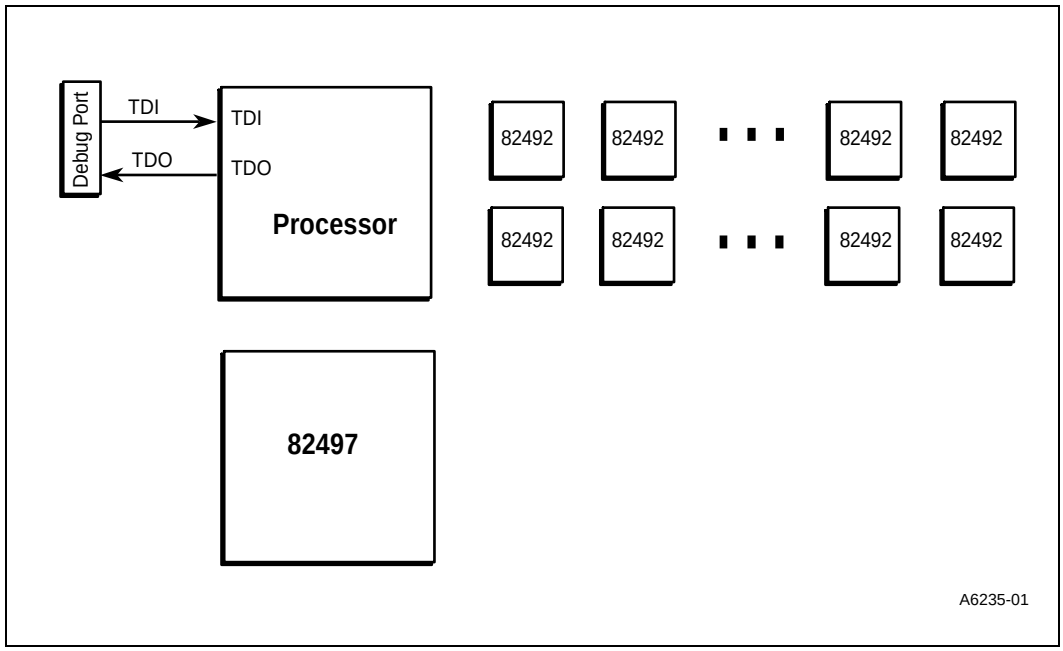
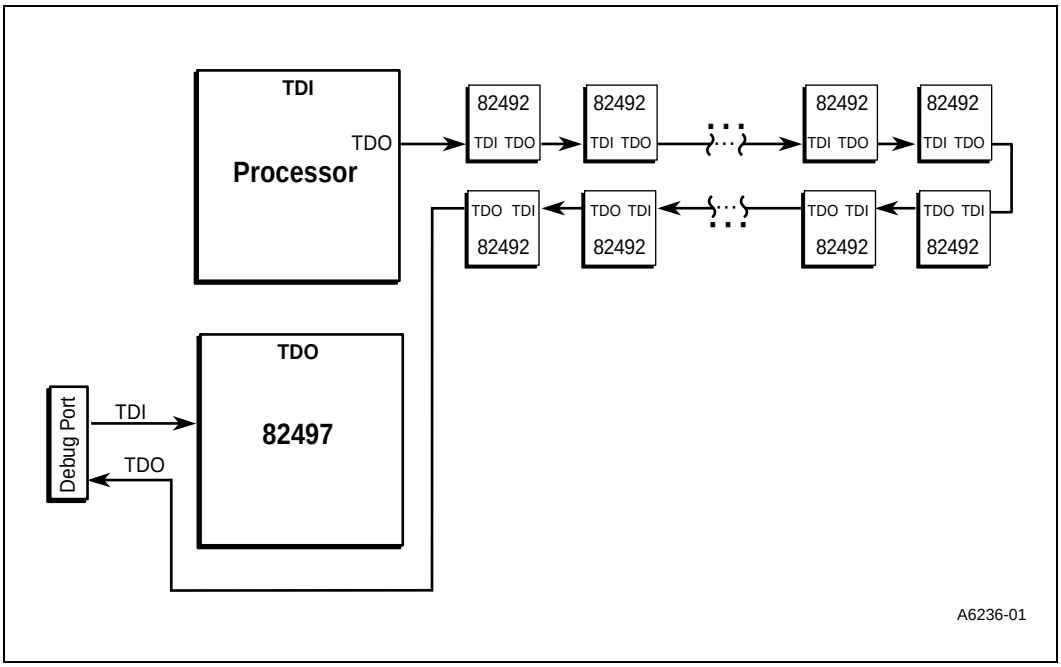


Figure 13-7. Example of Multiple Components in Scan Chain



13.7.2 Special Adapter Descriptions

For those designs where board real estate is a concern or where the design is finished and it is too late to implement the debug port, it may be possible to use a special “debug port adapter” to replace the on-board debug port described in the previous sections. The purpose of the adapter is to provide easy access to the boundary scan signals of the processor(s). For simplicity, the adapter should make the boundary scan signals accessible to the debug tool while at the same time preventing the target system from accessing them. Two debug port adapters are described: (1) for uniprocessor debug, (2) for dual-processor debug.

Standard PPGA adapters are available from many third-party tools vendors.

13.7.2.1 Uniprocessor Debug

A debug port adapter for use in uniprocessor systems, or dual-processor systems where only one processor will be debugged at a time, can be built by reworking two Pentium processor SPGA sockets (see Figure 13-8).

Note: This adapter can be used only when the processor is *not* included in the target system boundary scan string. In addition, when used in dual-processor systems you will only be able to debug the processor to which the adapter is connected.

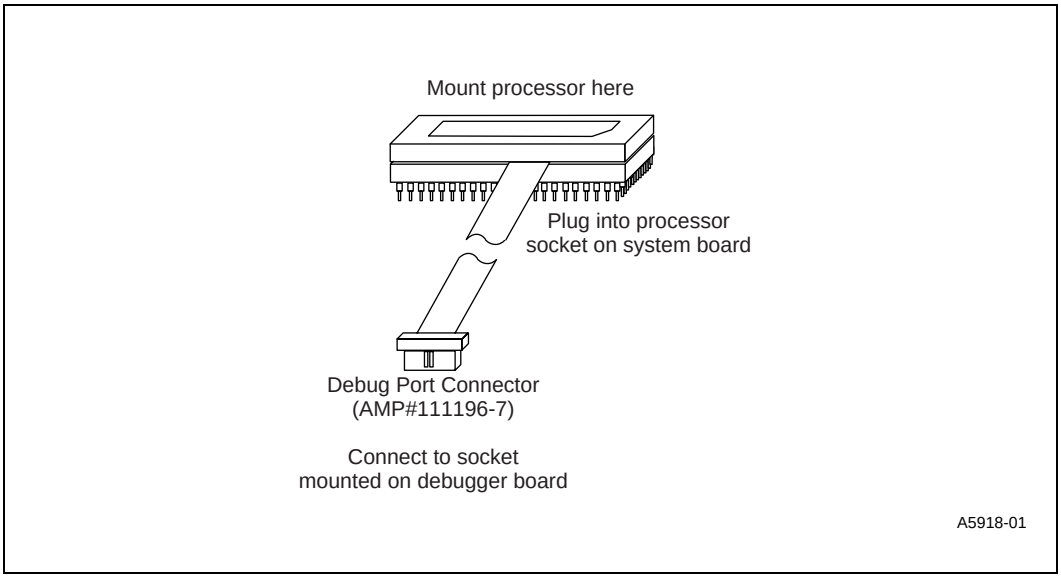
Table 13-3 show which pins to connect lines of appropriate 20- or 30- wire cable to on the top socket.

Table 13-3. SPGA Socket

Cable Wire Number	SPGA Pin Number	Signal
1	AA33	INIT
2	NC	DRESET
3	AK20	RESET
4	AD36 (V_{SS})	GND
5	AG03	SMIACT#
6	U37 (V_{CC3})	V_{CC}
7	AC35	R/S#
8	AB36 (V_{SS})	GND
9	NC	NC
10	Z36 (V_{SS})	GND
11	AC05	PRDY
12	N35	TDI
13	N33	TDO
14	P34	TMS
15	X36 (V_{SS})	GND
16	M34	TCK
17	R36 (V_{SS})	GND
18	Q33	TRST#
19	NC	DBINST#
20	NC	BSEN#

Note: You may connect the GND pins to any pin marked V_{SS} on the SPGA pinout diagram. The NC pins are no connects. You may simply cut those wires.

Figure 13-8. Uni-Processor Debug



Connect a double-row receptacle (AMP# 111196-7) to the debug port connector end of the cable. This is a 30-pin connector, so that it fits into the socket on the debugger buffer board.

Remove the following pins from the bottom socket:

R/S#	AC35
PRDY	AC05
TDI	N35
TDO	N33
TMS	P34
TCK	M34
TRST#	Q33

Connect the two sockets together. Make sure not to crush the wires between the pins.

13.7.2.2 Dual-Processor Debug

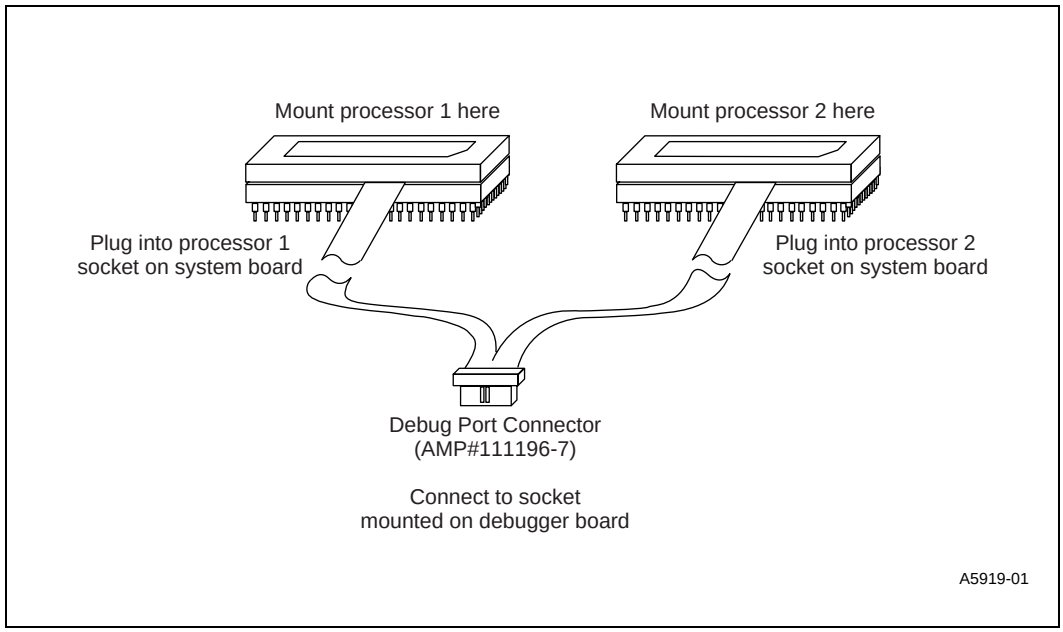
A debug port adapter for use in dual processor debugging can be built by reworking four Pentium processor-based SPGA sockets. (See Figure 13-9).

Note: This adapter can be used only when the processors are *not* included in the target system boundary scan string.

You will need to use two SPGA sockets per processor location. For this discussion, assume that the startup processor is called processor 1 and that the upgrade processor is called processor 2. Thus, you will use two SPGA sockets to connect to processor 1 and two SPGA sockets to connect to

processor 2. Certain debug port signals must be shared by Processor 1 and Processor 2. These signals must be connected from the debug port connector end of the cable (on which you will place a double-row receptacle: AMP# 111196-7) to both double SPGA sockets.

Figure 13-9. Dual-Processor Debug Port Adapter



Connect lines of 30-wire cable to the pins on the top SPGA sockets for both processor 1 and 2. Following are the signals which should be connected to each processor socket. Make sure to connect the shared lines to both top sockets.

Figure 13-10. Shared Pins for Dual-Processor Adapter

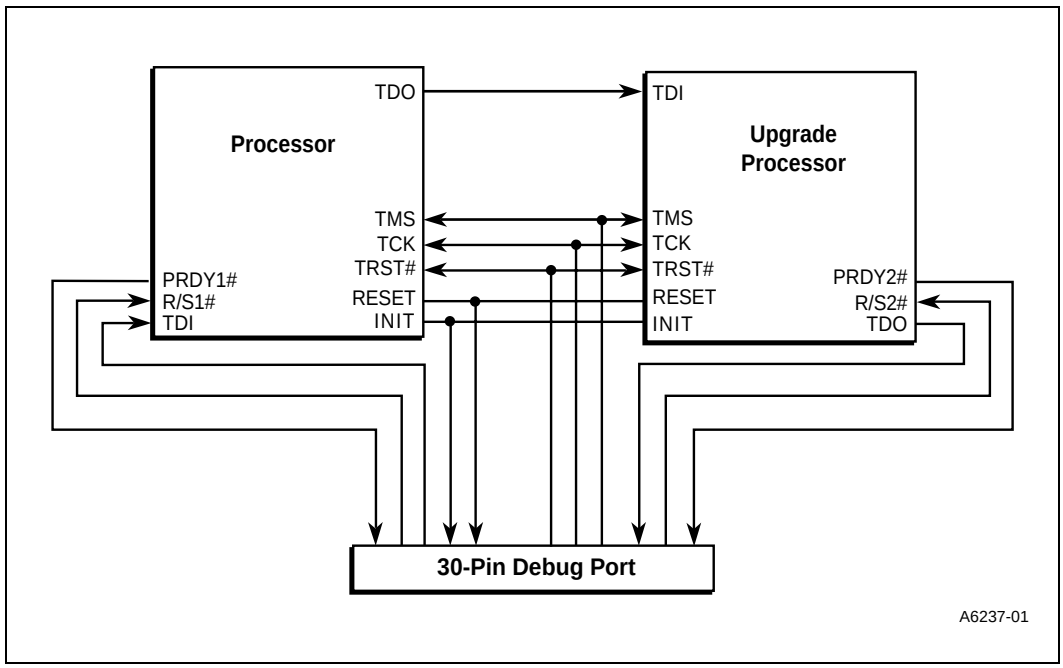


Table 13-4. Debug Port Connector Pinout

Cable Wire Number	SPGA Pin Number	Processor Socket	Signal
1	AA33	1,2	INIT
2	NC		DBRESET
3	AK20	1,2	RESET
4	V_{SS}	1	GND
5	AG03		SMIACT#†
6	V_{CC}	1	V_{CC}
7	AC35	1	R/S1#
8	V_{SS}	1	GND
9	NC		NC
10	V_{SS}	1	GND
11	AC05	1	PRDY1
12	N35	1	TDI
13	N33	2	TDO
14	P34	1,2	TMS
15	V_{SS}	1	GND
16	M34	1,2	TCK
17	V_{SS}	1	GND
18	Q33	1,2	TRST#
19	NC		DBINST#
20	NC		BSEN#
21	AC05	2	PRDY2
22	V_{SS}	2	GND
23	AC35	2	R/S2#
24	NC		NC
25	NC		NC
26	NC		NC
27	NC		NC
28	NC		NC
29	V_{SS}	2	GND
30	NC		NC

† Contact your third-party tools vendor for information on implementing SMIACT# in a dual processor system.

Note: You can connect the V_{CC} and GND pins to any convenient power or ground pin.

Connect a double-row receptacle (AMP# 111196-7) to the debug port end of the cable. This is a 30-pin connector, so that it fits into the socket on the debugger buffer board.

Remove the following pins from the bottom of both double sockets:

R/S#	AC35
PRDY	AC05
TDI	N35
TDO	N33
TMS	P34
TCK	M34
TRST#	Q33

Connect each set of two sockets together. Make sure not to crush the wires between the pins.

Model Specific Registers and Functions 14

This chapter introduces the model specific registers (MSRs) as they are implemented on the embedded Pentium® processor family. Model specific registers are used to provide access to features that are generally tied to implementation dependent aspects of a particular processor. For example, testability features that provide test access to physical structures such as caches, and branch target buffers are inherently model specific. Features to measure the performance of the processor or particular components within the processor are also model specific.

The features provided by the model specific registers are expected to change from processor generation to processor generation and may even change from model to model within the same generation. Because these features are implementation dependent, they are not recommended for use in portable software. Specifically, software developers should not expect that the features implemented within the MSRs will be supported in an upward or downward compatible manner across generations or even across different models within the same generation.

The embedded Pentium processor with MMX™ technology MSRs are different than the embedded Pentium processor MSRs. When possible, fields were preserved between the two processors. Differences between the MSRs are noted throughout this chapter.

14.1 Model Specific Registers

The embedded Pentium processor processor family implements the RDMSR and WRMSR instructions to read and write the MSR's respectively. A feature bit in EDX (bit 5), reported by the CPUID instruction, indicates whether the processor supports the RDMSR and WRMSR instructions. The Pentium processor with MMX technology implements a new instruction called RDPMC (Read Performance Monitoring Counter). This instruction enables the user to read the performance monitoring counters in "Current Privilege Level = 3" given bit 8 is set in CR4 (CR4.PCE).

14.1.1 Model Specific Register Usage Restrictions

Proper use of the MSR features described in this chapter requires that the CPUID instruction be used not only to validate that the FAMILY reported in the EAX register is equal to "5", but also to validate the specific MODEL number within that FAMILY. Note that this requirement is significantly more restrictive than is required for new architectural features where it is sufficient to validate that the FAMILY is equal to or greater than that of the first family to implement the new feature. For more information regarding the use of the CPUID instruction, refer to the *Intel Architecture Software Developer's Manual*.

14.1.2 Model Specific Register Access

Access to the model specific registers is provided through the RDMSR and WRMSR instructions. Access to a particular MSR is achieved by loading the ECX register with the appropriate ECX value from Table 14-1 below, and then executing either RDMSR or WRMSR. For more information regarding the use of these instructions, refer to the *Intel Architecture Software Developer's Manual.*

Table 14-1. Model Specific Register Descriptions

ECX Value (in Hex)	Register Name	Description
00	Machine Check Address(1)	Stores address of cycle causing the exception
01	Machine Check Type(1)	Stores cycle type of cycle causing the exception
02	Test Register 1	Parity Reversal Register
03	RESERVED	
04	Test Register 2(2)	Instruction Cache End Bit
05	Test Register 3	Cache Test Data
06	Test Register 4	Cache Test Tag
07	Test Register 5	Cache Test Control
08	Test Register 6	TLB Test Linear Address
09	Test Register 7	TLB Test Control & Physical Address 31–12
0A	RESERVED	
0B	Test Register 9	BTB Test Tag
0C	Test Register 10	BTB Test Target
0D	Test Register 11	BTB Test Control
0E	Test Register 12	New Feature Control
0F	RESERVED	
10	Time Stamp Counter	Performance Monitor
11	Control and Event Select	Performance Monitor
12	Counter 0	Performance Monitor
13	Counter 1	Performance Monitor
14+	RESERVED	

NOTES:
1. CR4.MCE must be 1 in order to utilize the machine check exception feature.
2. Reserved on the embedded Pentium® processor with MMX™ technology.

14.2 Testability And Test Registers

The processor provides testability access to the on-chip caches, TLBs, BTB and internal parity checking features through model specific test registers. The RDMSR/WRMSR instructions may be utilized by the processor to access the test registers.

14.2.1 Cache, TLB and BTB Test Registers

The processor contains several test registers. The purpose of these test registers is to provide direct access to the processor's caches, Translation Look-aside Buffers (TLB), and Branch Target Buffer (BTB) so test programs can easily exercise these structures. Because the architecture of the caches, TLBs, and BTB is different, a different set of test registers (along with a different test mechanism) is required for each processor family member. Most test registers are shared between the code and data caches.

The test registers should be written to for testability purposes only. Writing to the test registers during normal operation causes unpredictable behavior. Note that when the test registers are used to read or write lines directly to or from the cache, external inquire cycles must be inhibited to guarantee predictable results when testing. This is done by setting both CR0.CD and CR0.NW to "1". In addition, the INVD, WBINVD and INVLPG instructions may be executed before and after but not during testing.

Caution: Writing to the test registers during normal operation causes unpredictable behavior.

Since the on-board caches, TLBs, and BTB implemented in embedded Pentium processor with MMX technology differ than those in embedded Pentium processor, the test register interface differs.

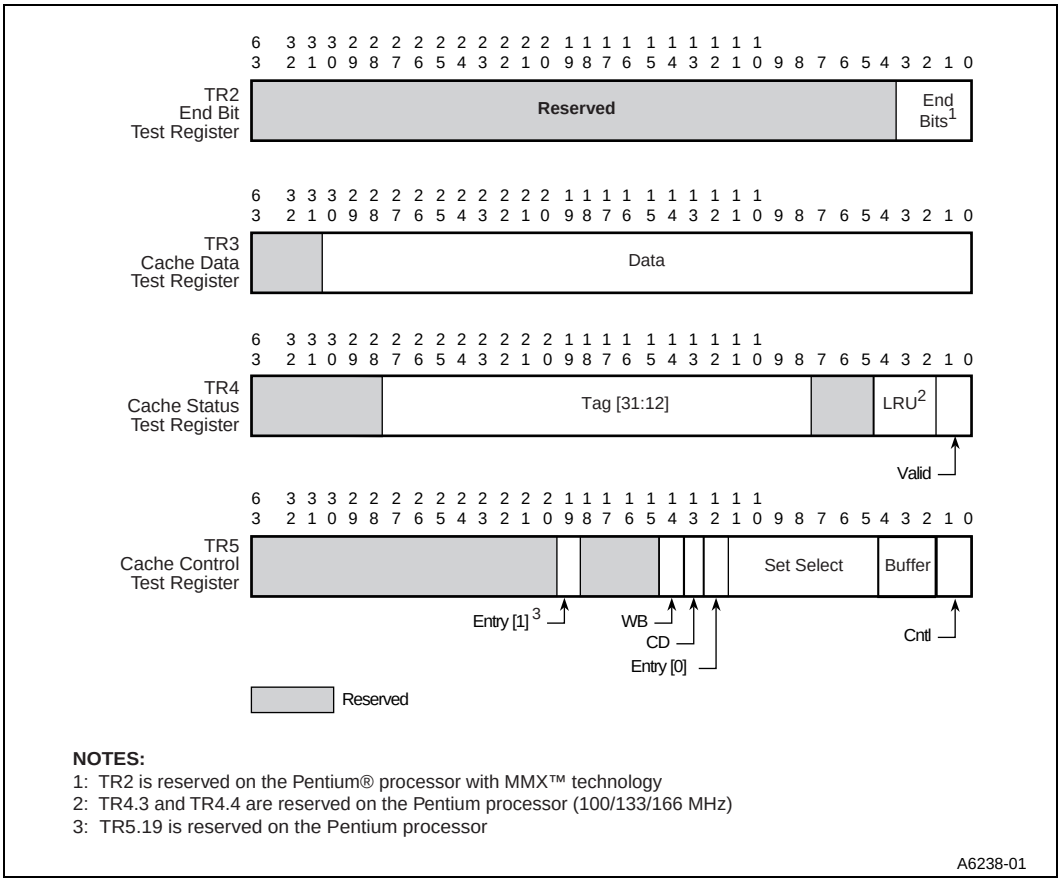
If a memory data access occurs during a code cache testability operation using the test registers, the data cache is checked before the external memory operation in initiated. If the access is a miss in the data cache, then if the accessed line is valid in the code cache, it is invalidated through the internal snooping mechanism. In addition, the same cache line fill buffer is used for cache testability writes and to temporarily store data from memory data reads. For this reason, memory data reads should be done with care or avoided to ensure data from the memory read does not overwrite data from the testability write in the cache line fill buffer.

Similarly, if a code access occurs during a data cache testability operation using the test registers, the code cache is checked before the external memory operation is initiated. If the access is a miss in the code cache, then the accessed line if valid in the data cache is invalidated (or written back and then invalidated if in the M state) through the internal snooping mechanism.

14.2.1.1 Cache Test Registers

The registers in Figure 14-1 provide direct access to the Pentium processor's code and data caches.

Figure 14-1. Cache Test Registers



NOTES:
1: TR2 is reserved on the Pentium® processor with MMX™ technology
2: TR4.3 and TR4.4 are reserved on the Pentium processor (100/133/166 MHz)
3: TR5.19 is reserved on the Pentium processor

A6238-01

On the embedded Pentium processor, TR2 is the End Bit Test Register for the code cache. It contains four end bits. Each end bit corresponds to one byte of instruction in TR3 during code cache testability access. Since a cache line has 32 bytes, eight accesses are needed to read or write the end bits for the entire cache line. TR2 is used for accesses to the code cache only. TR2 is reserved on the embedded Pentium processor with MMX technology.

End bits are used to indicate instruction boundaries on the embedded Pentium processor. The end bit mechanism aids the decode of two variable length instructions per clock by providing information on where the boundary between instruction is. If a given byte is the last byte in an instruction, the corresponding end bit is set to one. When a line is written into the code cache after a miss, all end bits corresponding to the line are initialized to one. As instructions are decoded, the end bits are checked for correctness and modified if incorrect. In order for two instructions to be issued in a single clock, the end bits of the u-pipe instruction must have the correct values, otherwise only one instruction will be issued. This does have the effect that instructions are usually not paired the first time that they are put in the code cache.

TR3 is the Cache Data Test Register. This is where the data is held on its way into or out of the cache. Prior to a cache testability write, software must load an entire cache line into the 32-byte fill buffer using TR3, 4 bytes at a time. Similarly, during a cache testability read, the processor extracts a specified 4-byte data quantity from a cache line and places the data in TR3. A 32-byte cache line may be written to or read from TR3 as eight 4-byte accesses.

TR4 is the Cache Status Test Register. It contains the tag, LRU and valid bits to be written to or read from the cache. Like TR3, TR4 must be loaded with the tag/LRU/valid bits prior to a testability write, and gets updated with the tag/LRU/valid bits as a result of a testability read. Note that TR4[31:28] are reserved and always return a zero as a result of a testability read.The two valid bits are interpreted differently by the code and data caches, depending upon the setting of TR5.CD bit. The encodings for TR4.valid are shown in Table 14-2. The encodings for the LRU bits are shown in Table 14-3 for the embedded Pentium processor and the embedded Pentium processor with MMX technology.

Table 14-2. Encoding for Valid Bits in TR4

TR5.CD=1 (Data Cache)	valid[1]	valid[0]	Meaning
	0	0	Cache line in I state
	0	1	Cache line in S state
	1	0	Cache line in E state
	1	1	Cache line in M state
TR5.CD=0 (Code Cache)	**valid[1]**	**valid[0]**	**Meaning**
	X	0	Cache line invalid
	X	1	Cache line valid

Table 14-3. Encoding of the LRU Bit in TR4

Pentium® Processor (100/133/166)			
LRU[0]		**Points to WAY**	
0		0	
1		1	
Pentium Processor with MMX™ Technology			
LRU[2]	**LRU[1]**	**LRU[0]**	**Points to WAY**
X	0	0	0
X	1	0	1
0	X	1	2
1	X	1	3

Note: The LRU bits for the instruction cache change state when an entry is read using the test registers, with CR0.CD=1. The LRU bits for the data cache, however, do not change their state during testability reads with CR0.CD=1.

TR5 is the Cache Control Test Register. It contains the writeback bit, the CD bit, the cache entry, the set address, the buffer select, and a two-bit control field, cntl.

The writeback bit determines whether that particular line is configured for writethrough or allows the possibility of writeback. It is used by the data cache only (i.e., if the writeback bit is set and a flush occurs (TR5.cntl=11), then if the addressed line in the data cache is modified, it will be invalidated and written back to the bus). The CD bit distinguishes between the code and data cache. The entry field selects one of the four ways in the embedded Pentium processor with MMX technology (two ways in the embedded Pentium processor) in the cache. The set address field selects one of 128 sets within the cache to be accessed. The buffer field selects one of the eight portions of a cache line to be visible through TR3. The control field selects one of the four possible operation modes. The encodings for the TR5 fields are shown in Table 14-4, Table 14-5, Table 14-6 and Table 14-7.

Table 14-4. Encoding of the WB Bit in TR5

WB	Writeback or Writethrough
0	Writethrough
1	Writeback

Table 14-5. Encoding of the Code/Data Cache Bit in TR5

CD	Cache
0	Code cache
1	Data cache

Table 14-6. Encoding of the Entry Bit in TR5

Entry[1]	Entry[0]	Way
0	0	0
0	1	1
1	0	2
1	1	3

Note: The Entry[1] bit, Way 2 and Way 3 are specific to the embedded Pentium processor with MMX technology.

Table 14-7. Encoding of the Control Bits in TR5

Cntl1	Cntl0	Command
0	0	Normal operation
0	1	Testability write
1	0	Testability read
1	1	Flush

Direct Cache Access

To access the cache for testing, the programmer specifies a set address and entry and requests a testability read or write. No tag comparison is done; the programmer can directly read/write a particular entry in a particular set. Note that since TR2 is reserved for the embedded Pentium processor with MMX technology, there is no TR2 access when reading an entry from the cache.

To write down an entry into the cache:

- Disable replacements by setting CR0.CD=1.
- For each 4-byte access:
  - — Write address into TR5.buffer. Here, TR5.cntl=00.
  - — Write data into TR3.
  - — Write end bits into TR2 (for instruction cache only).
- Write the desired tag, LRU and valid bits into TR4. Note that the contents of TR4 completely overwrites the previous tag, LRU and valid bits in the cache.
- Perform a testability write by loading TR5 with the appropriate CD, entry, set address, and cntl fields. Here, TR5.cntl=01.

To read an entry from the cache:

- For each 4-byte access:
  - — Write the appropriate CD, entry, set address, buffer and cntl fields into TR5. Here, TR5.cntl=10.
  - — Read data from TR3.
  - — Read end bits from TR2[3:0] (for instruction cache only).
  - — Read the tag, LRU, and valid bits from TR4. No hit/comparison is performed. Whatever was in that entry in that set is read into TR4, TR3, and TR2.

To invalidate the cache or invalidate an entry:

- When TR5.cntl=11 (flush), and CD=0 (code cache), the entire code cache is invalidated. However, if TR5.cntl=11 and CD=1 (data cache), the user can specify through the TR5.WB bit whether to invalidate the entire data cache, or invalidate and writeback only the cache line specified by TR5 (see Figure 14-8).

Table 14-8. Definition of the WB Bit in TR5

TR5.cntl=11	TR5.WB	Meaning
CD=0	X	Invalidate the entire code cache.
CD=1	0	Invalidate entire data cache. Modified lines are not written back.
CD=1	1	Invalidate line. Writeback if modified.

Note that TR2, TR3, and TR4 permit both reads and writes, whereas TR5 is a write-only register. The test registers should be written to for testability accesses only. Writing to the test registers during normal operation may cause unpredictable behavior. For example, inadvertent cache hits can be created.

Note: During cache testability operations, the internal snooping mechanism functions similar to that described in “Internal Snooping” on page 6-40. If a memory data access occurs during a code cache testability operation using the test registers, the date cache is checked before the external memory operation is initiated. If the access is a miss in the data cache, then the accessed line if valid in the code cache is invalidated through the internal snooping mechanism. In addition, the same cache line fill buffer is used for cache testability writes and to temporarily store data from memory data reads. For this reason, memory data reads should be done with care or avoided to

ensure data from the memory read does not overwrite data from the testability write in the cache line fill buffer.

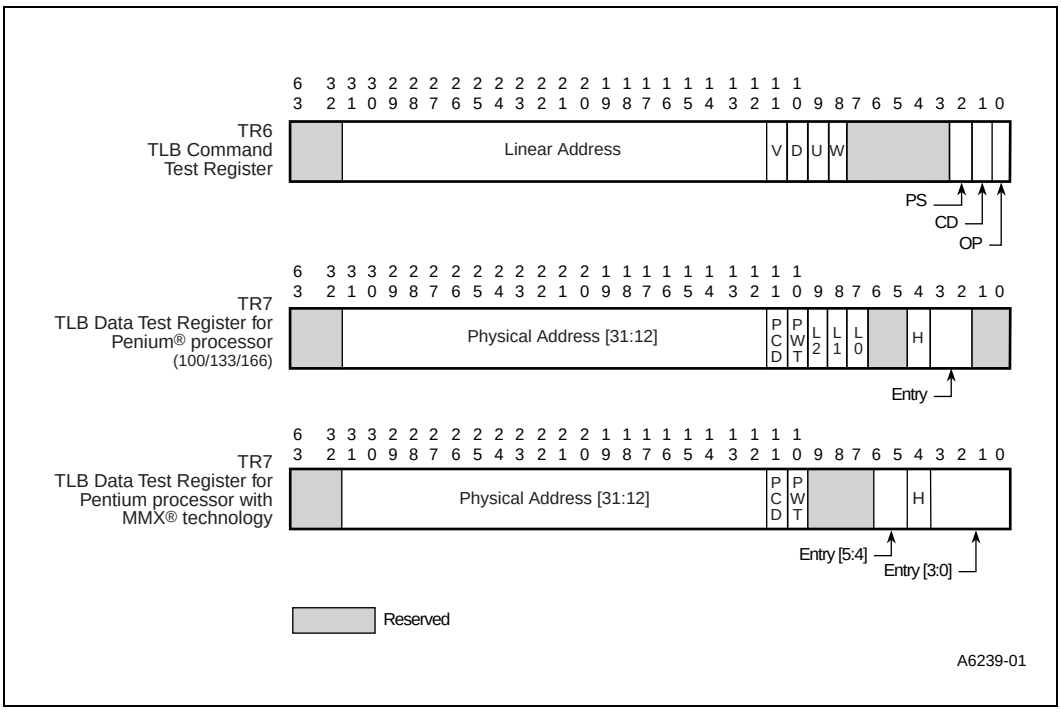
Similarly, if a code access occurs during a data cache testability operation using the test registers, the code cache is checked before the external memory operation is initiated. If the access is a miss in the code cache, then the accessed line if valid in the data cache is invalidated (or written back and then invalidated if in the M-state) through the internal snooping mechanism.

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

14.2.1.2 TLB Test Registers

The registers in Figure 14-2 provide access to the Pentium processor's code and data cache translation lookaside buffers (TLBs). Note that the data cache has two TLBs: a 64-entry TLB for 4-Kbyte data pages and an 8-entry TLB for 4-Mbyte data pages. The code cache contains only one 32-entry TLB for both 4-Kbyte code pages and 4-Mbyte code pages. The 4-Mbyte code pages are cached in 4-Kbyte increments (the PS bit in TR6 is ignored). The code cache contains one fully associative 32-entry TLB which is also integrated for both 4-Kbyte and 4-Mbyte pages. Note that, unlike the embedded Pentium processor, the embedded Pentium processor with MMX technology data cache contains one fully associative 64-entry TLB which is integrated for both 4-Kbyte and 4-Mbyte pages.

Figure 14-2. TLB Test Registers



TR6 is the TLB Command Test Register. It contains the linear address, code/data TLB select (CD), operation (Op) bits and the following status bits: valid (V), dirty (D), user (U), writeable (W), and page size (PS) bits.

The status bits are inputs to the TLB entry during testability writes, and outputs from the TLB entry during testability reads. The V bit indicates whether a TLB entry is valid or invalid during testability writes. The D bit indicates whether or not a write access was made to the page. The U bit indicates the privilege level that the processor must be in to access the page. The W bit is one of the factors in determining the read/write protection of the page. The PS (page size) bit specifies the page size for the TLB entry. The CD bit determines if the code or data TLB is being accessed. The Op bit distinguished between a read and write cycle.

The W-bit, D-bit, and PS-bit are defined only for the data TLB.

Tables 14-9 through 14-16 list the encodings for the fields in the TR6 register.

Table 14-9. Encoding for the Valid Bit in TR6

Valid	Valid/Invalid TLB Entry
0	Invalid
1	Valid

Table 14-10. Encoding for the Dirty Bit in TR6

D-bit	Write access made to page?
0	Write access was not made
1	Write access was made

Table 14-11. Encoding for the User Bit in TR6

U-bit	Privilege Level Access Allowed
0	PL=0,1,2,3
1	PL=0

Table 14-12. Encoding for the Writeable Bit in TR6

W-bit	Writes Allowed?
0	No writes, read only
1	Allows writes

Table 14-13. Encoding for the Page Size Bit in TR6

PS-bit	Page Size
0	4 KByte
1	4 MByte

Note: Normally the user should not allocate a page entry in both the TLBs; during testability however if a match is found in both, then the processor reports that it found it for the 4-Mbyte page size (PS=1).

Table 14-14. Encoding for the Operation Bit TR6

Op	Command
0	TLB write
1	TLB read

Table 14-15. Encoding for the Code/Data TLB in TR6

CD	Cache
0	Code TLB
1	Data TLB

TR7 is the TLB Data Test Register. In the embedded Pentium processor it contains bits 31:12 of the physical address, the hit indicator H, a two-bit entry pointer, and the status bits. The status bits of the Pentium processor include the two paging attribute bits PCD and PWT, and three LRU bits (L0, L1, and L2). PCD is the page level cache disable bit. PWT is the page level write through bit. The LRU bits determine which entry is to be replaced according to the pseudo-LRU algorithm. TLB reads which result in hits and TLB writes can change the LRU bits. The LRU bits reported for a test read are the value before the TLB read. The LRU bits are then changed according to the pseudo-LRU replacement algorithm. The two entry bits determine which one of the four ways to write to in the code or data TLB during testability writes.

In the embedded Pentium processor with MMX technology, the entry pointer has been extended from two bits to six bits. The six entry bits determine which one of the 64 entries to write to in the data TLB during testability writes. The lower five entry bits determine which one of the 32 entries to write to in the code TLB during testability writes. Also, the L0, L1 and L2 bits are reserved in the Pentium processor with MMX technology.

The H is the hit indicator. This bit needs to be set to 1 during testability writes. During testability reads, if the input linear address matches a valid entry in the TLB, the H bit is set to 1. The two entry bits determine in which one of the four ways to write to the TLB during testability writes. During testability reads, they indicate the way that resulted in a read bit.

TR6, and TR7 are read/write registers. The test registers should be written to for testability accesses only. Writing to the test registers during normal operation causes unpredictable behavior.

When reading from the code cache TLB (TR5.CD = 0), the TR6 register zeros out bits [31:12] (corresponding to the linear address) at the end of the TLB testability read cycle. This does not mean that an incorrect linear address was used. All operations happen normally (with whatever linear address was written into TR6 before the testability read operation).

TLB Access

Unlike the caches, the TLB is structured as a CAM cell and, thus, can only be searched (rather than directly read). In other words, the programmer can directly read/write a particular entry in a particular set of the code or data caches, however the TLB only reports a hit or a miss in the Hit bit in TR7. Dumping the TLB requires the programmer to step through the entire linear address space one page at a time. Also, please note the following changes which apply to the Pentium processor with MMX technology:

- LRU bits of TR7 (bits 9:7) are reserved on the embedded Pentium processor with MMX technology.
- The entry pointer in TR7 has been extended from two bits to six bits in the embedded Pentium processor with MMX technology.
- To assure correct functioning, software MUST flush the TLB after testability writes and prior to return to normal operation mode by writing to CR3.
- It is recommended that users do not use testability reads to load the TLB with overlapping 4 Kbyte and 4 Mbyte pages.

To write an entry into the TLB:

- Write the physical address bits [31:12], attribute bits, LRU bits and replacement entry into TR7, setting TR7.H=1.
- Write the linear address, protection bits, and page size bit into TR6, setting TR6.Op=0.

To read an entry from the TLB:

- Write the linear address, CD, and OP bits into TR6, setting TR6.Op=1.
- If TR7.H is set to 1, the read resulted in a hit. Read the translated physical address, attribute bits, and entry from TR7. Read the V, D, U, and W bits from TR6. If TR7.H is cleared to 0, the read was a miss and the physical address is undefined.

Note that when reading from the TLB, the PS bit in the TR6 register does not have to be set; the PS bit is actually written by the processor at the end of the TLB (testability) lookup. Based on the PS bit the user is supposed to infer whether the linear address found in the TLB corresponds to the 4-Kbyte or 4-Mbbyte page size. Normally the user should not allocate a page entry in both the TLBs; during testability however if a match is found in both, then the processor reports that it found it for the 4-Mbyte page size (PS=1).

Also note that when reading from the code cache TLB (TR5.CD=0), the TR6 register zeros out bits 12–31 (corresponding to the linear address) at the end of the TLB testability read cycle. This does not mean that an incorrect linear address was used. All operations happen normally (with whatever linear address was written into TR6 before the testability read operation).

14.2.1.3 Branch Target Buffer (BTB) Test Registers

The test registers in Figure 14-3 provide direct access to the branch target buffer. Note that the branch prediction mechanism should be disabled through test register 12 before doing any BTB testability access.

TR9 is the BTB Tag Test Register. Before writing any entry into the BTB, software must first load TR9 with the appropriate information. After reading any entry in the BTB, the processor places the retrieved information in TR9.

Figure 14-3. BTB Test Registers

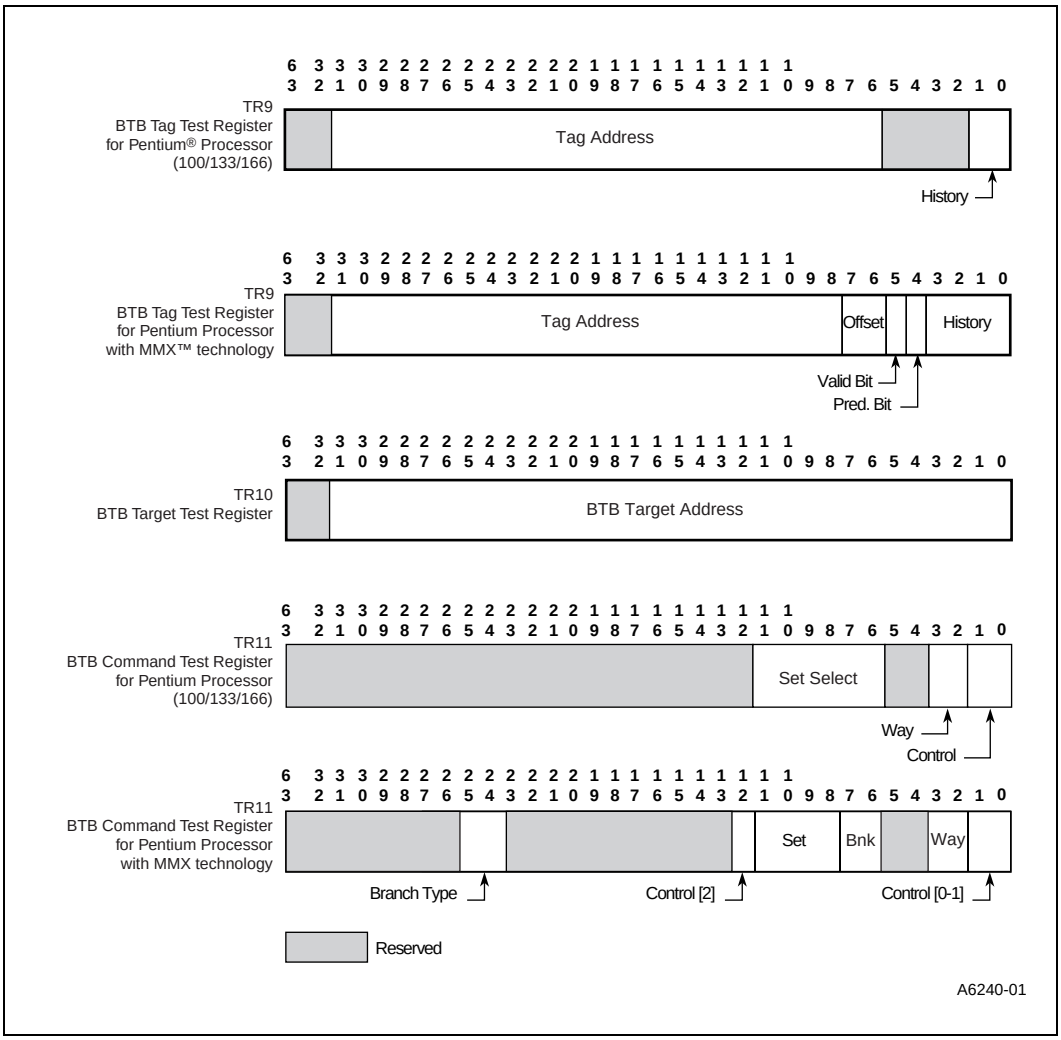


Table 14-16. TR9 Register Description (BTB Test Register)

Bits in the Embedded Pentium® Processor	Bits in the Embedded Pentium Processor with MMX™ Technology	TR9 Register Description (BTB Test Register)
63:32	63:32	Reserved
31:6	31:8	Tag Address: Bits 31:6 or 31:8 of the address of the last byte of the branch
N/A	7:6	Offset: Bits 1:0 of the address of the last byte of the branch
N/A	5	Valid bit: If set, the entry is allocated in the BTB
N/A	4	Prediction bit: Defines if this branch is predicted taken or not taken by the BTB
1:0	3:0	History: Contains the previous history for this branch

Table 14-17. TR10 Register Description (BTB Test Register)

Bits	TR10 Register Description (BTB Target Test Register)
63:32	Reserved
31:0	BTB Target Address: Linear address of the branch's target

Table 14-18. TR11 Register Description (BTB Command Test Register)

Bits in the Embedded Pentium® Processor	Bits in the Embedded Pentium Processor with MMX™ Technology	TR11 Register Description (BTB Command Test Register)
63:32	63:32	Reserved
31:12	31:26	Reserved
N/A	25:24	Branch type: 00 JCC (Jump if condition is met), 01 unconditional jump, 10 call, 11 return
N/A	23:13	Reserved
N/A	12	Control: Selects either Normal operation, or Testability Read/Write, Flush and Testability Read Tag
11:6	11:8	Set: Selects one of 64 sets to access in the embedded Pentium processor or 16 sets in the embedded Pentium processor with MMX technology
N/A	7:6	Bank: Selects one of the 4 banks per BTB cache line. The bank number corresponds to bits 3:2 of the branch address
5:4	5:4	Reserved
3:2	3:2	Way: Selects one of four ways within the Set (i.e., 00 = Way1, 01 = Way2, 10 = Way3 and 11 = Way4)
1:0	1:0	Control: Selects either Normal operation, or Testability Read/Write, Flush and Testability Read Tag

Note: The format for the control field is shown in Table 14-19.

TR10 is the BTB Target Test Register. Like TR9, TR10 must be loaded with the target address before a testability write. After a BTB testability read, the target address is placed in this register.

TR11 is the BTB Command Test Register. This register is used to issue read and write commands to the BTB. The set address field selects one of 16 sets (64 sets in the embedded Pentium processor) to access. The entry field selects one of four ways within the set on the embedded Pentium processor. A BTB testability cycle is initiated by loading TR11 controls bits with the appropriate values. The format for the control field is shown in Table 14-19.

Table 14-19. Format for TR11 Control Field

Cntl2[1]	Cntl1	Cntl0	Command
0	0	0	Normal operation
0	0	1	Testability write data
0	1	0	Testability read data
0	1	1	Testability BTB flush
1	0	1	Testability read TAG[2]

NOTES:
1. Applies to the embedded Pentium processor with MMX technology only.
2. Other combinations are reserved.

TR9, TR10 and TR11 are all read/write registers. The test registers should be written to for testability accesses only. Writing to the test registers during normal operation causes unpredictable behavior.

The following BTB testability cycles exist:

1. Testability read data. Reads the Target, branch type, offset, history-prediction (according to spec bit)., and prediction bit of a BTB line defined by a set, way and bank into the corresponding testability register field.
2. Testability read TAG and valid bit (Pentium processor with MMX technology only). Reads the Tag defined by the testability registers set, way and bank into the corresponding testability register field.
3. Testability BTB flush. Clear all BTB valid bits.
4. Testability Write Data. Writes all the BTB fields from the corresponding test registers. If there is an entry on the same bank and set, with the same TAG, the write overwrites this entry even if the way chosen in TR11 is different from the existing entry's way. (This is done to avoid having two entries in the same bank and same set, but different ways, with the same TAG.)

TR9, TR10, TR11 are all read/write registers. The test registers should be written to for testability accesses only. Writing to the test registers during normal operation causes unpredictable behavior.

Direct BTB Access

The BTB contents are directly accessible, in a manner similar to the code/data caches. Note that the branch prediction mechanism should be disabled before doing any BTB testability access.

To write an entry into the BTB for the embedded Pentium processor:

1. Disable BTB entry allocation by setting TR12.NBP=1 (see Feature Control section)
2. Write the tag address and history information in TR9

3. Write the target address in TR10
4. Write the appropriate set address, entry fields and control bits in TR11.

To write an entry into the BTB for the embedded Pentium processor with MMX technology:

1. Disable BTB entry allocation by setting TR12.NBP=1 (see Feature Control section)
2. Write the tag address and history, offset, valid and prediction information in TR9
3. Write the target address in TR10
4. Write the appropriate set address and entry fields, way, bank, branch type and control bits in TR11.

To read an entry from the BTB for the embedded Pentium processor:

1. Perform a testability read by writing to TR11 with the appropriate set address entry fields.
2. Read the tag address and history information from TR9.
3. Read the target address from TR10.

To read an entry from the BTB for the embedded Pentium processor with MMX technology:

1. Disable BTB entry allocation by setting TR12.NBP=1 (see Feature Control section)
2. Perform a testability read by writing to TR11 with the appropriate set address and entry fields, way, bank and control bits.
3. Read the tag address, history information, offset, prediction and valid bits from TR9.
4. Read the target address from TR10.
5. Read the branch type from TR11.
6. Perform a testability read tag by writing to TR11 with the appropriate set address, way, bank and control bits.
7. Read the branch tag from TR9

Note: Read Tag and Read data does not destroy the other's cycle fields in TR9. This means that the read from TR9 can be done only once after both cycles were executed.

14.2.1.4 Parity Reversal Register (TR1)

A model specific register, TR1, the Parity Reversal Register (PRR), allows the parity check mechanism to be tested. Figure 14-4 shows the format of the PRR.

Figure 14-4. Parity Reversal Register

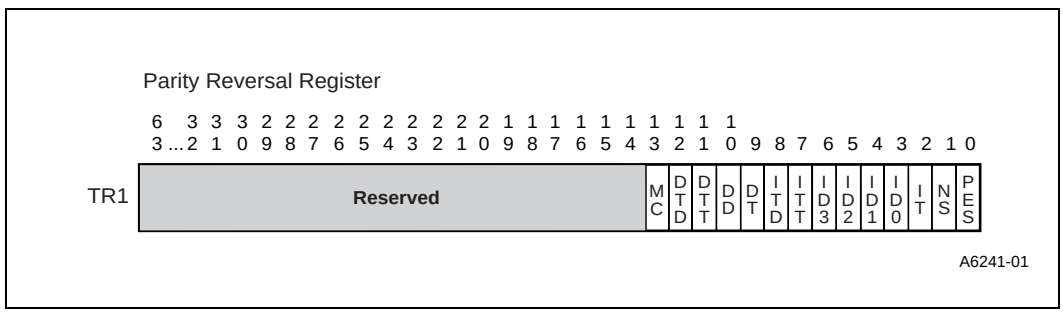


Table 14-20 lists each of the bits in the parity reversal register and their function.

Table 14-20. Parity Reversal Register Bit Definition

Bit Name	Description
PES	Parity Error Summary, set on any parity error
NS	0 = set PRR.PES, assert IERR#, and shutdown on parity error 1 = set PRR.PES, and assert IERR# on parity error
IT	code (instruction) cache tag
ID0	code cache data even bits 126, 124... 2,0
ID1	code cache data odd bits 127, 125... 3,1
ID2	code cache data even bits 254, 252... 130,128
ID3	code cache data odd bits 255, 253... 131, 129
ITT	code TLB tag
ITD	code TLB data
DT	data cache tag
DD	data cache data, use byte writes for individual access
DTT	data TLB tag
DTD	data TLB data
MC	microcode, reverse parity on read

Writing a one into bits 2-12 reverses the sense of the parity generation for any write into the corresponding array. This includes both normal cache replacements as well as testability writes and data writes. Parity is checked during both normal reads and testability reads.

To test parity error detection, software should write a one into the appropriate bit of the parity reversal register (PRR), perform a testability write into the array, and then perform a testability read. Upon successful detection of the parity error, the Pentium processor asserts the IERR# pin and may shutdown. Alternatively, after writing a one into the appropriate bit of the PRR, software may perform a normal write and read of the array by creating a cache miss and doing a read.

As an option, software may mask the shutdown by setting PRR.NS to 1 if the system is unable to recover from a shutdown. To determine if a parity error has occurred, software may read the parity error summary bit, PRR.PES. Hardware sets this bit on any parity error, and it remains set until cleared by software.

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 if the bit was already set.

Bit 0 of TR1 is read/write. The remaining bits are write only. The test registers should be written to for testability accesses only. Writing to the test registers during normal operation causes unpredictable behavior.

14.3 New Feature Control (TR12)

The new features of branch prediction, execution tracing, and instruction pairing in the Pentium processor can be selectively enabled or disabled through individual bits in test register TR12 (Figure 14-5). The branch prediction, execution tracing, and instruction pairing features of the Pentium processor family can be selectively enabled or disabled through individual bits in test register TR12. In addition, level 1 caching can be disabled without affecting the PCD output to allow testing of a second level cache.

Figure 14-5. Test Register (TR12)

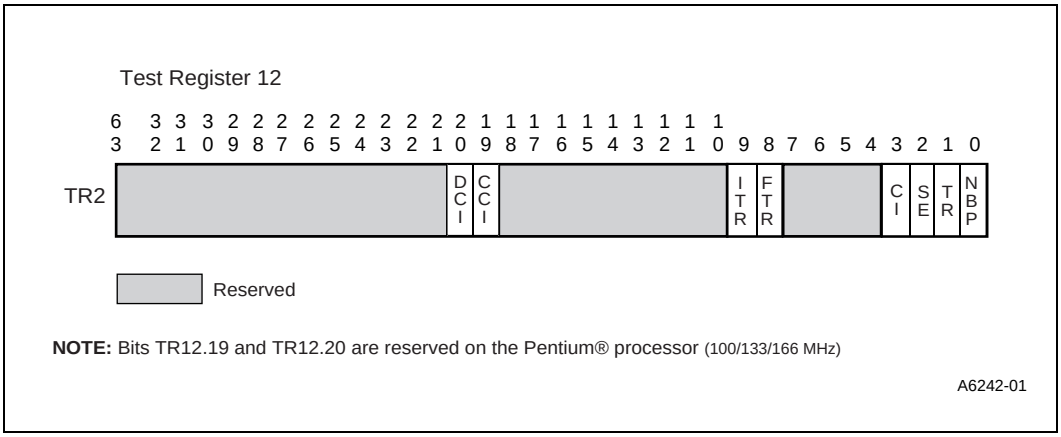


Table 14-21. New Feature Controls

Name	Position	Function
NBP	0	No Branch Prediction controls the allocation of new entries in the BTB. When TR12.NBP is clear, the code cache allocates entries in the BTB. When TR12.NBP is set, no new entry is allocated in the BTB, however, entries already in the BTB may continue to cause a BTB hit and result in the pipeline being reloaded from the predicted branch target. To completely disable branch prediction, first set TR12.NBP to 1 and then flush the entire BTB by loading CR3.
TR	1	Execution Tracing controls the Branch Trace message Special Cycle. When the TR12.TR bit is set to 1, a branch trace message special cycle is generated whenever a taken branch is executed. Two cycles are produced: one for the linear address of the instruction causing the taken branch, and one for the branch target linear address.
SE	2	Single Pipe Execution controls instruction pairing. When TR12.SE is cleared to zero, instructions are issued to both the u and v pipes contingent on pairing restrictions. When TR12.SE is set to one, the v pipe is disabled and instructions are issued only to the u pipe. Microcoded instructions are designed to utilize both pipes concurrently, independent of the state of TR12.SE. Note that all instructions requiring microcode are not pairable.
CI	3	Cache Inhibit controls line fill behavior. When TR12.CI is reset to 0, the on-chip data and instruction caches operate normally. When TR12.CI is set to 1, all cache line fills are inhibited and all bus cycles due to cache misses are run as single transfer cycles (CACHE# is not asserted). Unlike CR0.CD, TR12.CI does not affect the state of the PCD output pin. This allows the first level cache to be disabled while the second level cache is still active and can be tested. Note that the contents of the instruction and data caches are not affected by the state of TR12.CI, e.g., they are not flushed. The second level cache test sequence should be: set TR12.CI to 1, flush the internal caches, run the second level cache tests.
	4-7	Reserved
FTR	8	Fast Execution Tracing is similar to Execution Tracing (TR12.TR). If TR12.FTR is set to 1 while execution tracing is enabled (TR12.TR = 1), only one branch trace message special cycle is produced containing the linear address of the instruction causing the taken branch.
ITR	9	IO Trap Restart enables proper interrupt prioritization to support restarting IO accesses trapped by System Management Mode.
	10–18	Reserved
CCI	19†	Code Cache Inhibit is the same instruction as Cache Inhibit (CI), but only applies to the code cache.
DCI	20†	Data Cache Inhibit is the same instruction as Cache Inhibit (CI), but only applies to the data cache.
	21–63	Reserved

† These bits are reserved on the Pentium processor (100/133/166).

TR12.NBP, TR12.TR, TR12.SE, and TR12.CI are initialized to zero on reset. This register is write only and the reserved bits should be written with zeros.

14.4 Performance Monitoring

The processor includes features to measure and monitor various parameters that contribute to the performance of the processor. This information can be then used for compiler and memory system tuning. For memory system tuning, it is possible to measure data and instruction cache hit rates, and time spent waiting for the external bus. The performance monitor allows compiler writers to gauge the effectiveness of instruction scheduling algorithms by measuring address generation interlocks and parallelism.

While the performance monitoring features that are provided by the Pentium processor are generally model specific and available only to privileged software, the Pentium processor also provides an *architectural* Time Stamp Counter that is available to the user. With this notable exception, the performance monitor features and the events they monitor are otherwise implementation dependent, and consequently, they are not considered part of the Pentium processor *architecture*. The performance monitor features are expected to change in future implementations.

Note: It is essential that software abide by the usage restrictions for accessing model specific registers as discussed in section “Model Specific Register Usage Restrictions” on page 14-1.

14.4.1 Performance Monitoring Feature Overview

Processor performance monitoring features include:

Table 14-22. Architectural Performance Monitoring Features

RDTSC	Read Time Stamp Counter - a user level instruction to provide read access to a 64-bit free-running counter
RDPMC	Read Performance Monitoring Counter - this instruction enables reading of the performance monitoring counters (in CPL = 3) provided bit 8 of CR4 (CR4.PCE) is set. Note: The RDPMC instruction is only defined on the embedded Pentium processor with MMX technology. Execution of the RDPMC instruction in a embedded Pentium processor will result in an invalid opcode exception.
CPUID (EDX.TSC)	Time Stamp Counter Feature Bit - Bit 4 of EDX is set to 1 to indicate that the processor implements the TSC and RDTSC instruction
CR4.TSD	Time Stamp Disable - A method for a supervisor program to disable user access to the time stamp counter in secure systems. When bit 2 of CR4 is set to 1, an attempt to execute the RDTSC instruction generates an general protection exception (#GP).

Table 14-23. Model Specific Performance Monitoring Features

CTR0, CTR1	Counter 0, Counter 1 - two programmable counters
CESR	Control and Event Select Register - programs CTR0, CTR1
TSC	Time Stamp Counter - provides read and write access to the architectural 64-bit counter in a manner that is model specific.
PM0/BP0, PM1/BP1	Event Monitoring Pins - These pins allow external hardware to monitor the activity in CTR0 and CTR1.

14.4.2 Time Stamp Counter (TSC)

A dedicated, free-running, 64-bit time stamp counter is provided on chip. Note that on the Pentium processor, this counter increments on every clock cycle, although it is not guaranteed that this will be true on future processors. As a time stamp counter, the RDTSC instruction reports values that are guaranteed to be unique and monotonically increasing. Portable software should not expect that the counter reports absolute time or clock counts. The user level RDTSC (Read Time Stamp Counter) instruction is provided to allow a program of any privilege level to sample its value. A bit in CR4, TSD (Time Stamp Disable) is provided to disable this instruction in secure environments. Supervisor mode programs may sample this counter using the RDMSR instruction or reset/preset this counter with a WRMSR instruction. The counter is cleared after reset.

While the user level RDTSC instruction and a corresponding 64-bit time stamp counter will be provided in all future Pentium processor compatible processors, access to this counter via the RDMSR/WRMSR instructions is dependent upon the particular implementation.

14.4.3 Programmable Event Counters (CTR0, CTR1)

Two programmable 40-bit counters CTR0 and CTR1 are provided. The implementation of these two counters is slightly different between the embedded Pentium processor with MMX technology and the embedded Pentium processor. In the embedded Pentium processor each counter may be programmed to count any event from a pre-determined list of events. These events, which are described in the *Events* section of this chapter, are selected by programming the Control and Event Select Register (CESR). In the embedded Pentium processor with MMX technology some additional events were added and cannot be assigned to either of the two counters independently. These new events are paired, so when one event is assigned to counter 0, a second related event is automatically assigned to counter 1. The counters are not affected by writes to CESR and must be cleared or pre-set when switching to a new event. The counters are undefined after RESET.

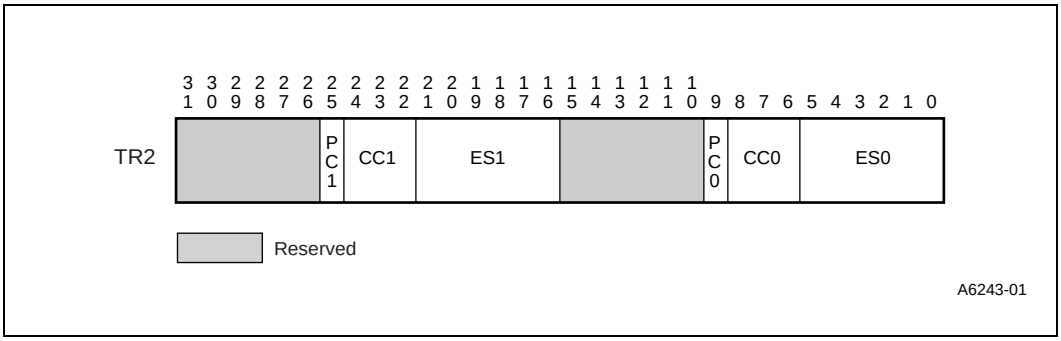
Associated with each counter is an event pin (PM1/BP1, PM0/BP0) which externally signals the occurrence of the selected event.

Note that neither the CTR0/CTR1 nor CESR are part of the processor state that is automatically saved and restored during a context switch. If it is desired to coordinate the use of the programmable counters in a multiprocessing system, it is the software's responsibility to share or restrict the use of these counters through a semaphore or other appropriate mechanism.

14.4.4 Control and Event Select Register (CESR)

A 32-bit Control and Event Select Register (CESR) is used to control operation of the programmable counters and their associated pins. Figure 14-6 depicts the CESR. For each counter, the CESR contains a 6-bit Event Select field (ES), a Pin Control bit (PC), and a three bit control field (CC). It is not possible to selectively write a subset of the CESR. If only one event needs to be changed, the CESR must first be read, the appropriate bits modified, and all bits must be written back. At reset, all bits in the Control and Event Select Register are cleared.

Figure 14-6. Control and Event Select Register



14.4.4.1 Event Select (ES0, ES1)

Up to two events may be monitored by placing the appropriate event code in the Event Select field. The events and codes are listed in the Events section of this chapter.

14.4.4.2 Counter Control (CC0, CC1)

A three bit field is used to control the operation of the counter. the highest order bit selects between counting events and counting clocks. The middle bit enables counting when the CPL=3. The low order bit enables counting when the CPL=0, 1 or 2.

CC	Meaning
000	Count Nothing (Disable Counter)
001	Count the selected Event while the CPL=0, 1 or 2
010	Count the selected Event while the CPL=3
011	Count the selected Event regardless of the CPL
100	Count Nothing (Disable Counter)
101	Count Clocks while the CPL=0, 1 or 2
110	Count Clocks while the CPL=3
111	Count Clocks regardless of the CPL

While a counter need not be stopped to sample its contents, it must be stopped and cleared or preset before switching to a new event.

14.4.4.3 Pin Control (PC0, PC1)

Associated with CTR0 and CTR1 are two pins, PM0 and PM1 (PM0/BP0, PM1/BP1), and two bits which control their operation, PC0 and PC1. These pins may be programmed by the PC0/PC1 bits in the CESR to indicate either that the associated counter has incremented or that it has overflowed. Note that the external signalling of the event on the pins will lag the internal event by a "few" clocks as the signals are latched and buffered.

PC	PM pin signals when the corresponding counter:
0	has incremented
1	has overflowed

When the pins are configured to signal that a counter has incremented, it should be noted that although the counters may increment by 1 or 2 in a single clock, the pins can only indicate that the event occurred. Moreover, since the internal clock frequency may be higher than the external clock frequency, a single external clock may correspond to multiple internal clocks.

A "count up to" function may be provided when the event pin is programmed to signal an overflow of the counter. Because the counters are 40 bits, a carry out of bit 39 indicates an overflow. A counter may be preset to a specific value less than $2^{40} - 1$. After the counter has been enabled and the prescribed number of events has transpired, the counter will overflow. Approximately 5 clocks later, the overflow is indicated externally and appropriate action, such as signaling an interrupt, may then be taken.

When the performance monitor pins are configured to indicate when the performance monitor counter has incremented and an "occurrence event" is being counted, the associated PM pin is asserted (high) each time the event occurs. When a "duration event" is being counted the associated PM pin is asserted for the entire duration of the event. When the performance monitor pins are configured to indicate when the counter has overflowed, the associated PM pin is not asserted until the counter has overflowed.

The PM0/BP0, PM1/BP1 pins also serve to indicate breakpoint matches during in Circuit Emulation, during which time the counter increment or overflow function of these pins is not available. After RESET, the PM0/BP0, PM1/BP1 pins are configured for performance monitoring, however a hardware debugger may re-configure these pins to indicate breakpoint matches.

14.4.5 Performance Monitoring Events

Events may be considered to be of two types: those that count OCCURRENCES, and those that count DURATION. Each of the events listed below is classified accordingly.

Occurrences events are counted each time the event takes place. If the PM0 or PM1 pins are configured to indicate when a counter increments, they are asserted each clock the counter increments. Note that if an event can happen twice in one clock the counter increments by 2, however the PM0/1 pins are asserted only once.

For Duration events, the counter counts the total number of clocks that the condition is true. When configured to indicate when a counter increments, the PM0 and PM1 pins are asserted for the duration of the event.

Table 14-24 lists the events that can be counted, and their encodings for the Control and Event Select Register.

The performance monitoring features present in the embedded Pentium processor have been extended in the embedded Pentium processor with MMX technology. The event list is longer, and there is a new instruction defined to facilitate use of the instruction monitoring. To leave room for future additions all new embedded Pentium processor with MMX technology events are assigned to just one of the two events counters (CTR0, CTR1). It is not possible to assign these events to any of the two counters at will. “Twin events” (such as “D1 starvation and FIFO is empty”) are assigned to different counters to allow their concurrent measurement.

The Read Performance Monitoring Counter (RDPMC) is implemented in the embedded Pentium processor with MMX technology. See the *Intel Architecture Software Developer’s Manual* for more information about the RDPMC instruction.

Table 14-24. Performance Monitoring Events (Sheet 1 of 4)

Decimal Encoding	Binary Encoding	Counter 0	Counter 1	Performance Monitoring Event	Occurrence or Duration?
0	000000	Yes	Yes	Data Read	Occurrence
1	000001	Yes	Yes	Data Write	Occurrence
2	000010	Yes	Yes	Data TLB Miss	Occurrence
3	000011	Yes	Yes	Data Read Miss	Occurrence
4	000100	Yes	Yes	Data Write Miss	Occurrence
5	000101	Yes	Yes	Write (hit) to M- or E-state lines	Occurrence
6	000110	Yes	Yes	Data Cache Lines Written Back	Occurrence
7	000111	Yes	Yes	External Snoops	Occurrence
8	001000	Yes	Yes	External Data Cache Snoop Hits	Occurrence
9	001001	Yes	Yes	Memory Accesses in Both Pipes	Occurrence
10	001010	Yes	Yes	Bank Conflicts	Occurrence

NOTE: Shaded areas only apply to the embedded Pentium® processor with MMX™ technology.

Table 14-24. Performance Monitoring Events (Sheet 2 of 4)

Decimal Encoding	Binary Encoding	Counter 0	Counter 1	Performance Monitoring Event	Occurrence or Duration?
11	001011	Yes	Yes	Misaligned Data memory or I/O References	Occurrence
12	001100	Yes	Yes	Code Read	Occurrence
13	001101	Yes	Yes	Code TLB Miss	Occurrence
14	001110	Yes	Yes	Code Cache Miss	Occurrence
15	001111	Yes	Yes	Any Segment Register Loaded	Occurrence
16	010000	Yes	Yes	Reserved	
17	010001	Yes	Yes	Reserved	
18	010010	Yes	Yes	Branches	Occurrence
19	010011	Yes	Yes	BTB Hits	Occurrence
20	010100	Yes	Yes	Taken Branch or BTB hit	Occurrence
21	010101	Yes	Yes	Pipeline Flushes	Occurrence
22	010110	Yes	Yes	Instructions Executed	Occurrence
23	010111	Yes	Yes	Instructions Executed in the v pipe e.g. parallelism/pairing	Occurrence
24	011000	Yes	Yes	Clocks while a bus cycle is in progress (bus utilization)	Duration
25	011001	Yes	Yes	Number of clocks stalled due to full write buffers	Duration
26	011010	Yes	Yes	Pipeline stalled waiting for data memory read	Duration
27	011011	Yes	Yes	Stall on write to an E- or M-state line	Duration
28	011100	Yes	Yes	Locked Bus Cycle	Occurrence
29	011101	Yes	Yes	I/O Read or Write Cycle	Occurrence
30	011110	Yes	Yes	Non-Cacheable memory reads	Occurrence
31	011111	Yes	Yes	Pipeline stalled because of an address generation interlock	Duration
32	100000	Yes	Yes	Reserved	
33	100001	Yes	Yes	Reserved	
34	100010	Yes	Yes	FLOPs	Occurrence
35	100011	Yes	Yes	Breakpoint match on DR10 Register	Occurrence
36	100100	Yes	Yes	Breakpoint match on DR1 Register	Occurrence
37	100101	Yes	Yes	Breakpoint match on DR2 Register	Occurrence
38	100110	Yes	Yes	Breakpoint match on DR3 Register	Occurrence
39	100111	Yes	Yes	Hardware Interrupts	Occurrence
40	101000	Yes	Yes	Data Read or Data Write	Occurrence
41	101001	Yes	Yes	Data Read Miss or Data Write Miss	Occurrence

NOTE: Shaded areas only apply to the embedded Pentium® processor with MMX™ technology.

Table 14-24. Performance Monitoring Events (Sheet 3 of 4)

Decimal Encoding	Binary Encoding	Counter 0	Counter 1	Performance Monitoring Event	Occurrence or Duration?
42	101010	Yes	No	Bus Ownership Latency	Duration
42	101010	No	Yes	Bus Ownership Transfers	Occurrence
43	101011	Yes	No	MMX instructions executed in u pipe	Occurrence
43	101011	No	Yes	MMX instructions executed in v pipe	Occurrence
44	101100	Yes	No	Cache M-Sate line Sharing	Occurrence
44	101100	No	Yes	Cache Line Sharing	Occurrence
45	101101	Yes	No	EMMS instructions executed	Occurrence
45	101101	No	yes	Transition between MMX™ instructions and FP instructions	Occurrence
46	101110	Yes	No	Bus Utilization Due to processor Activity	Duration
46	101110	No	Yes	Writes to Non-Cacheable Memory	Occurrence
47	101111	Yes	No	Saturating MMX instructions executed	Occurrence
47	101111	No	Yes	Saturations performed	Occurrence
48	110000	Yes	No	Number of Cycles Not in HLT State	Duration
48	110000	No	Yes	Number of Cycles Not in HLT State	Duration
49	110001	Yes	No	MMX instruction data reads	Occurrence
49	110001	No	Yes	MMX instructions data read misses	Occurrence
50	110010	Yes	No	Floating Point Stalls	Duration
50	110010	No	Yes	Taken Branches	Occurrence
51	110011	Yes	No	D 1 Starvation and FIFO is empty	Occurrence
51	110011	No	Yes	D1 Starvation and only one instruction in FIFO	Occurrence
52	110100	Yes	No	MMX instruction data writes	Occurrence
52	110100	No	Yes	MMX instruction data write misses	Occurrence
53	110101	Yes	No	Pipeline flushed due to wrong branch prediction	Occurrence
53	110101	No	Yes	Pipeline flushes due to wrong branch predictions resolved in WB-stage	Occurrence
54	110110	Yes	No	Misaligned data memory references on MMX instruction	Occurrence
54	110110	No	Yes	Pipeline stalled waiting for MMX instruction data memory read	Duration
55	110111	Yes	No	Returns Predicted Incorrectly or not predicted at all	Occurrence
55	110111	No	Yes	Returns Predicted (Correctly and Incorrectly)	Occurrence
56	111000	Yes	No	MMX multiply unit interlock	Duration

NOTE: Shaded areas only apply to the embedded Pentium® processor with MMX™ technology.

Table 14-24. Performance Monitoring Events (Sheet 4 of 4)

Decimal Encoding	Binary Encoding	Counter 0	Counter 1	Performance Monitoring Event	Occurrence or Duration?
56	111000	No	Yes	MOVD/MOVQ state stall due to previous operation	Duration
57	111001	Yes	No	Returns	Occurrence
57	111001	No	Yes	Reserved	
58	111010	Yes	No	BTB false entries	Occurrence
58	111010	No	Yes	BTB miss prediction on a Not-Taken branch	Occurrence
59	111011	Yes	No	Number of clocks stalled due to full write buffers while executing MMX instructions	Duration
59	111011	No	Yes	Stall on MMX instruction write to E- or M-state line	Duration

NOTE: Shaded areas only apply to the embedded Pentium® processor with MMX™ technology.

14.4.6 Description of Events

The following descriptions clarify the events. The event codes are provided in parenthesis.

Data Read (0, 000000), Data Write (1, 000001), Data Read or Data Write (40, 101000):

These are memory data reads and/or writes (internal data cache hit and miss combined), I/O is not included. The individual component reads and writes for split cycles are counted individually. Data Memory Reads that are part of TLB miss processing are not included. These events may occur at a maximum of two per clock.

Data TLB Miss (2, 000010):

This event counts the number of misses to the data cache translation look-aside buffer.

Data Read Miss (3, 000011), Data Write Miss (4, 000100), Data Read Miss or Data Write Miss (41, 101001):

These are memory read and/or write accesses that miss the internal data cache whether or not the access is cacheable or non-cacheable. Additional reads to the same cache line after the first BRDY# of the burst linefill is returned but before the final (fourth) BRDY# has been returned, will not cause the Data Read Miss counter to be incremented additional times. Data accesses that are part of TLB miss processing are not included. Accesses directed to I/O space are not included.

Write (hit) to M- or E-state lines (5, 000101):

This measures the number of write hits to exclusive or modified lines in the data cache. (These are the writes which may be held up if EWBE# is inactive.) This event may occur at a maximum of two per clock.

Data Cache Lines Written Back (6, 000110):

This counts ALL Dirty lines that are written back, regardless of the cause. Replacements and internal and external snoops can all cause writeback and are counted.

External Snoops (7, 000111), Data Cache Snoop Hits (8, 001000):

The first event counts accepted external snoops whether they hit in the code cache or data cache or neither. Assertions of EADS# outside of the sampling interval are not counted. No internal snoops are counted. The second event applies to the data cache only. Snoop hits to a valid line in either the data cache, the data line fill buffer, or one of the write back buffers are all counted as hits.

Memory Accesses in Both Pipes (9, 001001):

Data memory reads or writes which are paired in the pipeline. Note that these accesses are not necessarily run in parallel due to cache misses, bank conflicts, etc.

Bank Conflicts (10, 001010):

These are the number of actual bank conflicts.

Misaligned Data Memory or I/O References (11, 001011):

Memory or I/O reads or writes that are misaligned. A two or four byte access is misaligned when it crosses a four byte boundary; an eight byte access is misaligned when it crosses an eight byte boundary. Ten byte accesses are treated as two separate accesses of eight and two bytes each.

Code Read (12, 001100), Code TLB Miss (13, 001101), Code Cache Miss (14, 001110):

Total instruction reads and reads that miss the code TLB or miss the internal code cache whether or not the read is cacheable or non-cacheable. Individual eight byte non-cacheable instruction reads are counted.

Any Segment Register Loaded (15, 001111):

Writes into any segment register in real or protected mode including the LDTR, GDTR, IDTR, and TR. Segment loads are caused by explicit segment register load instructions, far control transfers, and task switches. Far control transfers and task switches causing a privilege level change will signal this event twice. Note that interrupts and exceptions may initiate a far control transfer.

Branches (18, 010010):

In addition to taken conditional branches, jumps, calls, returns, software interrupts, and interrupt returns, the Pentium processor treats the following operations as causing taken branches: serializing instructions, VERR and VERW instructions, some segment descriptor loads, hardware interrupts (including FLUSH#), and programmatic exceptions that invoke a trap or fault handler. Both Taken and Not Taken Branches are counted. The pipe is not necessarily flushed. The number of branches actually executed is measured, not the number of predicted branches.

BTB Hits (19, 010011):

Hits are counted only for those instructions that are actually executed.

Taken Branch or BTB Hit (20, 010100):

This is a logical OR of taken branches and BTB hits (defined above). It represents an event that may cause a hit in the BTB. Specifically, it is either a candidate for a space in the BTB, or it is already in the BTB.

Pipeline Flushes (21, 010101):

BTB Misses on taken branches, mis-predictions, exceptions, interrupts, and some segment descriptor loads all cause pipeline flushes. This event counter will not be incremented for serializing instructions (serializing instructions cause the prefetch queue to be flushed but will not trigger the Pipeline Flushed event counter) and software interrups (software interrupts do not flush the pipeline).

Instructions Executed (22, 010110):

Up to two per clock. Invocations of a fault handler are considered instructions. All hardware and software interrupts and exceptions will also cause the count to be incremented. Repeat prefixed string instructions will only increment this counter once despite the fact that the repeat loop executes the same instruction multiple times until the loop criteria is satisfied. This applies to all the Repeat string instruction prefixes (i.e., REP, REPE, REPZ, REPNE, and REPNZ). This counter will also only increment once per each HLT instruction executed regardless of how many cycles the processor remains in the HALT state.

Instructions Executed in the v pipe e.g. parallelism/pairing (23, 010111):

Same as the Instructions executed counter except it only counts the number of instructions actually executed in the v pipe. It indicates the number of instructions that were paired.

Clocks while a bus is in progress (bus utilization) (24, 011000):

Including HLDA, AHOLD, BOFF# clocks.

Number of clocks stalled due to full write buffers (25, 011001):

This event counts the number of clocks that the internal pipeline is stalled due to full write buffers. Full write buffers stall data memory read misses, data memory write misses, and data memory write hits to S state lines. Stalls on I/O accesses are not included.

Pipeline stalled waiting for data memory read (26, 011010):

Data TLB Miss processing is also included. The pipeline stalls while a data memory read is in progress including attempts to read that are not bypassed while a line is being filled.

Locked Bus Cycle (28, 011100):

LOCK prefix or LOCK instruction, Page Table Updates, and Descriptor Table Updates. Only the Read portion of the Locked Read-Modify-Write is counted. Split Locked cycles (SCYC active) count as two separate accesses. Cycles restarted due to BOFF# are not recounted.

I/O Read or Write Cycle (29, 011101):

Bus cycles directed to I/O space. Misaligned I/O accesses will generate two bus cycles. Bus cycles restarted due to BOFF# are not re-counted.

Non-cacheable memory reads (30, 011110):

Non-cacheable instruction or data memory read bus cycles. Includes read cycles caused by TLB misses; does not include read cycles to I/O space. Cycles restarted due to BOFF# are not re-counted.

Pipeline stalled because of an address generation interlock (31, 011111):

Number of address generation interlocks (AGIs). An AGI occurring in both the u- and v- pipelines in the same clock signals this event twice. An AGI occurs when the instruction in the execute stage of either of u- or v-pipelines is writing to either the index or base address register of an instruction in the D2 (address generation) stage of either the u- or v- pipelines.

FLOPs (34, 100010);

Number of floating point adds, subtracts, multiplies, divides, remainders, and square roots. The transcendental instructions consist of multiple adds and multiplies and will signal this event multiple times. Instructions generating the divide by zero, negative square root, special operand, or stack exceptions will not be counted. Instructions generating all other floating point exceptions will be counted. The integer multiply instructions and other instructions which use the floating-point arithmetic circuitry will be counted.

Breakpoint match on DR0 Register (35, 100011),

Breakpoint match on DR1 Register (36, 100100),

Breakpoint match on DR2 Register (37, 100101),

Breakpoint match on DR3 Register (38, 100110):

If programmed for one of these breakpoint match events, the performance monitor counters will be incremented in the event of a breakpoint match whether or not breakpoints are enabled. However, if breakpoints are not enabled, code breakpoint matches will not be checked for instructions executed in the v-pipe and will not cause this counter to be incremented (they are checked on instruction executed in the u-pipe only when breakpoints are not enabled). These events correspond to the signals driven on the BP[3:0] pins. Please refer to the Debugging chapter of this volume for more information.

Hardware Interrupts (39, 100111):

Number of taken INTR and NMI only.

Bus ownership latency (42, 101010/0), Bus ownership transfers (42, 101010/1):

The first event measures the time from LRM bus ownership request to bus ownership granted, the time from the earlier of PBREQ (0), PHITM# or HITM# to PBGNT. The second event is count of the number of PBREQ (0). The ratio of these two events is the average stall time due to bus ownership conflict.

MMX instructions executed in U pipe (43, 101011/0):

Total number of MMX instructions executed in U-pipe.

MMX instructions executed in V pipe (43, 101011/1):

Total number of MMX instructions executed in V-pipe.

Cache M-state line sharing (44, 101100/0):

Counts the number of times a processor identified a hit to a modified line due to a memory access in the other processor (PHITM (O)). If the average memory latencies of the system are known, this event enables the user to count the **Write Backs on PHITM(O)** penalty and the **Latency on Hit Modified(I)** penalty.

Cache line sharing (44, 101100/1):

Counts the number of shared data lines in the L1 cache (PHIT (O)).

EMMS instructions executed (45, 101101/0):

Counts number of EMMS instructions executed.

Transition between MMX instructions and FP instructions (45, 101101/1):

Counts first floating point instruction following any MMX instruction or first MMX instruction following a floating point instruction. May be used to estimate the penalty in transitions between FP state and MMX state. An even count indicates the processor is in MMX state. an odd count indicates it is in FP state.

Bus utilization due to processor activity (46, 101110/0):

Counts the number of clocks the bus is busy due to the processor's own activity, i.e., the bus activity which is caused by the processor.

Writes to non-cacheable memory (46, 101110/1):

Counts the number of write accesses to non-cacheable memory. It includes write cycles caused by TLB misses and I/O write cycles. Cycles restarted due to BOFF# are not recounted.

Saturating MMX instructions executed (47, 101111/0):

Counts saturating MMX instructions executed, independently of whether or not they actually saturated. Saturating MMX instructions may perform either add, subtract or pack operations.

Saturations performed (47, 101111/1):

Counts number of MMX instructions that used saturating arithmetic and that at least one of its results actually saturated; i.e., if an MMX instruction operating on four dwords saturated in three out of the four results, the counter will be incremented by one only.

Number of cycles not in HLT state (48, 110000/0):

This event counts the number of cycles the processor is not idle due to HLT instruction. This event will enable the user to calculate "net CPI". Note that during the time that the processor is executing the HLT instruction, the Time Stamp Counter is not disabled. Since this event is controlled by the Counter Controls CC0, CC1 it can be used to calculate the CPI at CPL=3 which the TSC cannot provide.

Clocks stalled on Data cache TLB miss (48, 110000/1):

Counts the number of clocks the pipeline is stalled due to a data cache translation look-aside buffer (TLB) miss. This is the same as the event with encoding 011010 (pipeline stalled waiting for data memory read), but only for TLB miss.

MMX instruction data reads (49, 110001/0):

Analogous to “Data reads,” counting only MMX instruction accesses.

MMX instruction data read misses (49, 110001/1):

Analogous to “Data read misses,” counting only MMX instruction accesses.

Floating Point stalls (50, 110010/0):

This event counts the number of clocks while pipe is stalled due to a floating-point freeze.

Taken Branches (50, 110010/1):

This event counts the number of taken branches.

D1 starvation and FIFO is empty (51, 110011/0), D1 starvation and only one instruction in FIFO (51, 110011/1):

The D1 stage can issue 0, 1, or 2 instructions per clock if those are available in an instructions FIFO buffer. The first event counts how many times D1 cannot issue ANY instructions since the FIFO buffer is empty. The second event counts how many times the D1-stage issues just a single instruction since the FIFO buffer had just one instruction ready. Combined with previously defined events, Instruction Executed (010110) and Instruction Executed in the V-pipe (010110), the second event enables the user to calculate the numbers of time pairing rules prevented issuing of two instructions.

MMX instruction data writes (52, 110001/1):

Analogous to “Data writes,” counting only MMX instruction accesses.

MMX instruction data write misses (52, 110100/1):

Analogous to “Data write misses,” counting only MMX instruction accesses.

Pipeline flushes due to wrong branch prediction (53, 110101/0), Pipeline flushes due to wrong branch prediction resolved in WB-stage(53, 110101/1):

Counts any pipeline flush due to a branch which the pipeline did not follow correctly. It includes cases where a branch was not in the BTB, cases where a branch was in the BTB but was mispredicted, and cases where a branch was correctly predicted but to the wrong address. Branches are resolved in either the Execute stage (E-stage) or the Writeback stage (WB-stage). In the later case, the misprediction penalty is larger by one clock. The two events count the number of pipeline flushes due to wrong branch predictions. The first event counts the number of wrong branch predictions resolved in either the E-stage or the WB-stage. The second event counts the number of wrong branch prediction resolved in the WB-stage. The difference between these two counts is the number of E-stage resolved branches.

Misaligned data memory reference on MMX instruction (54, 110110/0):

Analogous to “Misaligned data memory reference,” counting only MMX instruction accesses.

Pipeline stalled waiting for MMX instruction data memory read (54, 110110/1):

Analogous to “Pipeline stalled waiting for data memory read,” counting only MMX instruction accesses.

Returns predicted incorrectly or not predicted at all (55, 110111/0):

These are the actual number of Returns that were either incorrectly predicted or were not predicted at all. It is the difference between the total number of executed returns and the number of returns that were correctly predicted. Only RET instructions are counted (e.g., IRET instructions are not counted.).

Returns predicted (correctly and incorrectly) (55, 110111/1):

This is the actual number of Returns for which a prediction was made. Only RET instructions are counted (e.g. IRET instructions are not counted).

MMX multiply unit interlock (56, 111000/0):

This is the number of clocks the pipe is stalled since the destination of previous MMX multiply instruction is not ready yet. The counter will not be incremented if there is another cause for a stall. For each occurrence of a multiply interlock this event will be counted twice (if the stalled instruction comes on the next clock after the multiply) or by one (if the stalled instruction comes two clocks after the multiply).

MOVD/MOVQ store stall due to previous operation (56, 111000/1):

Number of clocks a MOVD/MOVQ store is stalled in D2 stage due to a previous MMX operation with a destination to be used in the store instruction.

Returns (57, 111001/0):

This is the actual number of Returns executed. Only RET instructions are counted (e.g., IRET instructions are not counted). Any exception taken on a RET instruction and any interrupt recognized by the processor on the instruction boundary prior to the execution of the RET instruction will also cause this counter to be incremented.

BTB false entries (58, 111010/0):

Counts the number of false entries in the Branch Target Buffer. False entries are causes for misprediction other than a wrong prediction.

BTB miss prediction on a Not-Taken Branch (58, 111010/1):

Counts the number of times the BTB predicted a Not-Taken branch as Taken.

Number of clocks stalled due to full write buffers while executing MMX instructions (59, 111011/0):

Analogous to “Number of clocks stalled due to full write buffers,” counting only MMX instruction accesses.

Stall on MMX instruction write to an E- or M-state line (59, 111011/1):

Analogous to “Stall on write to an E- or M-state line,” counting only MMX instruction accesses.

A

B

C

D

E

F

H

I

K

L

M

N

O

P

R

S

T

U

V

W