Bus Functional Description 19

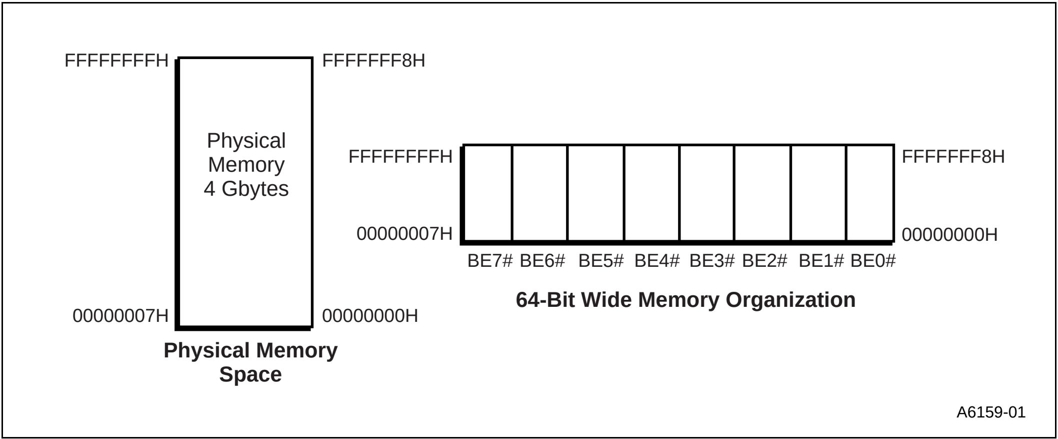
Embedded Pentium® family processors support the same bus functionality. The processor bus supports a 528-Mbyte/s data transfer rate at 66 MHz. All data transfers occur as a result of one or more bus cycles. This chapter describes the processor bus cycles and the processor data transfer mechanism.

19.1 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.

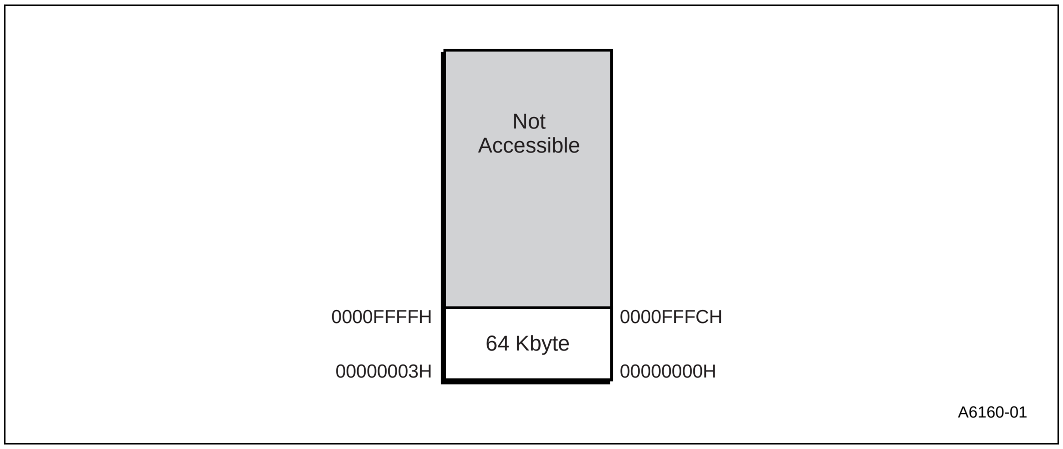
In hardware, memory space is organized as a sequence of 64-bit quantities. Each 64-bit location has eight individually addressable bytes at consecutive memory addresses (see Figure 19-1).

Figure 19-1. 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 19-2 for a conceptual diagram of the I/O space.

Figure 19-2. 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 19-2.

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 19-2, BHE# and BLE# can be decoded from the byte enables according to Table 19-3 and Table 19-4.

To address 8-bit memories, the lower three address lines (A2–A0) must be decoded from the byte enables as indicated in Table 19-2.

19.2 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 19-1. 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 19-2. 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 19-3 and Table 19-4).

Table 19-1. 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 19-2. 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 19-3 and Table 19-4).

Table 19-2. 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 19-3. 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 19-4. 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 19-5. When BE3'# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BE3'#
Low	X	X	X	Low	X	X	X	Low

Table 19-6. When BE2'# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BE2'#
X	Low	X	X	X	Low	X	X	Low

Table 19-7. When BE1'# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BE1'#
X	X	Low	X	X	X	Low	X	Low

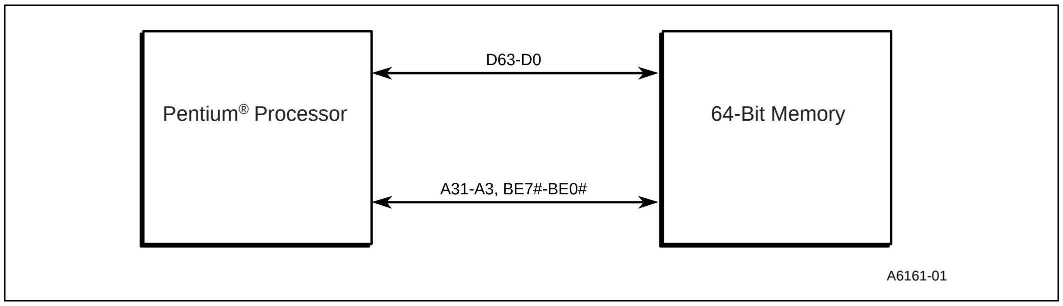
Table 19-8. When BE0'# is Active

BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#	BE0'#
X	X	X	Low	X	X	X	Low	Low

19.2.1 Interfacing With 8-, 16-, 32-, and 64-Bit Memories

In 64-bit physical memories such as Figure 19-3, 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 19-3. 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 19-4 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 19-2 through Table 19-8.

Figure 19-4. Addressing 32-, 16- and 8-Bit Memories

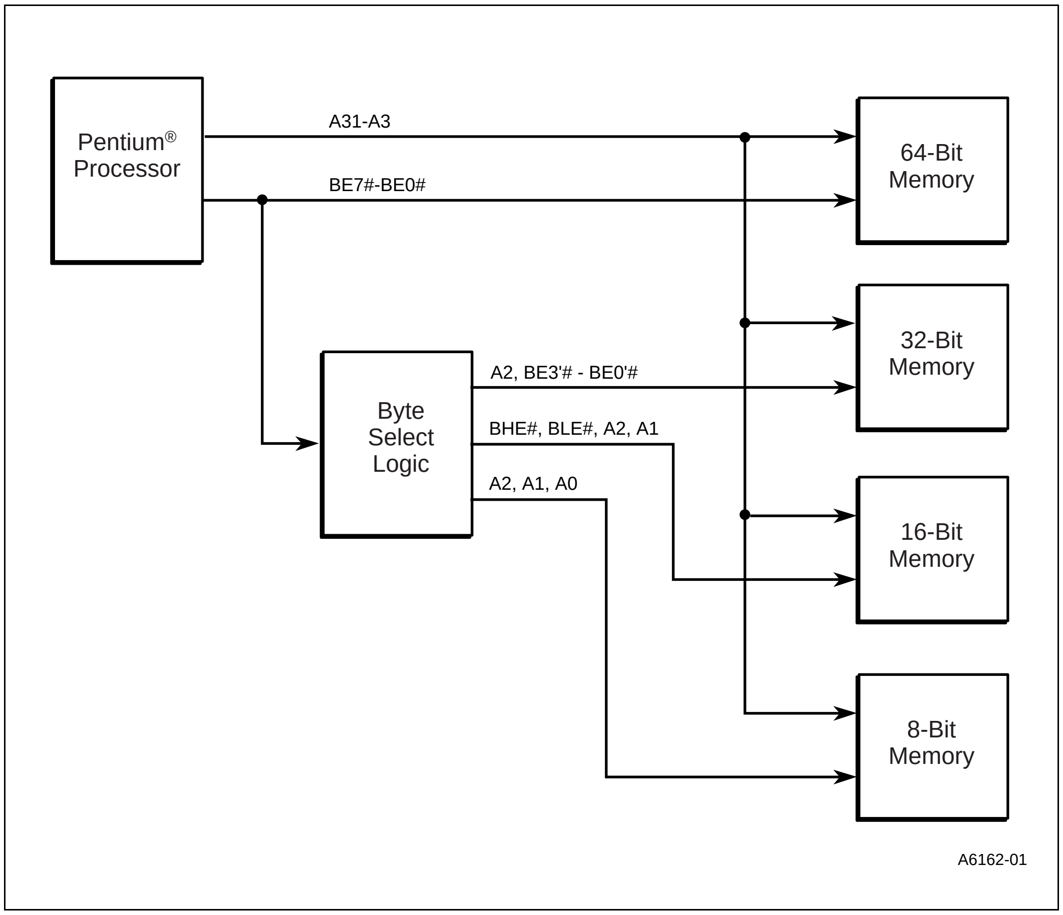
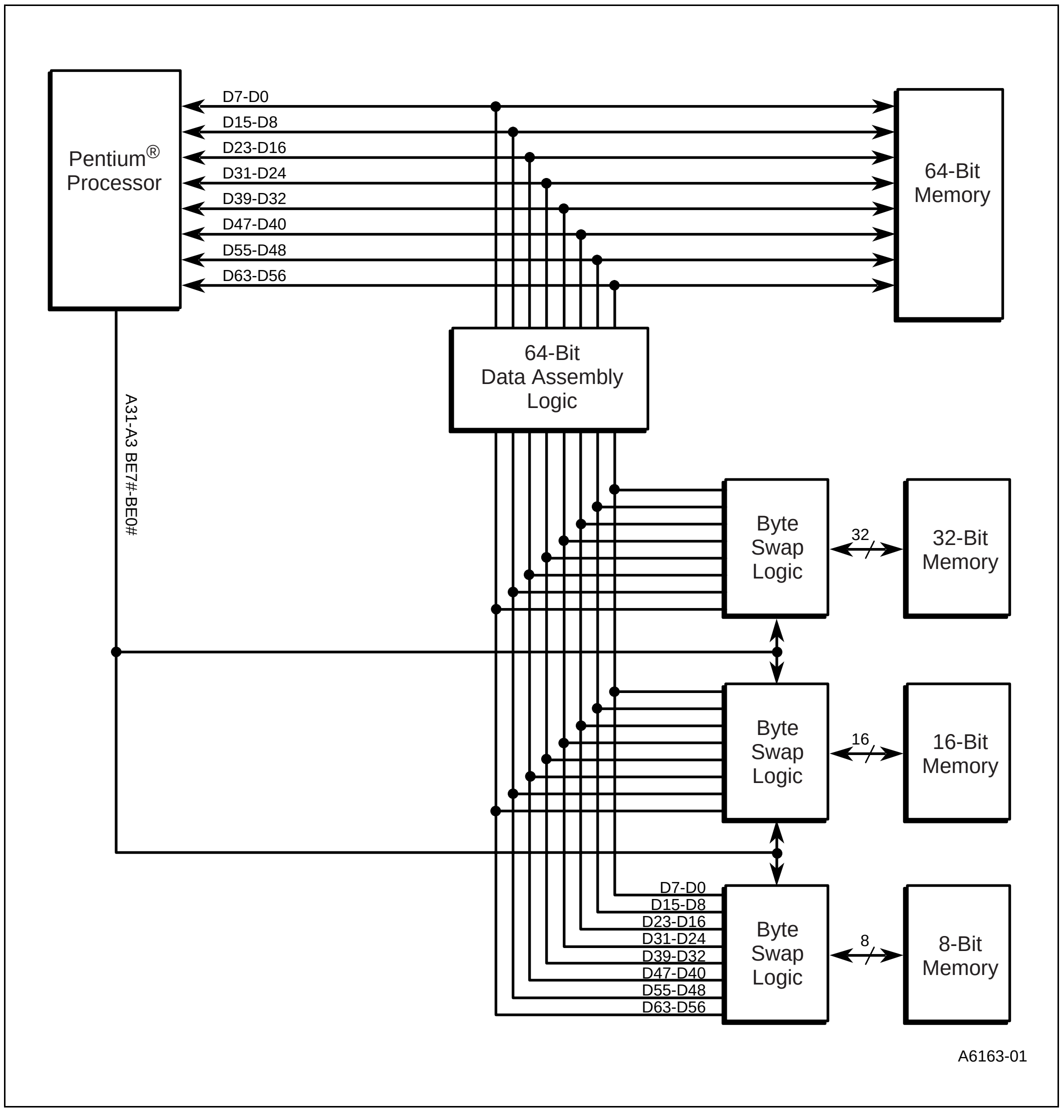


Figure 19-5 shows the processor data bus interface to 32-, 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to and received from the processor on the correct data pins (see Table 19-1). For memory widths smaller than 64 bits, byte assembly logic is needed to return all bytes of data requested by the processor in one cycle.

Figure 19-5. 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 19-9 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 19-9. 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

19.3 Bus State Definition

This section describes the processor bus states in detail. See Figure 19-6 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).

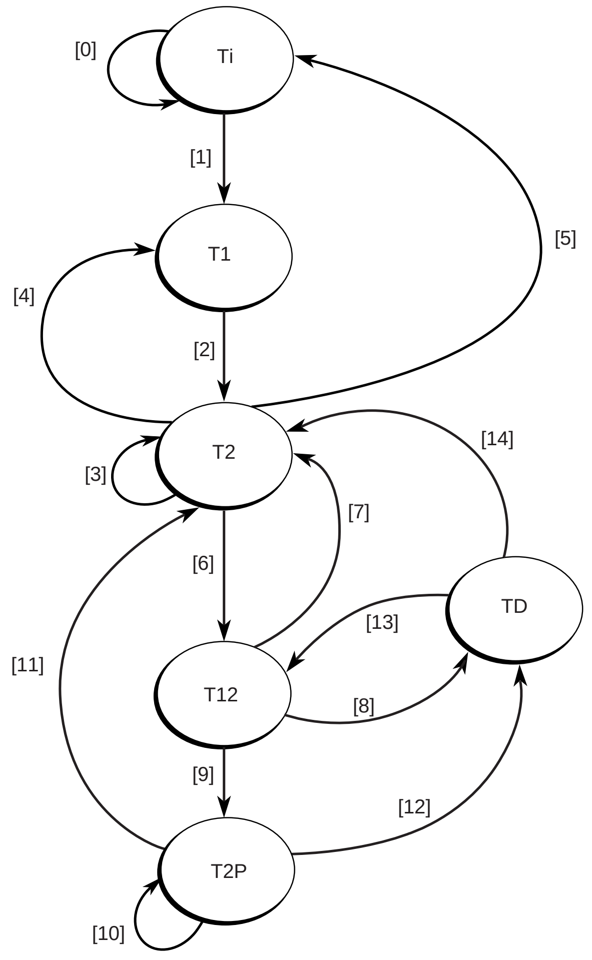
TD: This state indicates there is one outstanding bus cycle, that its address, status and ADS# have already been driven sometime in the past (in state T12), and that the data and BRDY# pins are not being sampled because the data bus requires one dead clock to turn around between consecutive reads and 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 19-10 gives a brief summary of bus activity during each bus state. Figure 19-6 shows the processor bus state diagram.

Table 19-10. 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 19-6. Processor Bus Control State Machine



NOTES:

ADS# asserted

If BOFF# is asserted during any state, a state transition to Ti occurs in the next clock (not shown)

If RESET is sampled asserted in any state, a state transition to Ti will occur (not shown

A6164-01

19.3.1 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.

19.4 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 19-328 describes each of the bus states, and shows the bus state diagram.

Table 19-11 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 19-11. 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 19-13)	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.

19.4.1 Single-Transfer Cycle

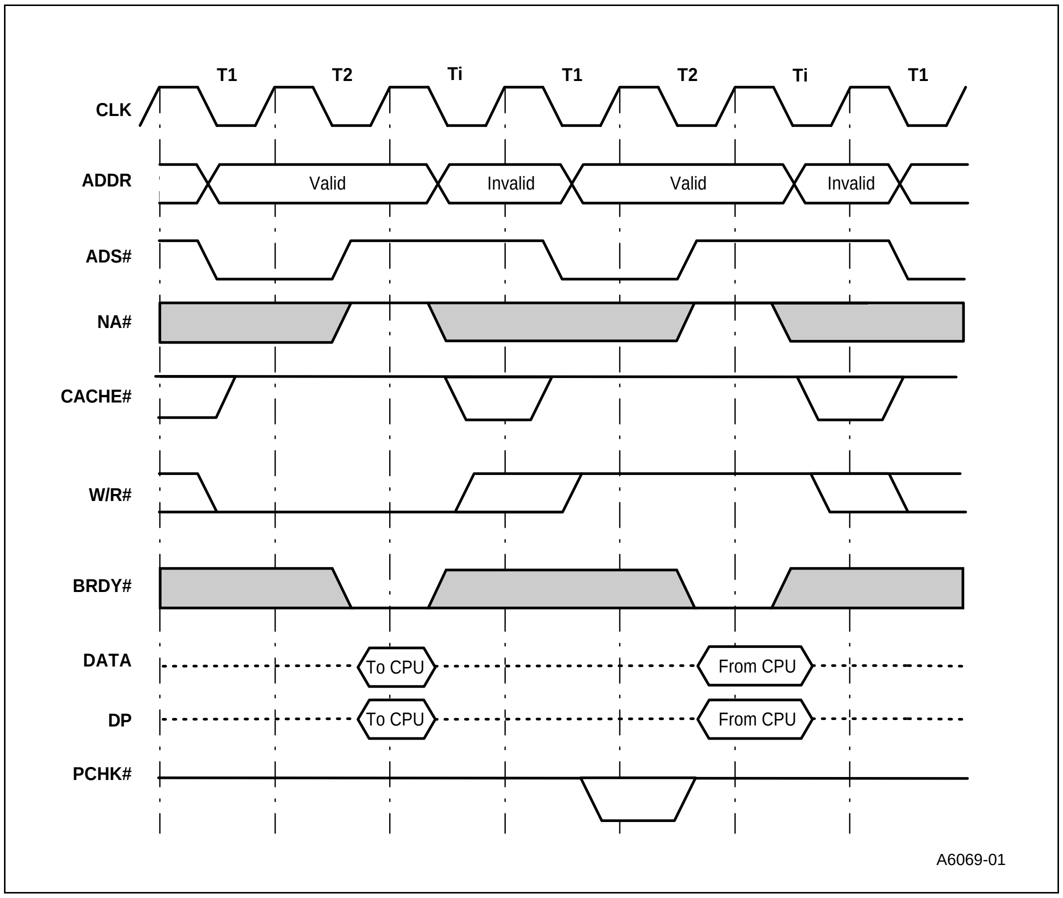
The processor supports a number of different types of bus cycles. The simplest type of bus cycle is a single-transfer non-cacheable 64-bit cycle, either with or without wait states. Non-pipelined read and write cycles with 0 wait states are shown in Figure 19-7.

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 19-328 for more information).

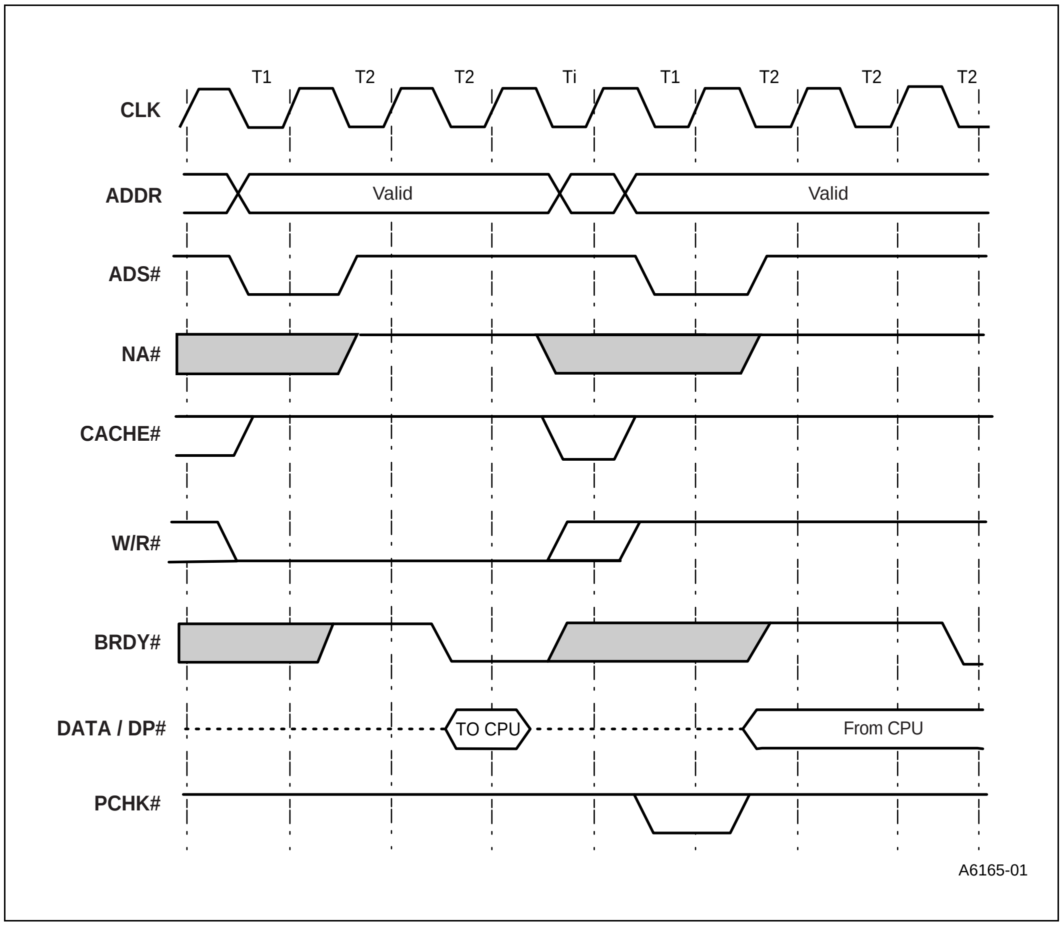
The timing of the data parity input, DP, and the parity check output, PCHK#, is also shown in Figure 19-7. DP is driven by the processor and returned to the processor in the same clock as the data. PCHK# is driven two clocks after BRDY# is returned for reads with the results of the parity check.

Figure 19-7. 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 19-8. 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 19-8. Non-Pipelined Read and Write with Wait States



19.4.2 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 19-12). 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 19-12.

Table 19-12. 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 19-11), 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.

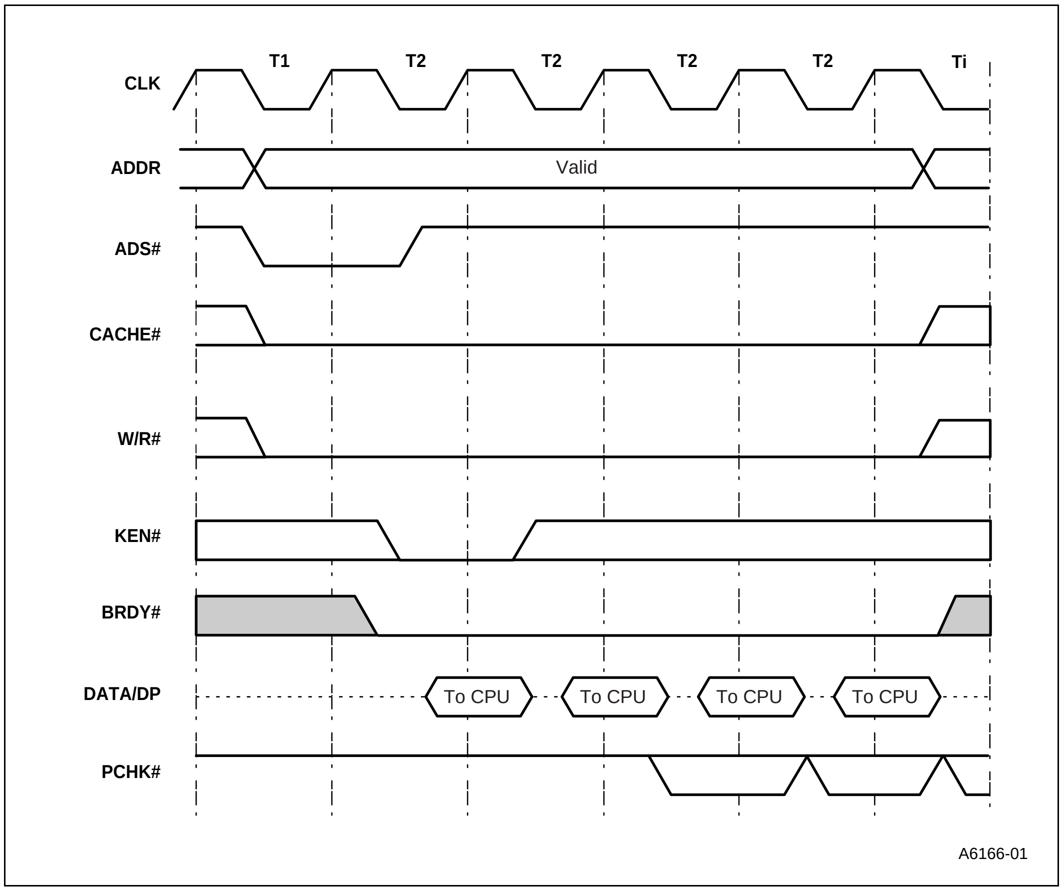
19.4.2.1 Burst Read Cycles

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 19-9 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.

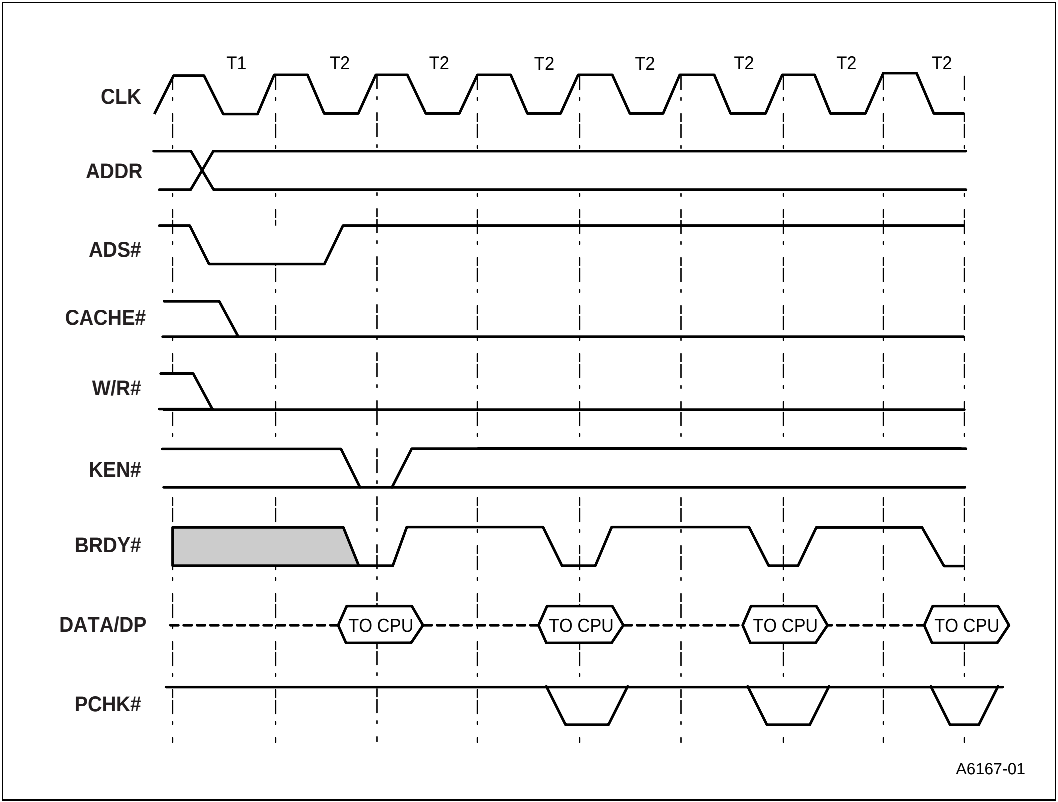
PCHK# is driven with the parity check status two clocks after each BRDY#.

Figure 19-9. 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 19-10.

Figure 19-10. Slow Burst Read Cycle



19.4.2.2 Burst Write Cycles

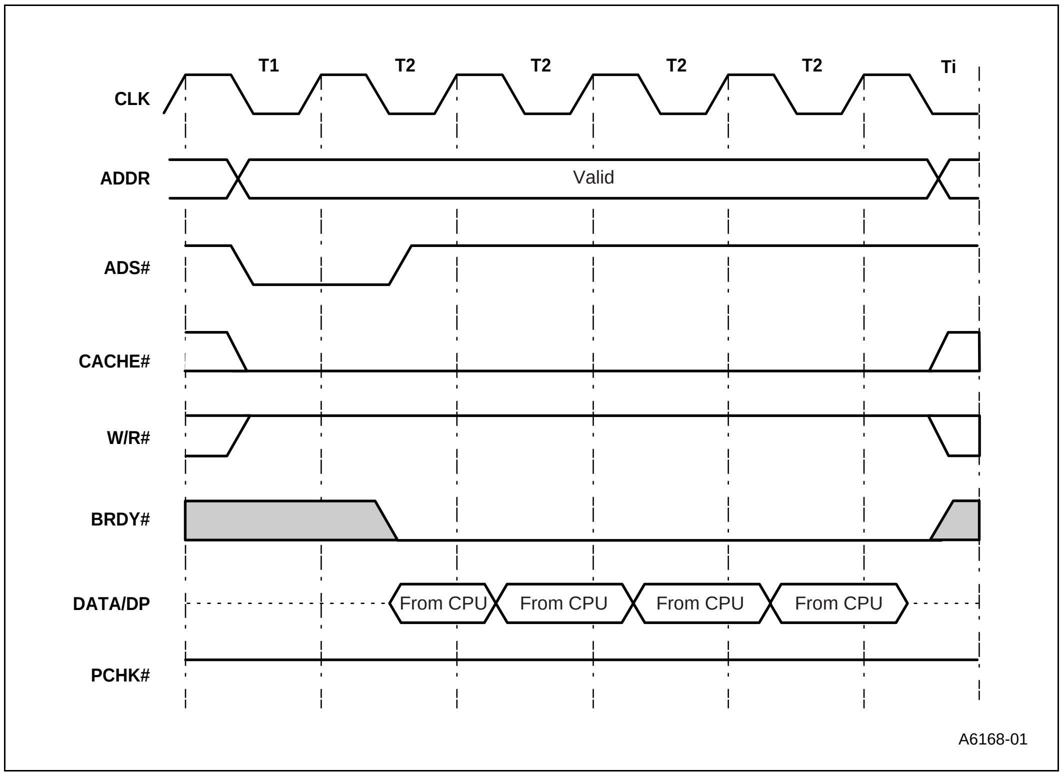
Figure 19-11 shows the timing diagram of basic burst write cycle. KEN# is ignored in burst write cycle. If the CACHE# pin is active (low) during a write cycle, it indicates that the cycle is a burst writeback cycle. Burst write cycles are always writebacks of modified lines in the data cache. Writeback 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 19-11. 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.

19.4.3 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.

19.4.3.1 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.

19.4.3.2 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.

19.4.3.3 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.

19.4.3.4 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.

19.4.3.5 LOCK# Timing and Latency

The timing of LOCK# is shown in Figure 19-12. 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 19-13 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 19-14 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 19-12. LOCK# Timing

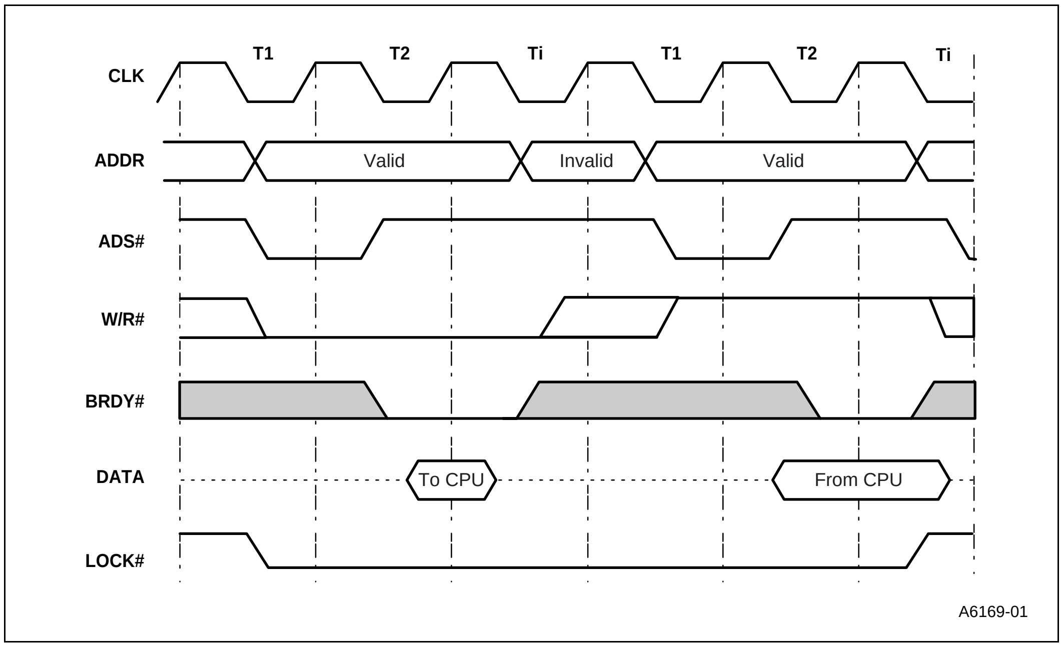


Figure 19-13. Two Consecutive Locked Operations

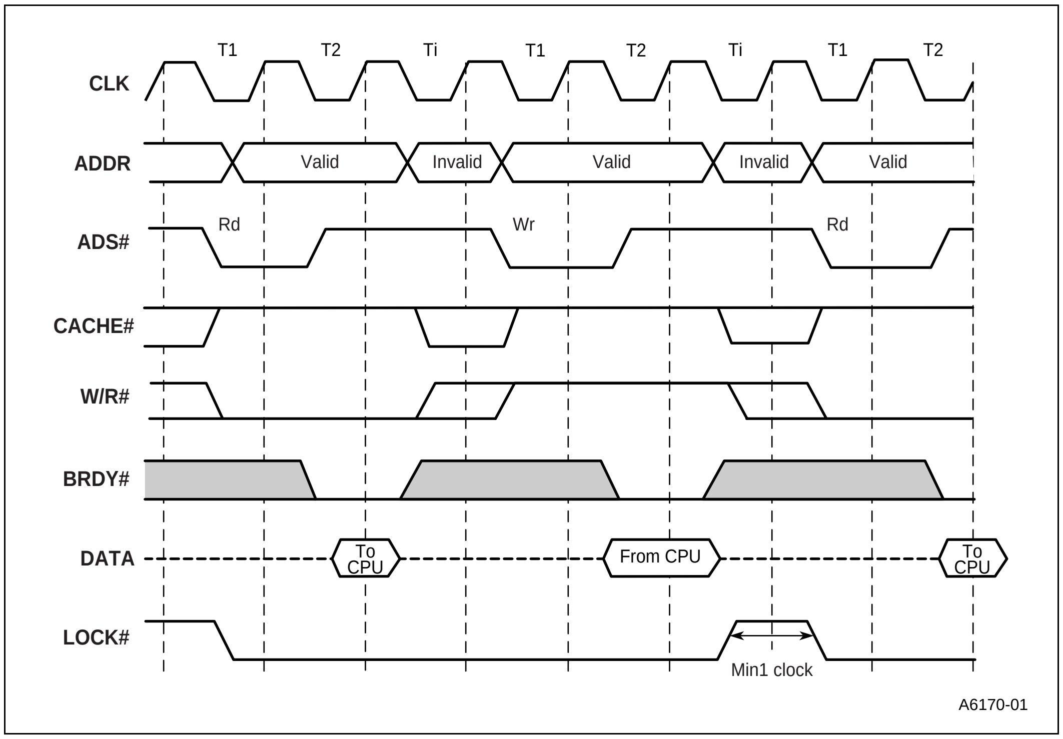
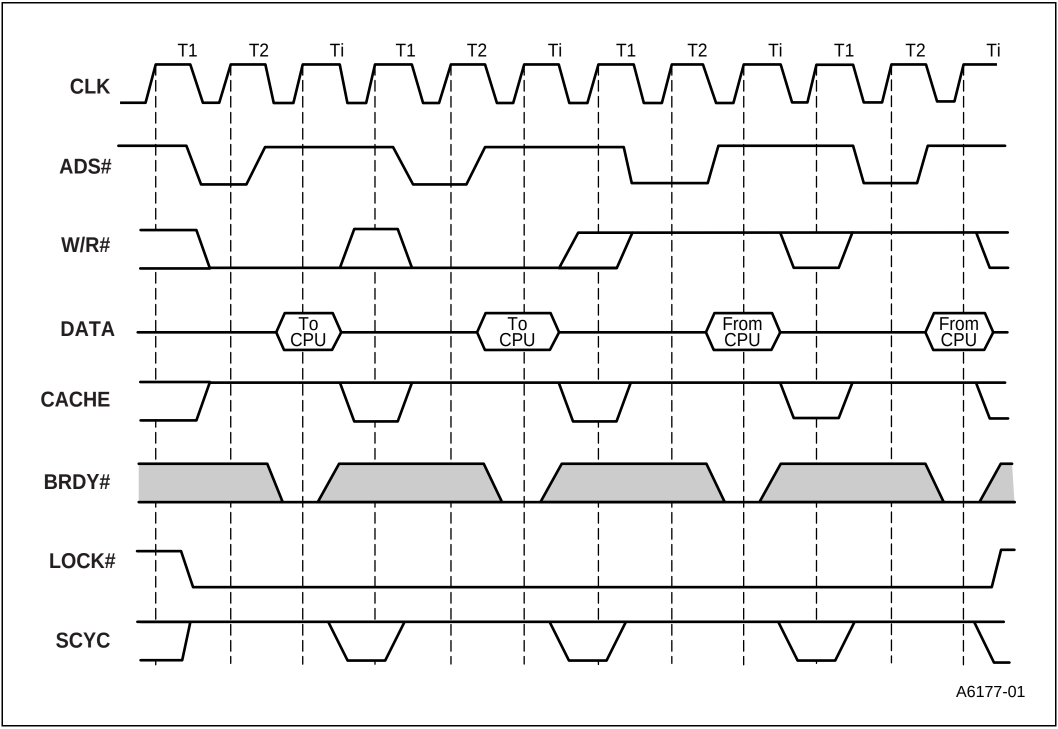


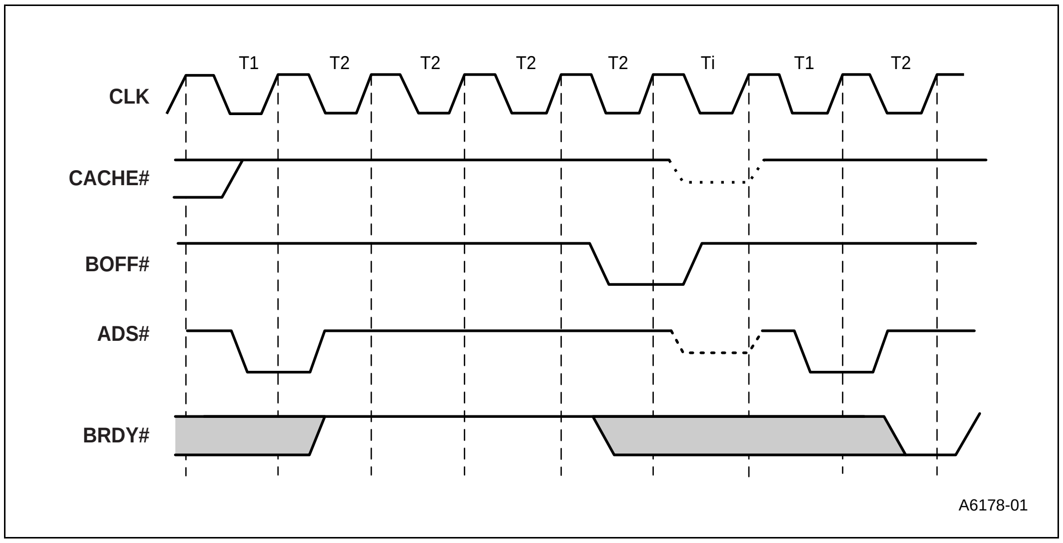
Figure 19-14. Misaligned Locked Cycles



19.4.4 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 19-15). 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 19-15. 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.

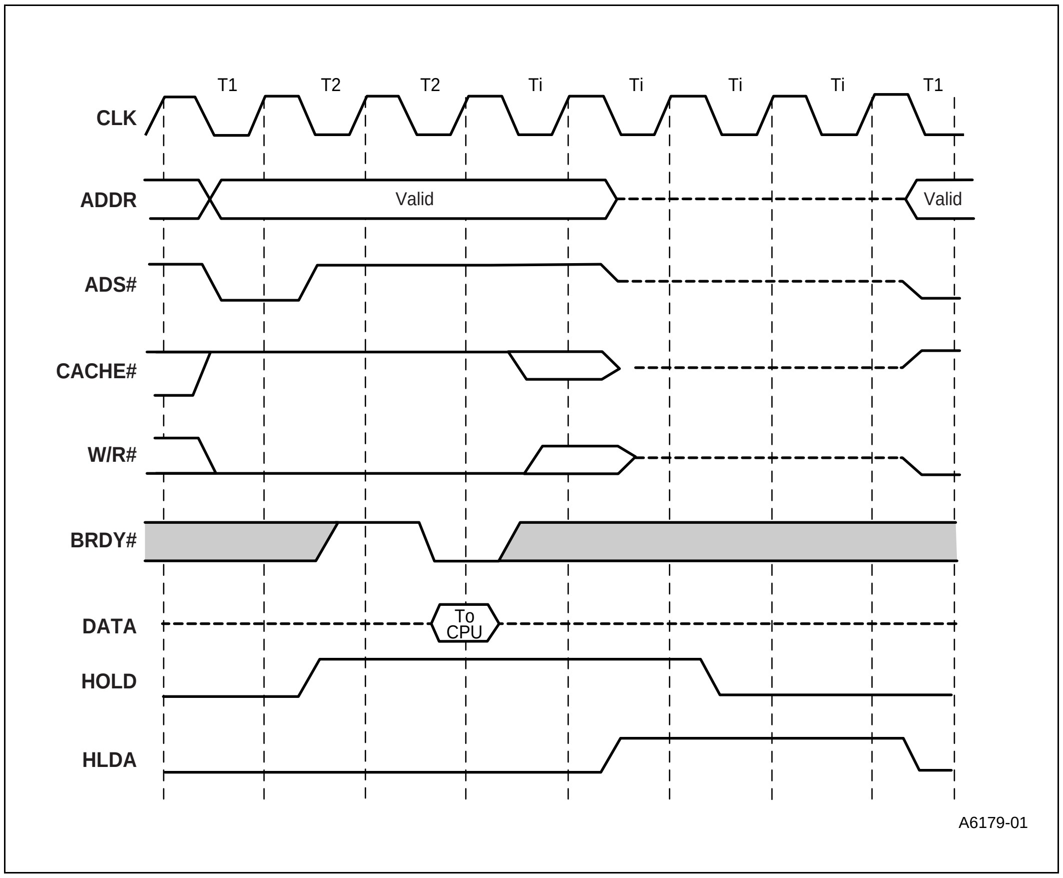
Asserting BOFF# in the same clock as ADS# may cause the processor to leave the ADS# signal floating low. Since ADS# is floating low, a peripheral device may think that a new bus cycle has begun even though the cycle was aborted. There are several ways to approach this situation:

- 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#.

19.4.5 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 19-16.

Figure 19-16. 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 19-16 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.

19.4.6 Interrupt Acknowledge

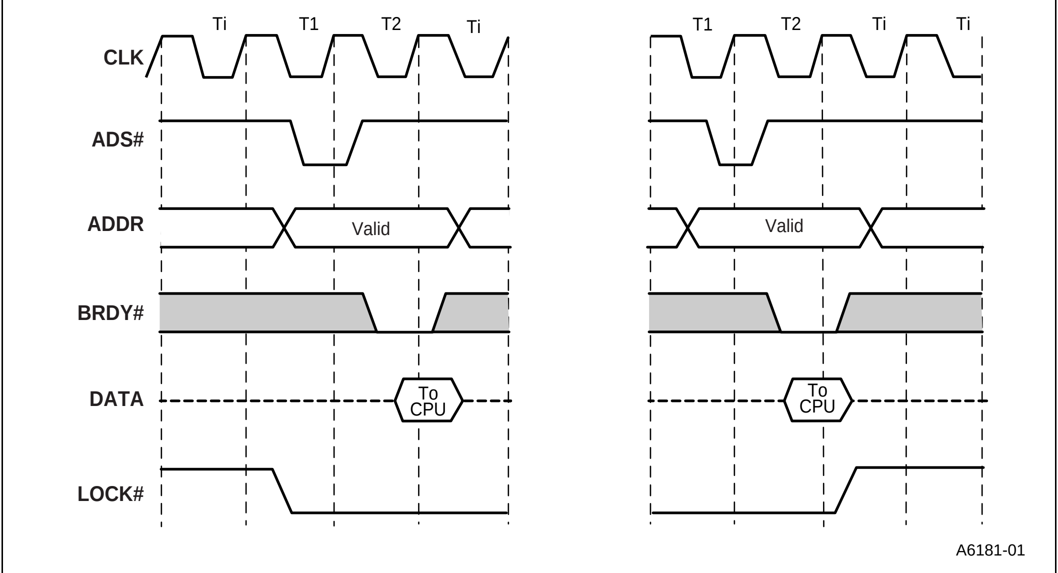
The processor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin (if interrupts are enabled). Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

An example interrupt acknowledge transaction is shown in Figure 19-17. 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 19-17. Interrupt Acknowledge Cycles



19.4.7 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.

19.4.8 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 19-13 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 19-13. 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.

19.4.9 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 22, “Error Detection” for details.

19.4.10 Pipelined Cycles

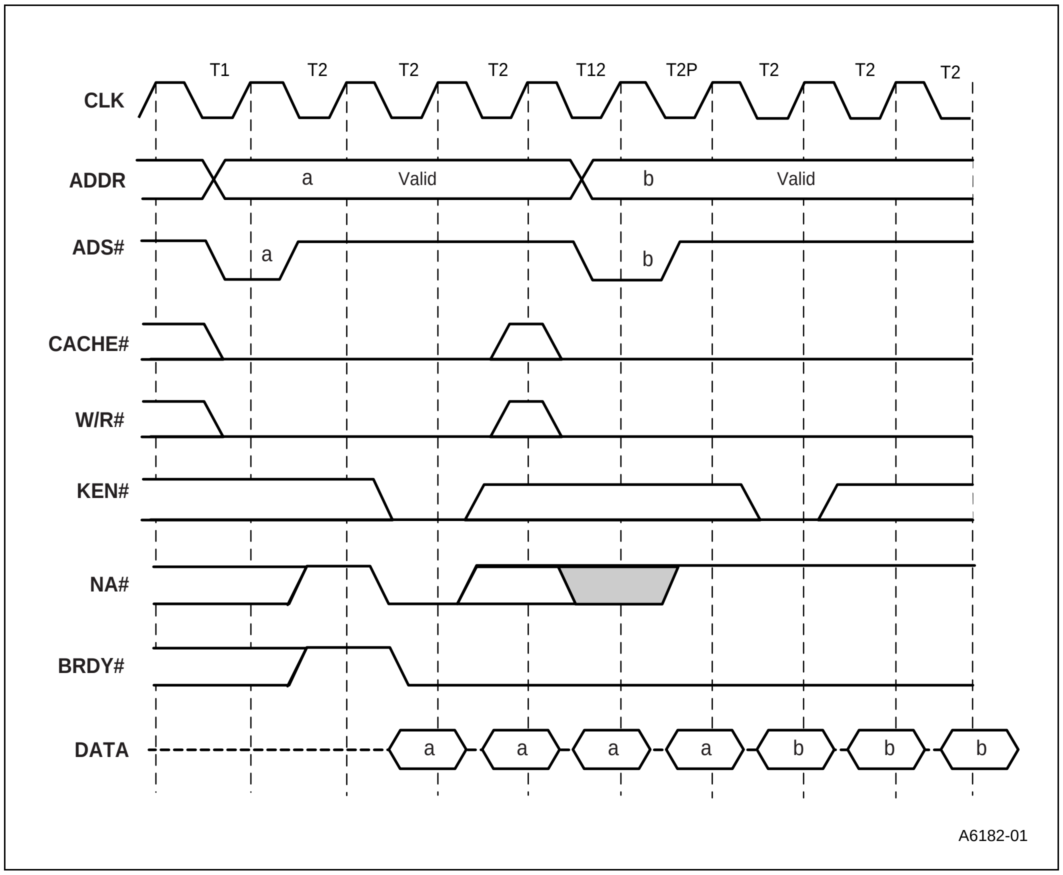
The NA# input indicates to the processor that it may drive another cycle before the current one is completed. Cacheability (KEN#) and cache policy (WB/WT#) indicators for the current cycle are sampled in the same clock NA# is sampled active (or the first BRDY# for that cycle, whichever comes first). 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.

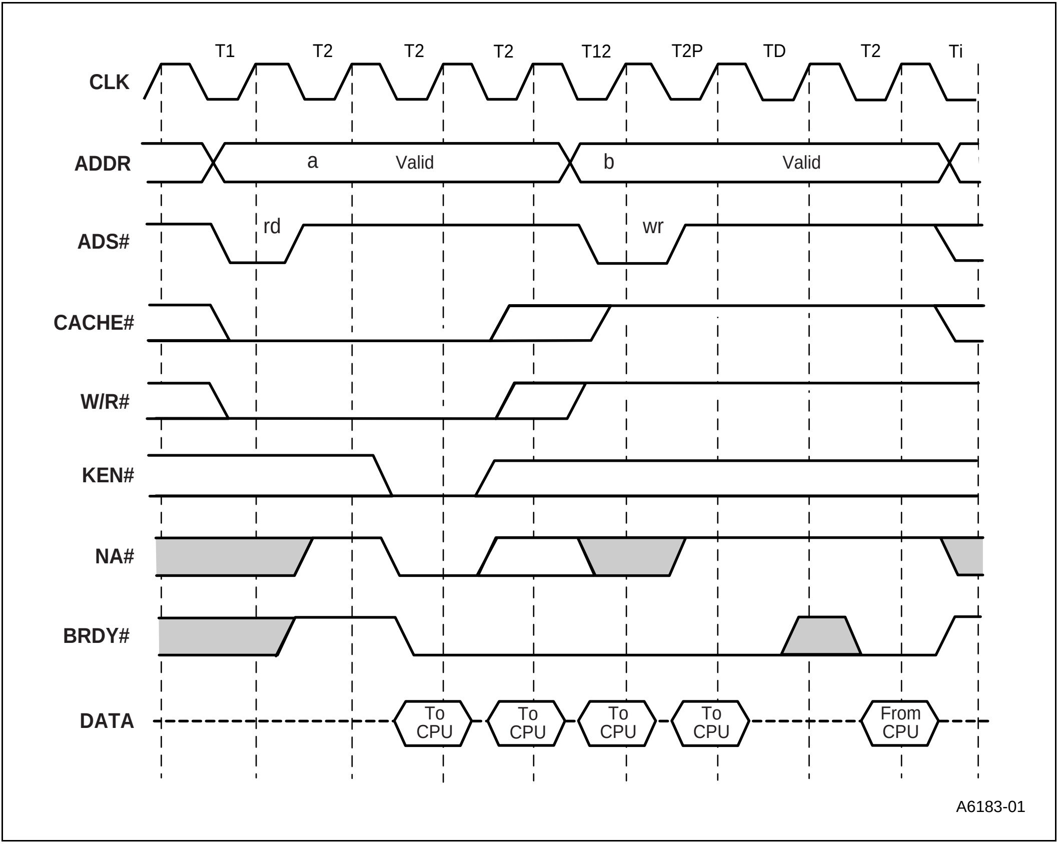
An example of burst pipelined back to back reads is shown in Figure 19-18. The assertion of NA# causes a pending cycle to be driven two clocks later. Note KEN# timing.

Figure 19-18. 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 19-6, “Processor Bus Control State Machine” on page 19-329). Pipelined back-to-back read/write cycles are shown in Figure 19-19.

Figure 19-19. Pipelined Back-to-Back Read/Write Cycles



19.4.10.1 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 19-20 and Figure 19-21 clarify this specification.

Figure 19-20 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 19-20. KEN# and WB/WT# Sampling with NA#

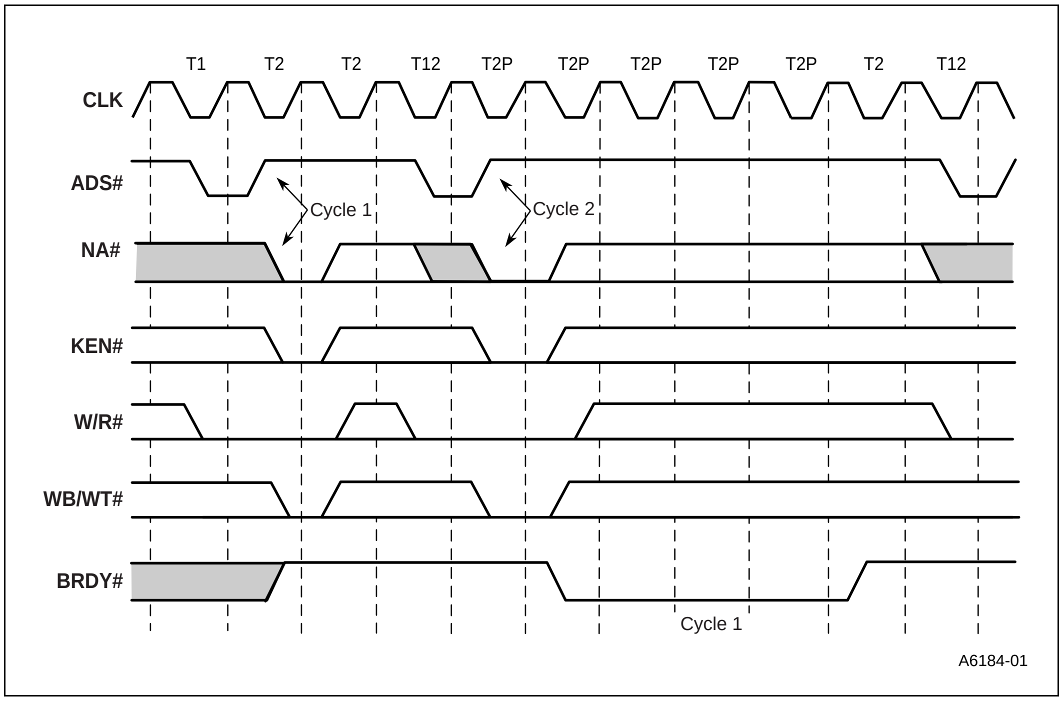
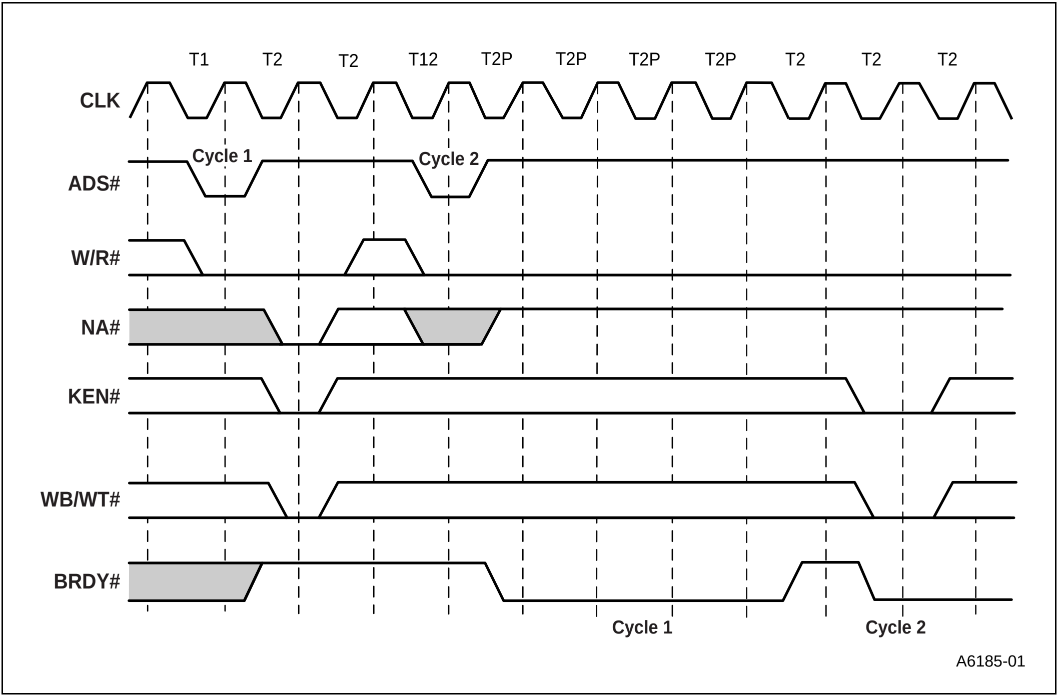


Figure 19-21 shows that two cycles are outstanding on the processor bus. The assertion of NA# caused the sampling of KEN# and WB/WT# for the first cycle. The assertion of the four BRDY#s for the first cycle DO NOT cause the KEN# and WB/WT# for the second cycle to be sampled. In this example, KEN# and WB/WT# for the second cycle are sampled with the first BRDY# for the second cycle.

Figure 19-21. KEN# and WB/WT# Sampling with BRDY#

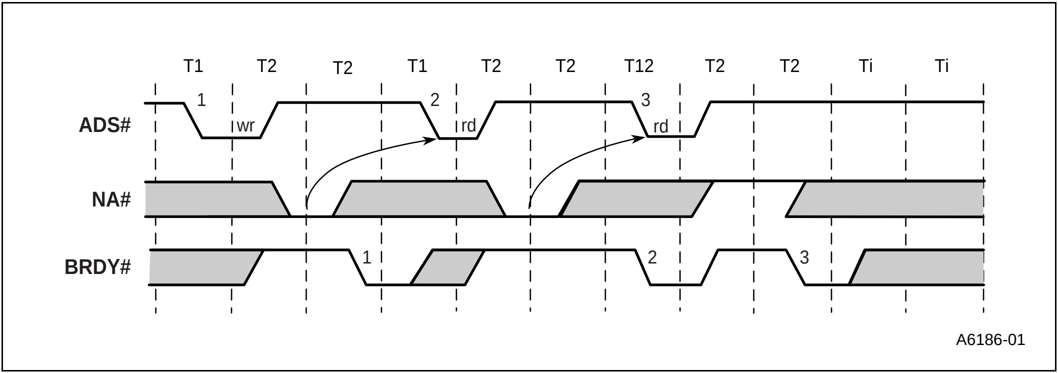


19.4.11 Dead Clock Timing Diagrams

The timing diagrams in Figure 19-22 and Figure 19-23 show bus cycles with and without a dead clock.

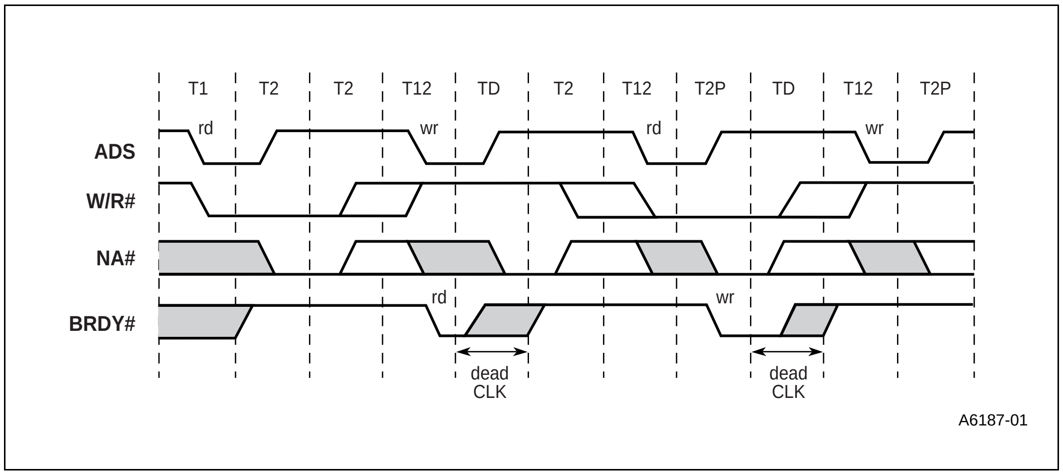
In Figure 19-22, cycles 1 and 2 can be either read or write cycles and no dead clock would be needed because only one cycle is outstanding when those cycles are driven. To prevent a dead clock from being necessary after cycle 3 is driven, it must be of the “same type” as cycle 2. That is if cycle 2 is 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 19-22. 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 19-23. Bus Cycles with TD Dead Clock



19.5 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 19-25.

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 19-24 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 19-25 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 19-24 and Figure 19-25 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 19-26. 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 19-24. Inquire Cycle that Misses the Processor Cache

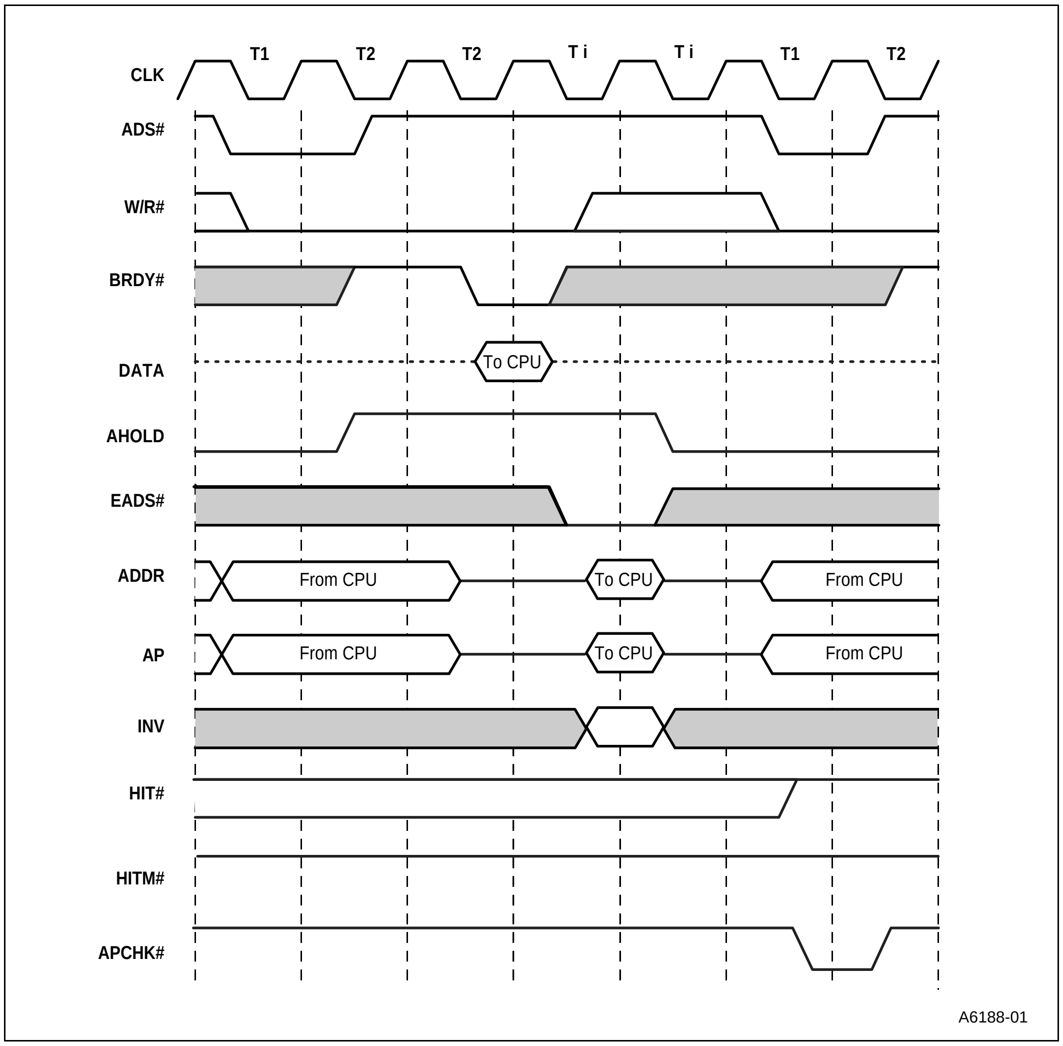
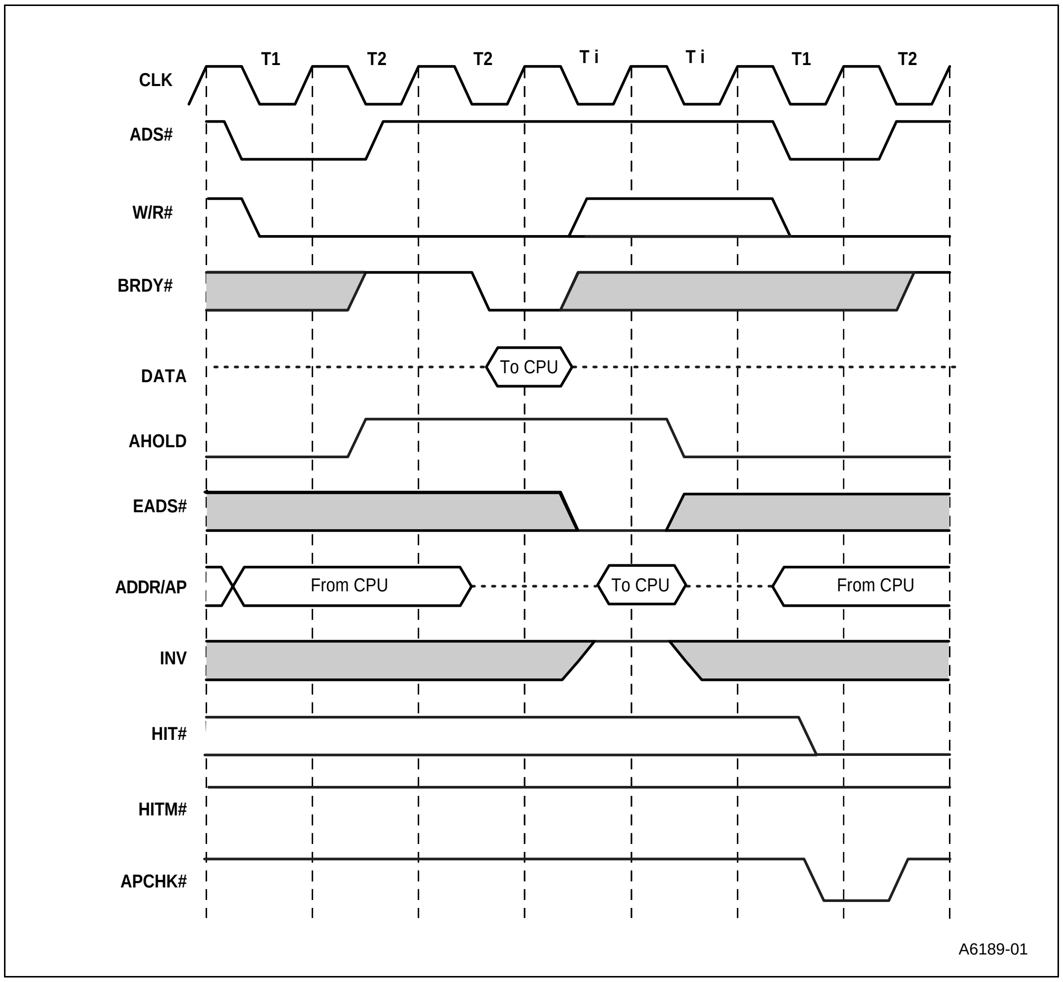


Figure 19-25. Inquire Cycle that Invalidates a Non-M-State Line



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 19-28). This can happen if an inquire cycle hits a line in one of the processor writeback buffers.

19.5.1 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 19-26 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 19-27 and Figure 19-28 depict restrictions (i) and (ii) respectively. Note that there are no restrictions on the assertion of AHOLD.

Figure 19-26. Inquire Cycle that Invalidates M-State Line

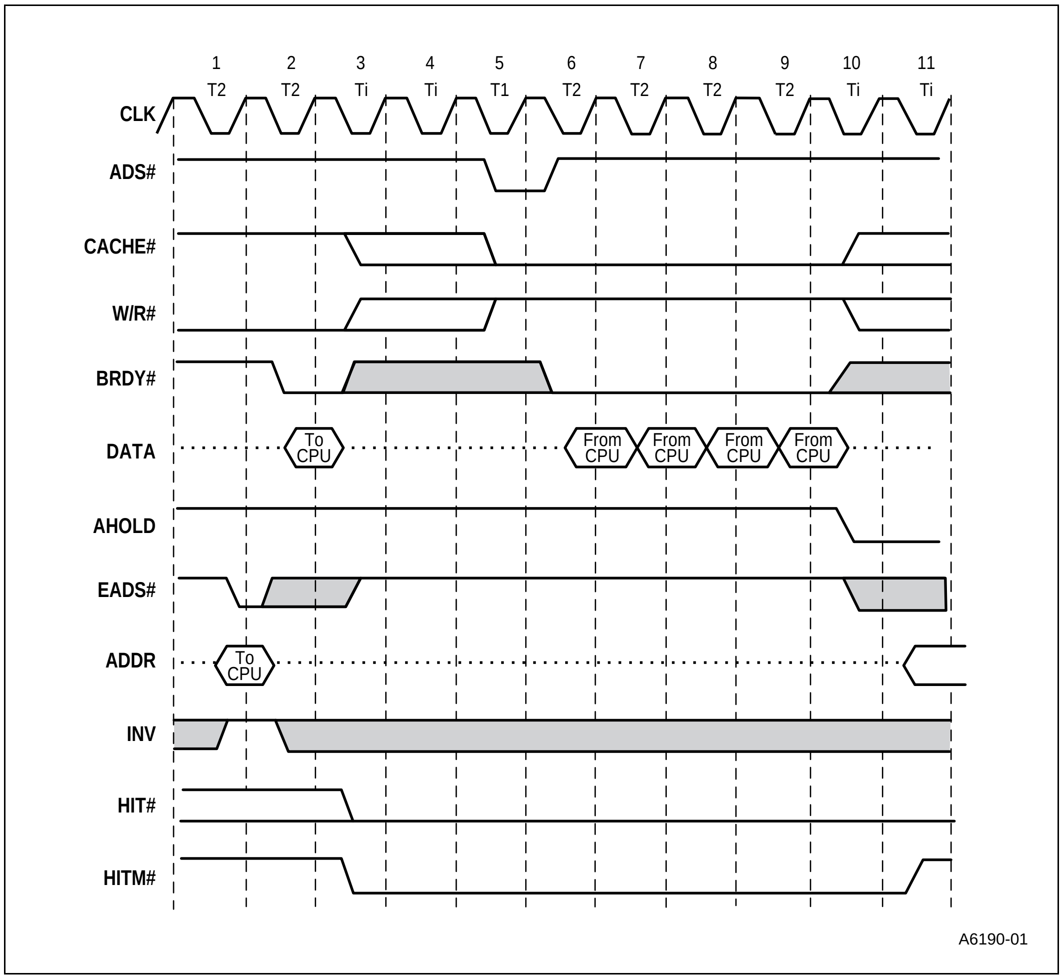


Figure 19-27 shows a writeback (due to a previous snoop that is not shown). ADS# for the writeback is asserted even though AHOLD is asserted. Note that AHOLD can be deasserted in Clock 2, 4, 7, or 9. AHOLD cannot be deasserted in Clock 1, 3, 5, 6, or 8.

Figure 19-27. AHOLD Restriction during Write Cycles

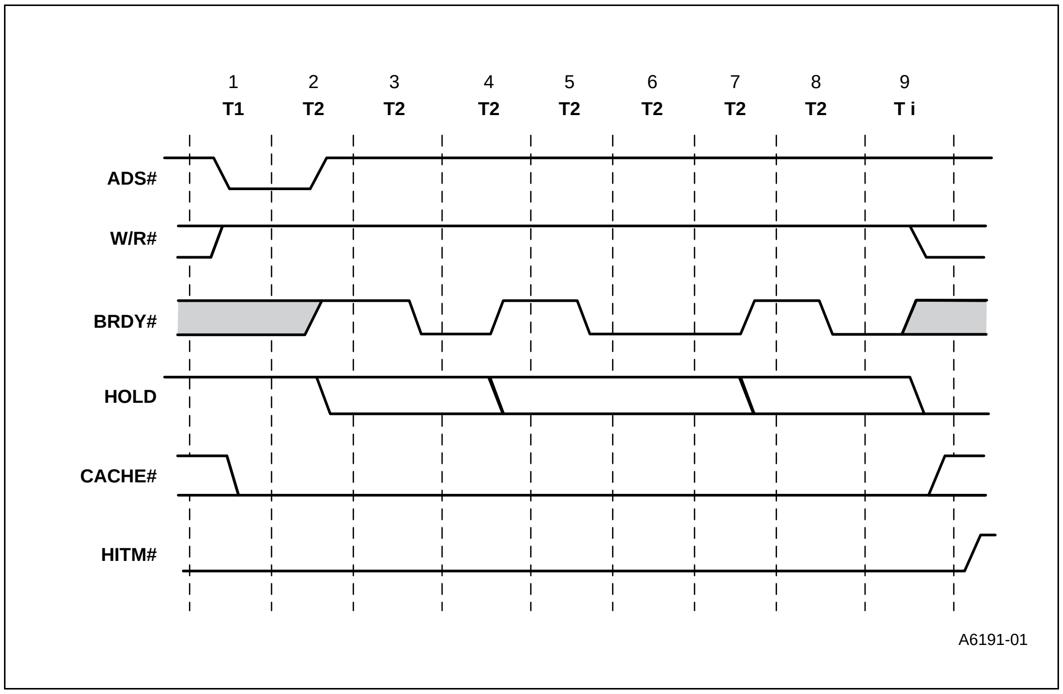
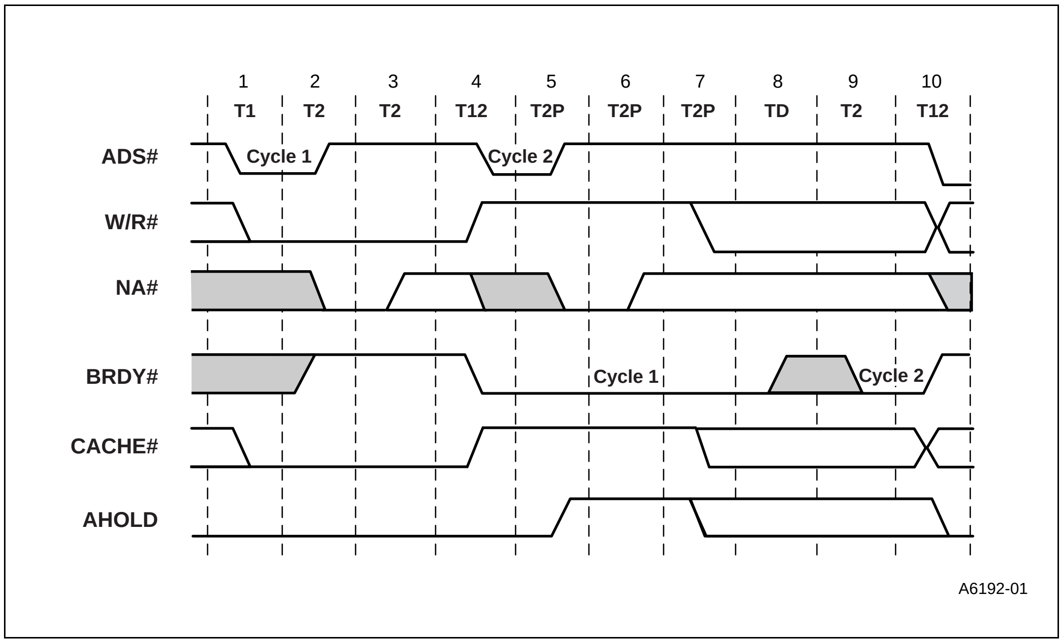


Figure 19-28 shows a write cycle being pipelined into a read cycle. Note that if AHOLD is asserted in Clock 5, it can be deasserted in Clock 7 before the TD, or in Clock 10 after the TD, but it cannot be deasserted in Clock 8 (the TD clock). AHOLD cannot be deasserted in Clock 9 because BRDY# for the write cycle is being returned.

Figure 19-28. AHOLD Restriction During TD



19.5.2 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.

19.5.3 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.

19.5.4 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 19-29 and Figure 19-30 illustrate the snoop responsibility pickup. Figure 19-29 shows a non-pipelined cycle, while Figure 19-30 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 19-29. Snoop Responsibility Pickup — Non-Pipelined Cycles

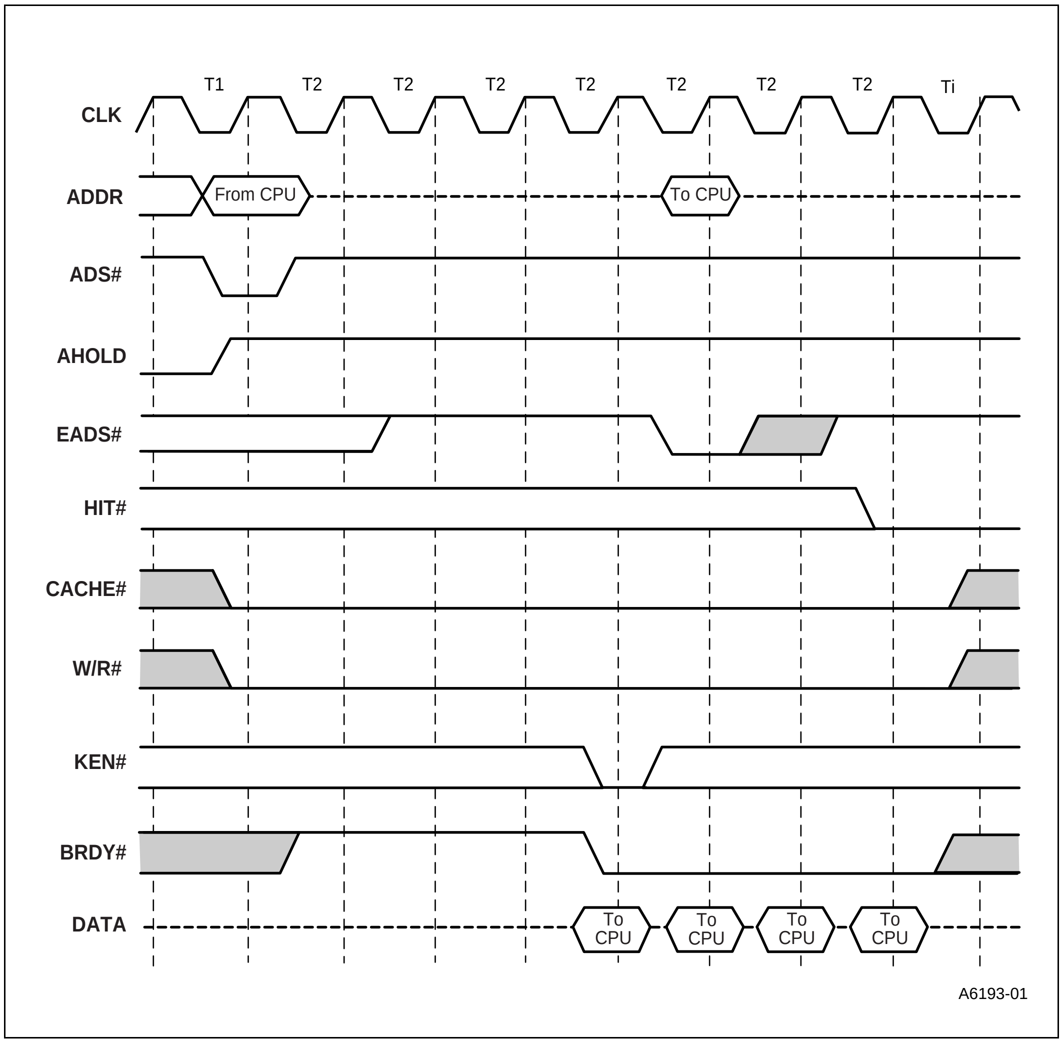
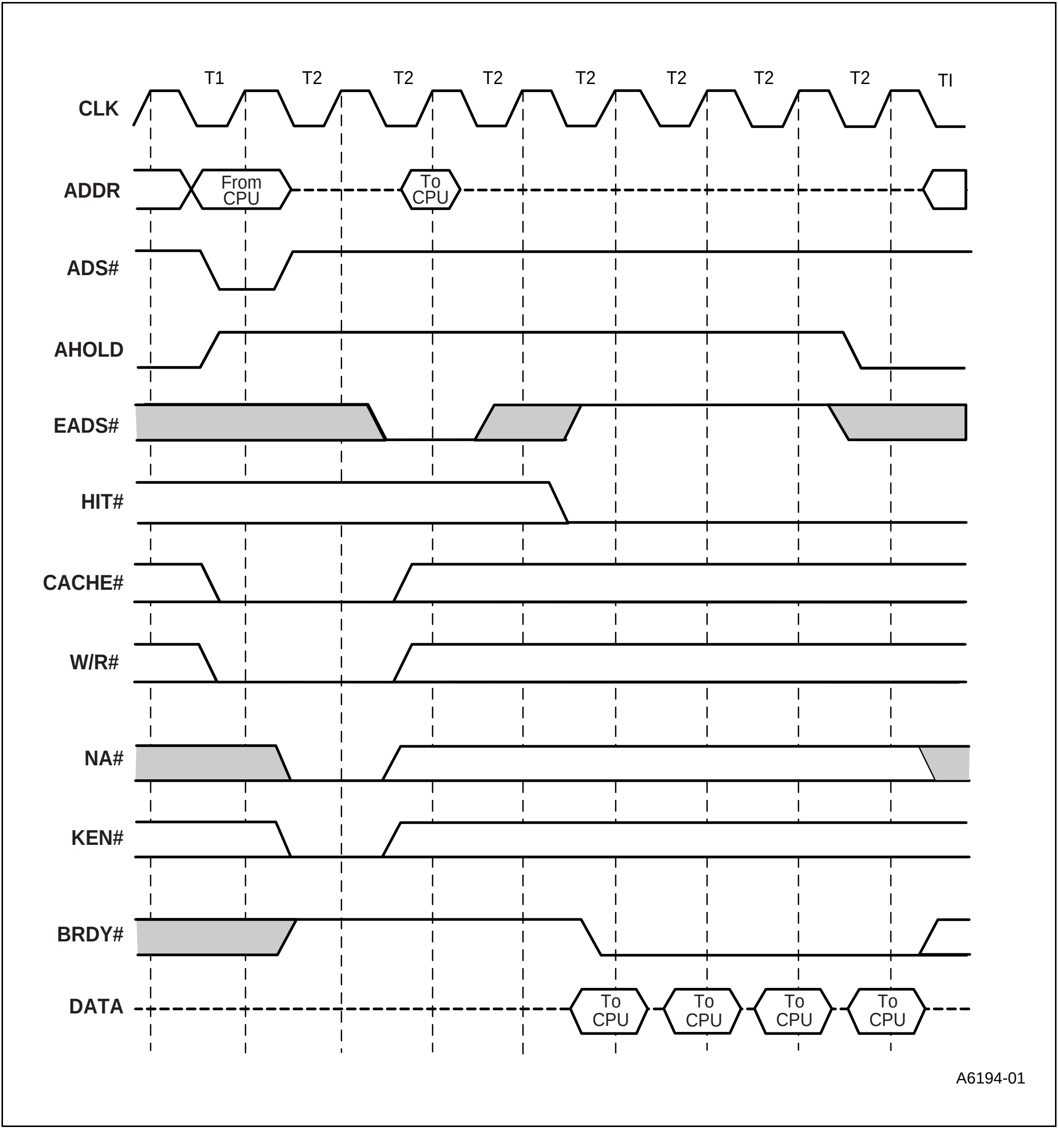


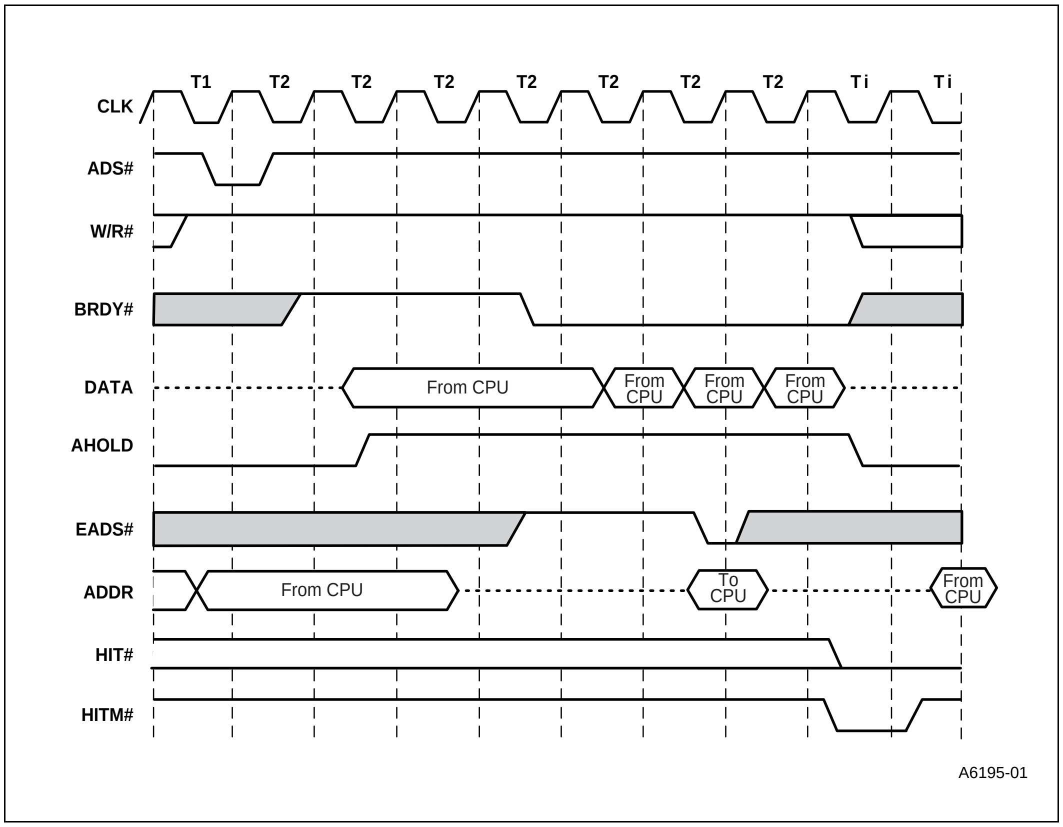
Figure 19-30. Snoop Responsibility Pickup — Pipelined Cycle



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 19-31 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 19-31. Latest Snooping of Writeback Buffer



19.6 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

19.6.1 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 17, “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).

19.6.2 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.

19.6.3 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.

19.6.4 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#.

19.6.5 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.

19.6.6 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.

19.6.7 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.

19.6.8 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.

19.6.9 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.

19.6.10 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.