3.3.8 Bus Arbitration

An external bus master can take control of the CPU's bus using either the HOLD/HLDA handshake signals or the back-off (BOFF#) input. Both mechanisms force the IBM 6x86 CPU to enter the bus hold state.

3.3.8.1 HOLD and HLDA

Using the HOLD/HLDA handshake, an external bus master requests control of the CPU's bus by asserting the HOLD signal. In response to an active HOLD signal, the CPU completes all outstanding bus cycles, enters the bus hold state by floating the bus, and asserts the HLDA output. The CPU remains in the bus hold state until HOLD is negated. Figures 3-18, 3-19 (Page 3-47) and 3-20 (Page 3-48) illustrate the timing associated with requesting HOLD during an idle bus, during a non-pipelined bus cycle and during a pipelined bus cycle, respectively.

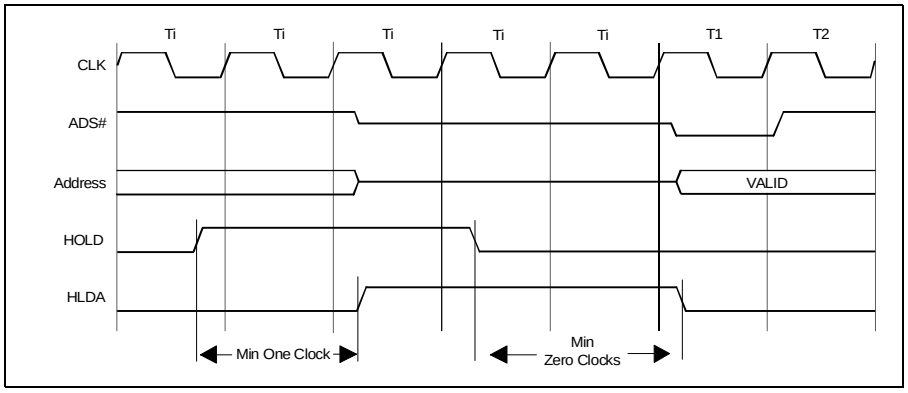

Figure 3-18. Requesting Hold from an Idle Bus

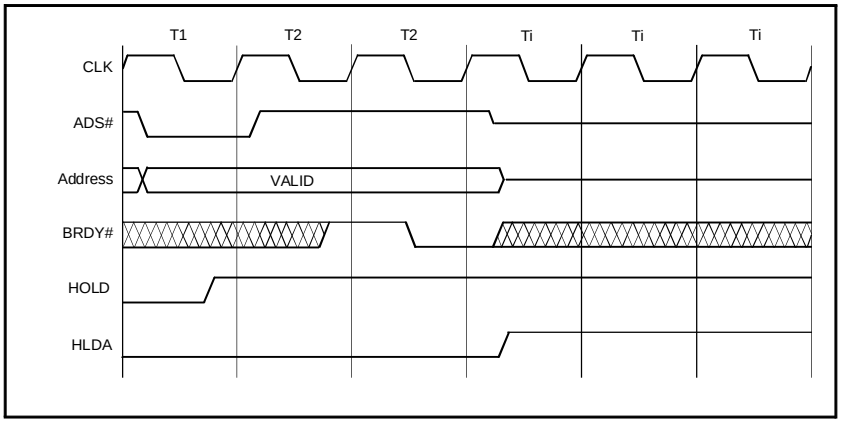

Figure 3-19. Requesting Hold During a Non-Pipelined Bus Cycle

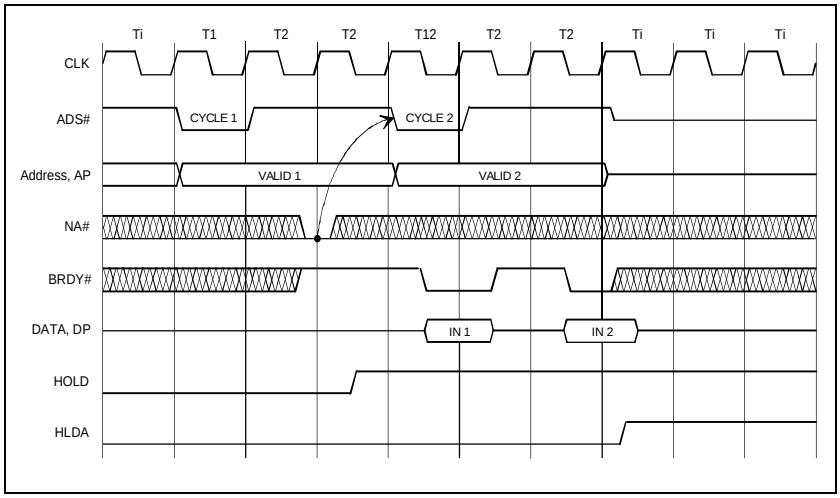

Figure 3-20. Requesting Hold During a Pipelined Bus Cycle

3.3.8.2 Back-Off Timing

An external bus master requests immediate control of the CPU's bus by asserting the back-off (BOFF#) input. The CPU samples BOFF# at each clock edge and responds by floating the bus in the next clock cycle as shown in Figure 3-21. The CPU remains in the bus hold state until BOFF# is negated.

If the assertion of BOFF# interrupts a bus cycle, the bus cycle is restarted in its entirety following the negation of BOFF#. If KEN# was sampled by the processor before the cycle was aborted, it must be returned with the same value during the restarted cycle. The state of WB/WT# may be changed during the restarted cycle.

If BOFF# and BRDY# are active at the same clock edge, the CPU ignores BRDY#. Any data returned to the CPU with the BRDY# is also ignored. If BOFF# interrupts a burst read cycle, the CPU does not cache any data returned prior to BOFF#. However, this data may be used for internal CPU execution.

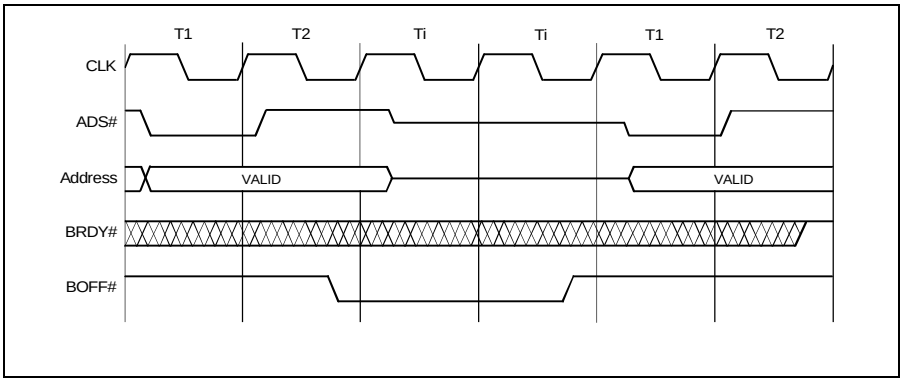

Figure 3-21. Back-Off Timing

3.3.9 Cache Inquiry Cycles

Cache inquiry cycles are issued by the system with the CPU in either a bus hold or address hold state. Bus hold is requested by asserting either HOLD or BOFF#, and address hold is requested by asserting AHOLD. The system initiates the cache inquiry cycle by asserting the EADS# input. The system must also drive the desired inquiry address on the address lines, and a valid state on the INV input.

In response to the cache inquiry cycle, the CPU checks to see if the specified address is present in the internal cache. If the address is present in the cache, the CPU checks the MESI state of the cache line. If the line is in the "exclusive" or "shared" state, the CPU asserts the HIT# output and changes the cache line state to "invalid" if the INV input was sampled logic high with EADS#.

If the line is in the "modified" state, the CPU asserts both HIT# and HITM#. The CPU then issues a bus cycle request to write the modified cache line to external memory. HITM# remains asserted until the write-back bus cycle completes. No additional cache inquiry cycles are accepted while HITM# is asserted. Write-back cycles always start at burst address 0. Once the write-back cycle has completed, the CPU changes the cache line state to "invalid" if the INV input was sampled logic high, or "shared" if the INV input was sampled low.

In addition to checking the cache, the CPU also snoops the internal line fill and cache write-back buffers in response to a cache inquiry cycle. The following sections describe the functional timing for cache inquiry cycles and the corresponding write-back cycles for the various types of inquiry cycles.

3.3.9.1 Inquiry Cycles Using HOLD/HLDA

Figure 3-22 illustrates an inquiry cycle where HOLD is used to force the CPU into a bus hold state. In this case, the system asserts HOLD and must wait for the CPU to respond with HLDA before issuing the cache inquiry cycle. To avoid address bus contention, EADS# should not be asserted until the second clock after HLDA as shown in the diagram. If the inquiry address hits on a modified cache line, HIT# and HITM# are asserted during the second clock following EADS#. Once HITM# asserts, the system must negate HOLD to allow the CPU to run the corresponding write-back cycle. The first cycle issued following negation of HLDA is the write-back bus cycle.

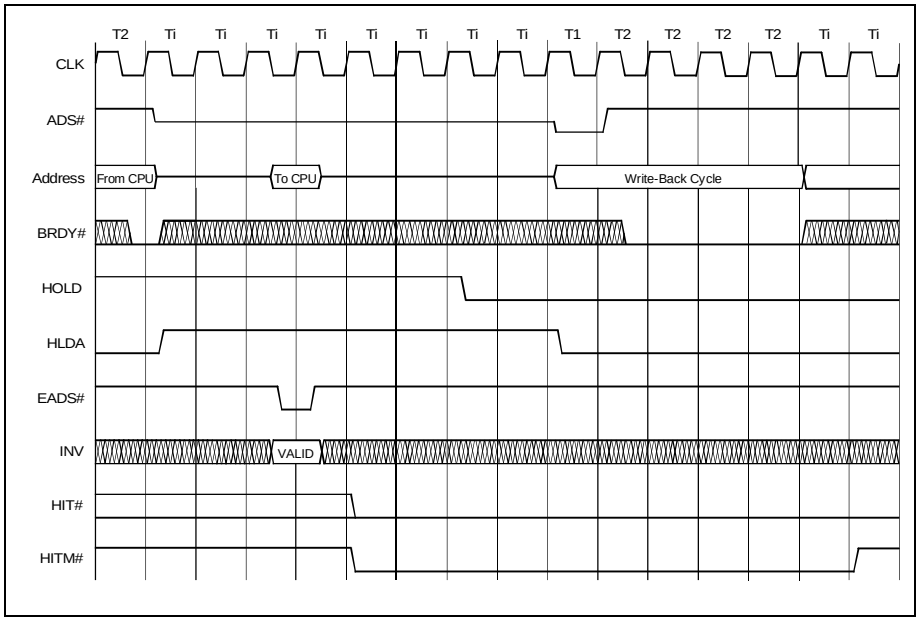

Figure 3-22. HOLD Inquiry Cycle that Hits on a Modified Line

3.3.9.2 Inquiry Cycles Using BOFF#

Figure 3-23 illustrates an inquiry cycle where BOFF# is used to force the CPU into a bus hold state. In this case, the system asserts BOFF# and the CPU immediately relinquishes control of the bus in the next clock. To avoid address bus contention, EADS# should not be asserted until the second clock edge after BOFF# as shown in the diagram. If the inquiry address hits on a modified cache line, HIT# and HITM# are asserted during the second clock following EADS#. Once HITM# asserts, the system must negate BOFF# to allow the CPU to run the corresponding write-back cycle. The first cycle issued following negation of BOFF# is the write-back bus cycle.

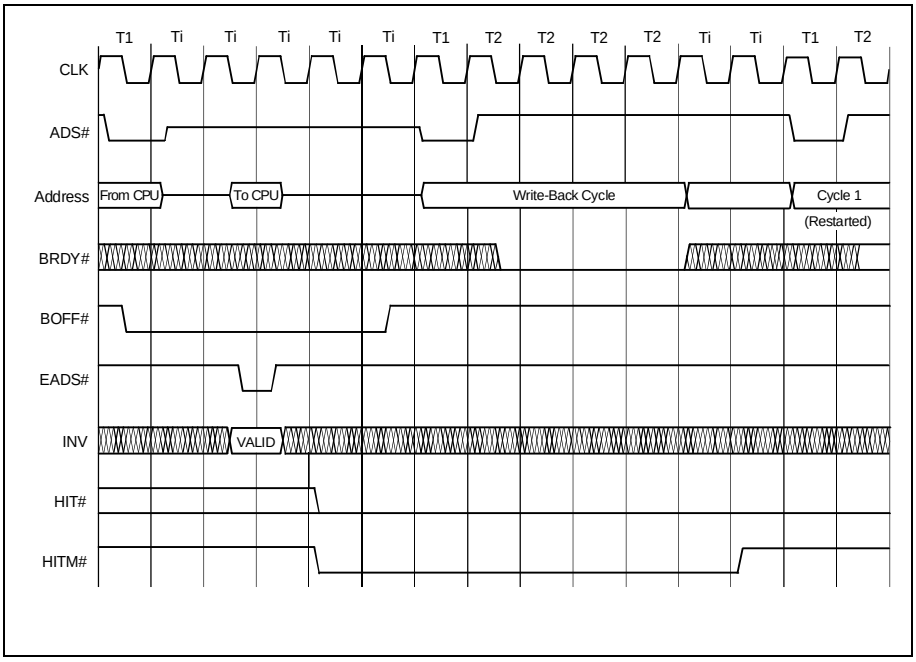

Figure 3-23. BOFF# Inquiry Cycle that Hits on a Modified Line

3.3.9.3 Inquiry Cycles Using AHOLD

Figure 3-24 illustrates an inquiry cycle where AHOLD is used to force the CPU into an address hold state. In this case, the system asserts AHOLD and the CPU immediately floats the address bus in the next clock. To avoid address bus contention, EADS# should not be asserted until the second clock edge after AHOLD as shown in the diagram. If the inquiry address hits on a modified cache line, the CPU asserts HIT# and HITM# during the second clock following EADS#. The CPU then issues the write-back cycle even if AHOLD remains asserted. ADS# for the write-back cycle asserts two clocks after HITM# is asserted. To prevent the address bus and data bus from switching simultaneously, the system must adhere to the restrictions on negation of AHOLD as shown in Figure 3-24.

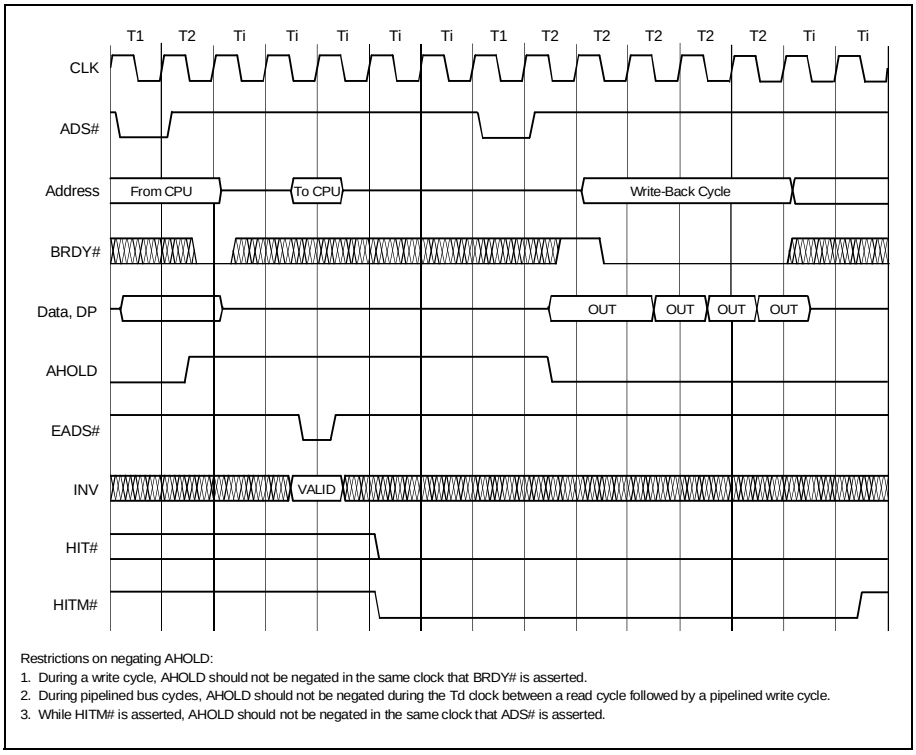

Figure 3-24. AHOLD Inquiry Cycle that Hits on a Modified Line

Figure 3-25 depicts an AHOLD inquiry cycle during a line fill. In this case, the write-back cycle occurs after the line fill is completed. At least one idle clock exists between the final BRDY# of the line fill and the ADS# for the write-back cycle. If the inquiry cycle hits on the address of the line fill that is in progress, the data from the line fill cycle is always used to complete the pending internal operation. However, the data is not placed in the cache if INV is sampled asserted with EADS#. The data is placed in the cache in a "shared" state if INV is sampled negated.

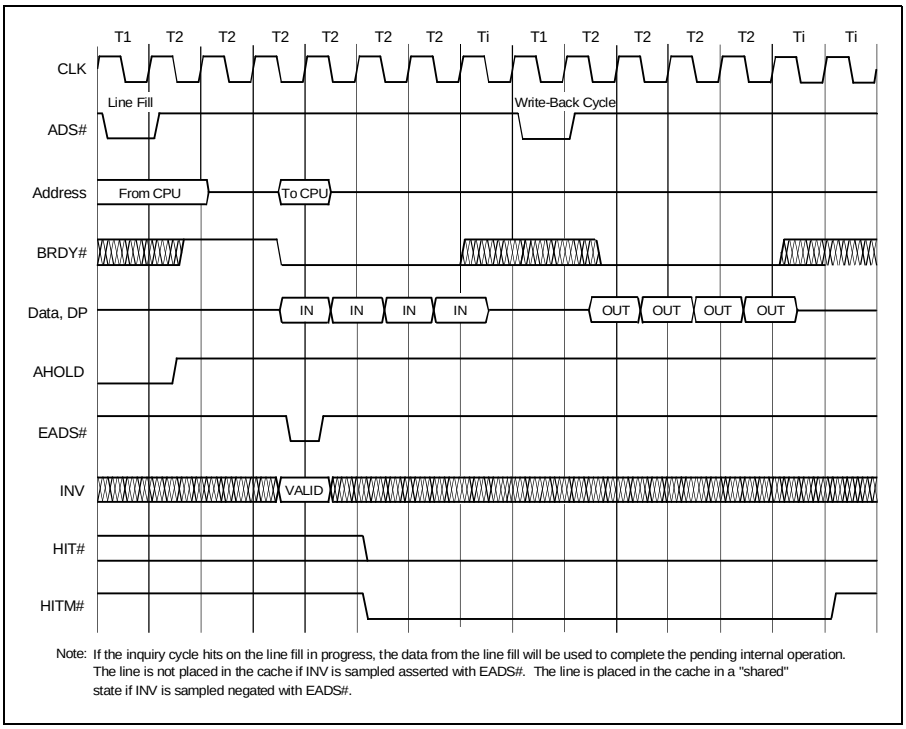

Figure 3-25. AHOLD Inquiry Cycle During a Line Fill

During cache inquiry cycles, the CPU performs address parity checking using A31-A5 and the AP signal. The CPU checks for even parity and asserts the APCHK# output if a parity error is detected. Figure 3-26 illustrates the functional timing of the APCHK# output.

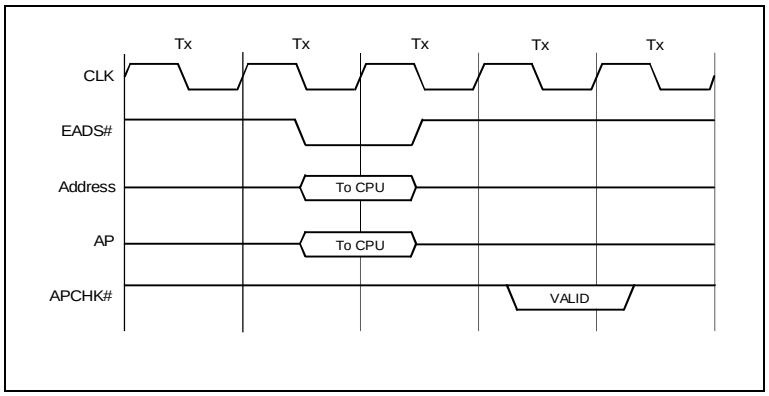

Figure 3-26. APCHK# Timing

3.3.10 Scatter/Gather Buffer Interface

The scatter/gather buffer interface signals, in conjunction with the byte enables and address hold, can be used by the system hardware to transfer data to/from a 32-bit peripheral interface bus. A 64-bit buffer resides in the CPU to assist the system in these transfers.

As shown in Figure 3-27 when BHOLD is asserted the CPU floats the byte enable outputs (BE7#-BE0#) in the next clock. While BHOLD is asserted, only the byte enables are disabled. The current bus cycle remains active and can be completed in the normal fashion. The CPU continues to generate additional bus cycles while BHOLD is asserted, so BHOLD should only be asserted while AHOLD is asserted.

Figure 3-27 also illustrates DHOLD timing. DHOLD forces the CPU to float the data and data parity buses in the next clock. While DHOLD is asserted, the current bus cycle remains active and additional bus cycles may be generated by the CPU.

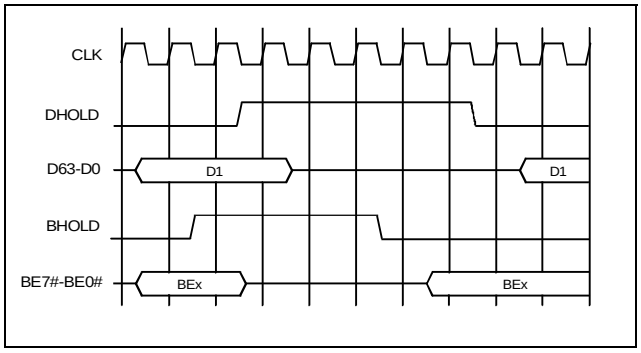

Figure 3-27. BHOLD and DHOLD Timing

Figures 3-28 and 3-29 (Page 3-58) illustrate CPU read and write cycles that access a 32-bit device using the scatter/gather buffer.

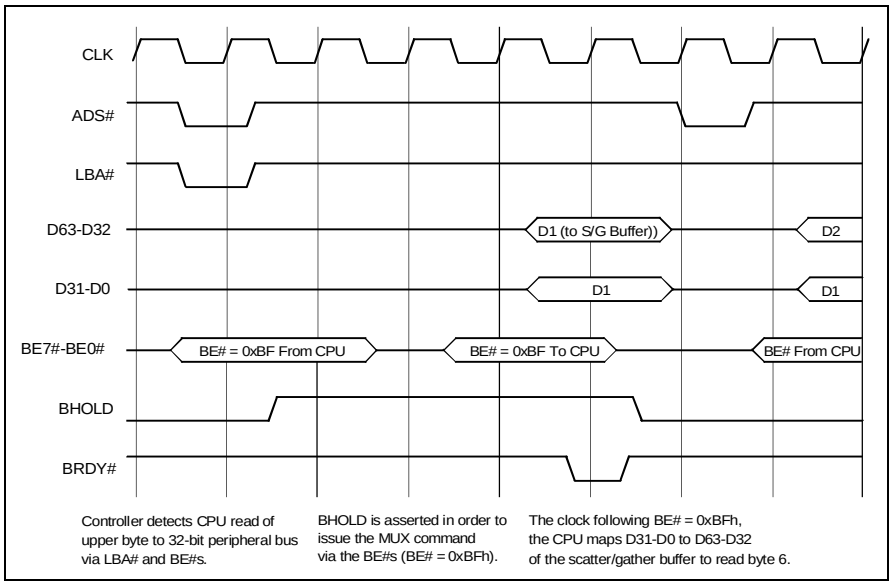

Figure 3-28. CPU Upper Byte Read from 32-Bit Bus Using Scatter/Gather

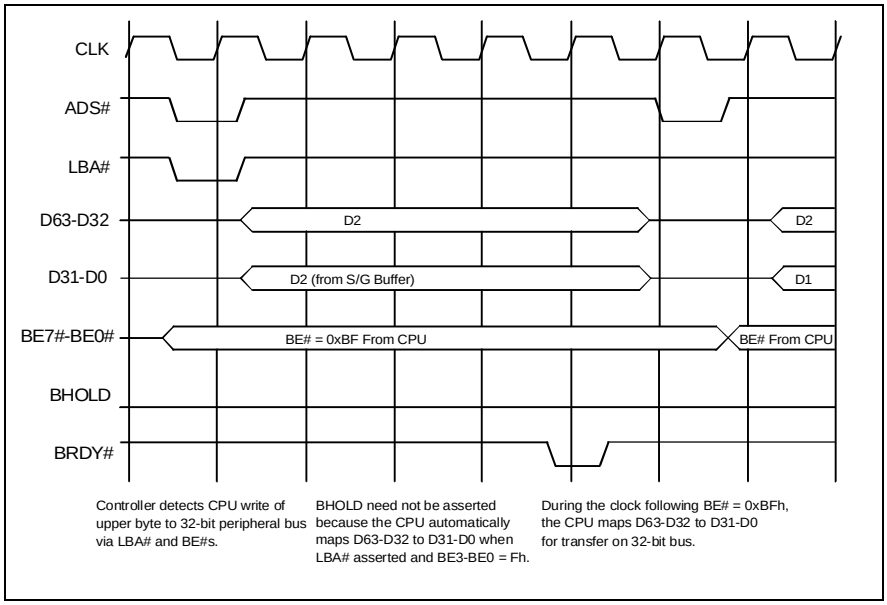

Figure 3-29. CPU Upper Byte Write to 32-Bit Bus Using Scatter/Gather

Figures 3-30 and 3-31 (Page 3-60) illustrate bus master reads and writes between a 32-bit device and 64-bit main memory. The CPU bus must be idle when a bus master initiates a scatter/gather cycle.

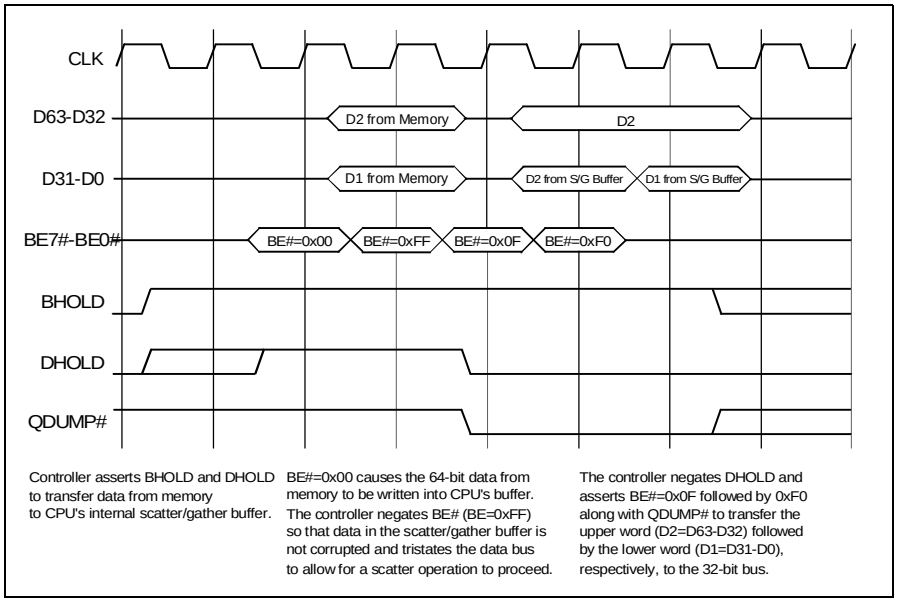

Figure 3-30. Bus Master Read from 64-Bit Memory to 32-Bit Bus

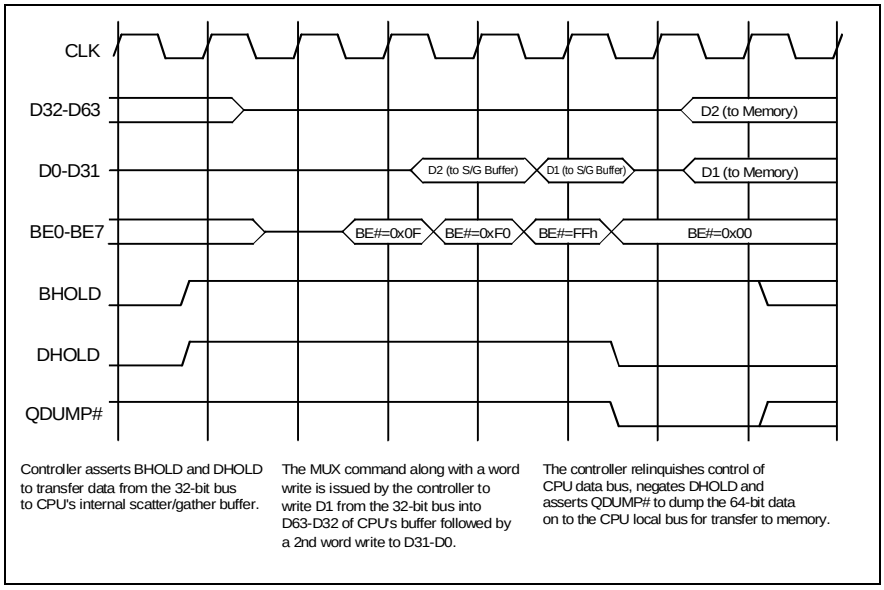

Figure 3-31. Bus Master Write to 64-Bit Memory from 32-Bit Bus

3.3.11 Power Management Interface

SUSP# Initiated Suspend Mode

The 6x86 CPU enters suspend mode when the SUSP# input is asserted and execution of the current instruction, any pending decoded instructions and associated bus cycles are completed. A stop grant bus cycle is then issued and the SUSPA# output is asserted. The CPU responds to SUSP# and asserts SUSPA# only if the SUSP bit is set in the CCR2 configuration register.

SUSP# is sampled (Figure 3-32) on the rising edge of CLK. SUSP# must meet specified setup and hold times to be recognized at a particular CLK edge. The time from assertion of SUSP# to activation of SUSPA# varies depending on which instructions were decoded prior to assertion of SUSP#. The minimum time from SUSP# sampled active to SUSPA# asserted is eight CLKs. As a maximum, the CPU may execute up to two instructions and associated bus cycles prior to asserting SUSPA#. The time required for the CPU to deactivate SUSPA# once SUSP# has been sampled inactive is five CLKs.

If the CPU is in a hold acknowledge state and SUSP# is asserted, the CPU may or may not enter suspend mode depending on the state of the CPU internal execution pipeline. If the CPU is in a SUSP# initiated suspend mode, one occurrence of NMI, INTR and SMI# is stored for execution once suspend mode is exited. The 6x86 CPU also recognizes and acknowledges the HOLD, AHOLD, BOFF# and FLUSH# signals while in suspend mode.

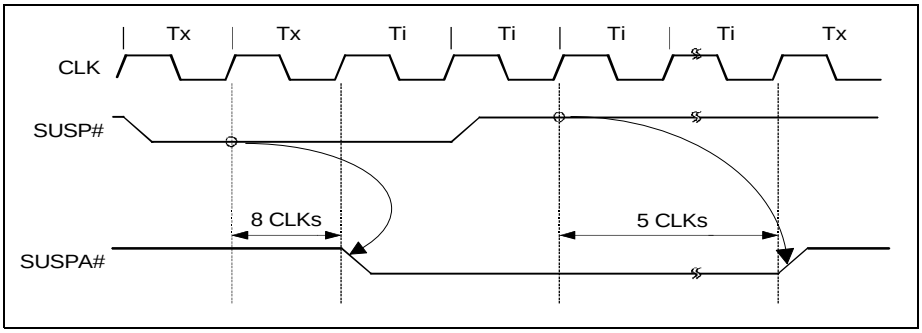

Figure 3-32. SUSP# Initiated Suspend Mode

HALT Initiated Suspend Mode

The CPU also enters suspend mode as a result of executing a HALT instruction if the SUSP HALT bit in CCR2 is set. The SUSPA# output is asserted no later than 40 CLKs following BRDY# sampled active for the HALT bus cycle as shown in Figure 3-33. Suspend mode is then exited upon recognition of an NMI, an unmasked INTR or an SMI#. SUSPA# is deactivated 10 CLKs after sampling of an active interrupt.

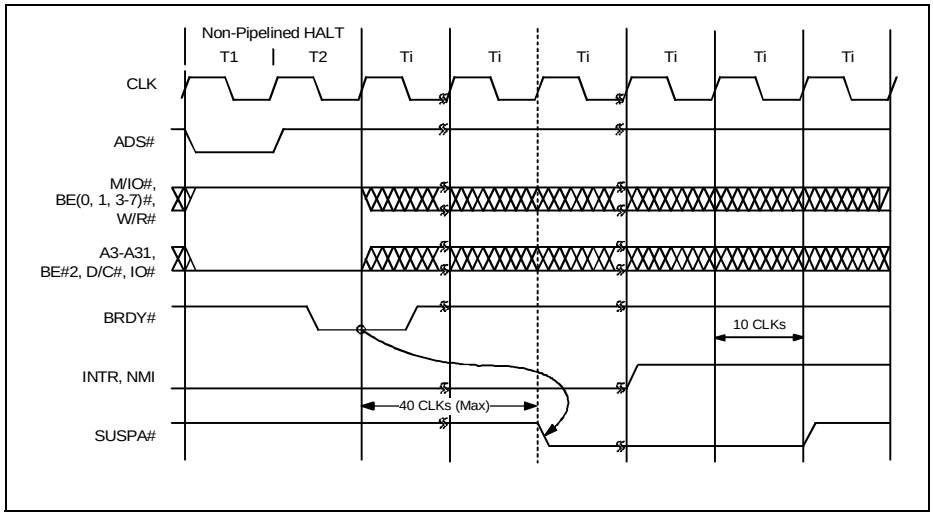

Figure 3-33. HALT Initiated Suspend Mode

Stopping the Input Clock

Once the CPU has entered suspend mode, the input clock (CLK) can be stopped and restarted without loss of any internal CPU data. The CLK input can be stopped at either a logic high or logic low state.

The CPU remains suspended until CLK is restarted and suspend mode is exited as described earlier. While the CLK is stopped, the CPU can no longer sample and respond to any input stimulus.

Figure 3-34 illustrates the recommended sequence for stopping the CLK using SUSP# to initiate suspend mode. CLK may be started prior to or following negation of the SUSP# input. The system must allow sufficient time for the CPU's internal PLL to lock to the desired frequency before exiting suspend mode.

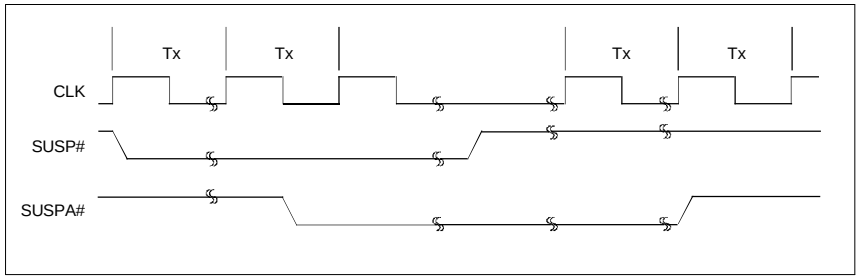

Figure 3-34. Stopping CLK During Suspend Mode

NOTICE TO CUSTOMERS: Some of the information contained in this document was obtained through a third party and IBM has not conducted independent tests of all product characteristics contained herein. The product described in this document is sold under IBM's standard warranty.

The information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not effect or change IBM's product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All the information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for any damages arising directly or indirectly from any use of the information contained in this document.

Printed in the United States of America
2-96

IBM Corporation
1000 River Street
Essex Junction, Vermont 05452-4299
United States of America