Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Active
Core Processor	68040
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	25MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	184-BCQFP
Supplier Device Package	184-CQFP (31.3x31.3)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68ec040fe25a

LIST OF ILLUSTRATIONS

The user programming model includes eight data registers, seven address registers, and a stack pointer register. The address registers and stack pointer can be used as base address registers or software stack pointers, and any of the 16 registers can be used as index registers. Two control registers are available in the user mode—the program counter (PC), which usually contains the address of the instruction that the MC68040 is executing, and the lower byte of the SR, which is accessible as the condition code register (CCR). The CCR contains the condition codes that reflect the results of a previous operation and can be used for conditional instruction execution in a program.

The supervisor programming model includes the upper byte of the SR, which contains operation control information. The vector base register (VBR) contains the base address of the exception vector table, which is used in exception processing. The source function code (SFC) and destination function code (DFC) registers contain 3-bit function codes. These function codes can be considered extensions to the 32-bit logical address. The processor automatically generates function codes to select address spaces for data and program accesses in the user and supervisor modes. Some instructions use the alternate function code registers to specify the function codes for various operations.

The cache control register (CACR) controls enabling of the on-chip instruction and data caches of the MC68040. The supervisor root pointer (SRP) and user root pointer (URP) registers point to the root of the address translation table tree to be used for supervisor and user mode accesses.

The translation control register (TCR) enables logical-to-physical address translation and selects either 4- or 8-Kbyte page sizes. There are four transparent translation registers, two for instruction accesses and two for data accesses. These registers allow portions of the logical address space to be transparently mapped and accessed without the use of resident descriptors in an ATC. The MMU status register (MMUSR) contains status information derived from the execution of a PTEST instruction. The PTEST instruction searches the translation tables for the logical address, specified by this instruction's effective address field and the DFC, and returns status information corresponding to the translation.

The user programming model can also access the entire floating-point programming model. The eight 80-bit floating-point data registers are analogous to the integer data registers. A 32-bit floating-point control register (FPCR) contains an exception enable byte that enables and disables traps for each class of floating-point exceptions and a mode byte that sets the user-selectable rounding and precision modes. A floating-point status register (FPSR) contains a condition code byte, quotient byte, exception status byte, and accrued exception byte. A floating-point exception handler can use the address in the 32-bit floating-point instruction address register (FPIAR) to locate the floating-point instruction that has caused an exception. Instructions that do not modify the FPIAR can be used to read the FPIAR in the exception handler without changing the previous value.

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
BTST	–(bit number of Destination) ø Z;	BTST Dn,<ea> BTST #<data>,<ea>
CAS	CAS Destination – Compare Operand ø cc; if Z, Update Operand ø Destination else Destination ø Compare Operand	CAS Dc,Du,<ea>
CAS2	CAS2 Destination 1 – Compare 1 ø cc; if Z, Destination 2 – Compare ø cc; if Z, Update 1 ø Destination 1; Update 2 ø Destination 2 else Destination 1 ø Compare 1; Destination 2 ø Compare 2	CAS2 Dc1–Dc2,Du1–Du2,(Rn1)–(Rn2)
CHK	If Dn < 0 or Dn > Source then TRAP	CHK <ea>,Dn
CHK2	If Rn < LB or If Rn > UB then TRAP	CHK2 <ea>,Rn
CINV	If supervisor state then invalidate selected cache lines else TRAP	CINVL <caches>, (An) CINVP <caches>, (An) CINVA <caches>
CLR	0 ø Destination	CLR <ea>
CMP	Destination – Source ø cc	CMP <ea>,Dn
CMPA	Destination – Source	CMPA <ea>,An
CMPI	Destination – Immediate Data	CMPI #<data>,<ea>
CMPM	Destination – Source ø cc	CMPM (Ay)+,(Ax)+
CMP2	Compare Rn < LB or Rn > UB and Set Condition Codes	CMP2 <ea>,Rn
CPUSH	If supervisor state then if data cache push selected dirty data cache lines; invalidate selected cache lines else TRAP	CPUSHL <caches>, (An) CPUSHP <caches>, (An) CPUSHA <caches>
DBcc	If condition false then (Dn–1 ø Dn; If Dn ≠ –1 then PC + d_n ø PC)	DBcc Dn,<label>
DIVS, DIVSL	Destination ÷ Source ø Destination	DIVS.W <ea>,Dn 32 ÷ 16 ø 16r:16q DIVS.L <ea>,Dq 32 ÷ 32 ø 32q DIVS.L <ea>,Dr:Dq 64 ÷ 32 ø 32r:32q DIVSL.L <ea>,Dr:Dq 32 ÷ 32 ø 32r:32q
DIVU, DIVUL	Destination ÷ Source ø Destination	DIVU.W <ea>,Dn 32 ÷ 16 ø 16r:16q DIVU.L <ea>,Dq 32 ÷ 32 ø 32q DIVU.L <ea>,Dr:Dq 64 ÷ 32 ø 32r:32q DIVUL.L <ea>,Dr:Dq 32 ÷ 32 ø 32r:32q
EOR	Source ⊕ Destination ø Destination	EOR Dn,<ea>
EORI	Immediate Data ⊕ Destination ø Destination	EORI #<data>,<ea>
EORI to CCR	Source ⊕ CCR ø CCR	EORI #<data>,CCR
EORI to SR	If supervisor state then Source ⊕ SR ø SR else TRAP	EORI #<data>,SR

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
FNEG[2]	–(Source) ø FPn	FNEG.<fmt> <ea>,FPn FNEG.X FPm,FPn FNEG.X FPn FrNEG.<fmt> <ea>,FPn[3] FrNEG.X FPm,FPn[3] FrNEG.X FPn[3]
FNOP[2]	None	FNOP
FRESTORE[2]	If in supervisor state then FPU State Frame ø Internal State else TRAP	FRESTORE <ea>
FSAVE[2]	If in supervisor state then FPU Internal State ø State Frame else TRAP	FSAVE <ea>
FScc[2]	If condition true then 1s ø Destination else 0s ø Destination	FScc.SIZE <ea>
FSGLDIV	FPn ÷ Source ø FPn	FSGLDIV.<fmt> <ea>,FPn FSGLDIV.X FPm,FPn
FSGLMUL	Source × FPn ø FPn	FSGMUL.<fmt> <ea>,FPn FSGLMUL.X FPm, FPn
FSQRT[2]	Square Root of Source ø FPn	FSQRT.<fmt> <ea>,FPn FSQRT.X FPm,FPn FSQRT.X FPn FrSQRT.<fmt> <ea>,FPn[3] FrSQRT FPm,FPn[3] FrSQRT FPn[3]
FSUB[2]	FPn – Source ø FPn	FSUB.<fmt> <ea>,FPn FSUB.X FPm,FPn FrSUB.<fmt> <ea>,FPn[3] FrSUB.X FPm,FPn[3]
FTRAPcc[2]	If condition true then TRAP	FTRAPcc FTRAPcc.W #<data> FTRAPcc.L #<data>
FTST[2]	Condition Codes for Operand ø FPCC	FTST.<fmt> <ea> FTST.X FPm
ILLEGAL	SSP – 2 ø SSP; Vector Offset ø (SSP); SSP – 4 ø SSP; PC ø (SSP); SSp – 2 ø SSP; SR ø (SSP); Illegal Instruction Vector Address ø PC	ILLEGAL
JMP	Destination Address ø PC	JMP <ea>
JSR	SP – 4 ø SP; PC ø (SP) Destination Address ø PC	JSR <ea>
LEA	<ea> ø An	LEA <ea>,An
LINK	SP – 4 ø SP; An ø(SP) SP ø An, SP+d øSP	LINK An,d_n

S—Supervisor Protection

This bit is set if the S-bit in the page descriptor is set. Setting this bit does not indicate that a violation has occurred.

CM—Cache Mode

This 2-bit field is copied from the CM bits in the page descriptor.

M—Modified

This bit is set if the M-bit is set in the page descriptor associated with the address.

W—Write Protect

This bit is set if the W-bit is set in any of the descriptors encountered during the table search. Setting this bit does not indicate that a violation has occurred.

T—Transparent Translation Register Hit

If the T-bit is set, then the PTEST address matches an instruction or data TTR, the R-bit is set, and all other bits are zero.

R—Resident

The R-bit is set if the PTEST address matches an instruction or data TTR or if the table search completes by obtaining a valid page descriptor.

3.2 LOGICAL ADDRESS TRANSLATION

The function of the MMUs is to translate logical addresses to physical addresses. The MMUs perform translations according to control information in translation tables. The operating system creates these translation tables and stores them in memory. The processor then fetches a translation table as needed and stores it in an ATC.

3.2.1 Translation Tables

The M68040 uses the ATCs in the instruction and data memory units with translation tables stored in memory to perform the translations from logical to physical addresses. The operating system loads the translation tables for a program into memory. No distinction is made in the translation of instruction accesses versus data accesses because the instruction and data MMUs access the same translation table for a specific privilege mode, either user or supervisor. This lack of distinction results in a merged instruction and data address space.

Figure 3-7 illustrates the three-level tree structure of a general translation table supported by the M68040. The root- and pointer-level tables contain the base addresses of the tables at the next level. The page-level tables contain either the physical address for the translation or a pointer to the memory location containing the physical address. Only a portion of the translation table for the entire logical address space is required to be resident in memory at any time—specifically, only the portion of the table that translates

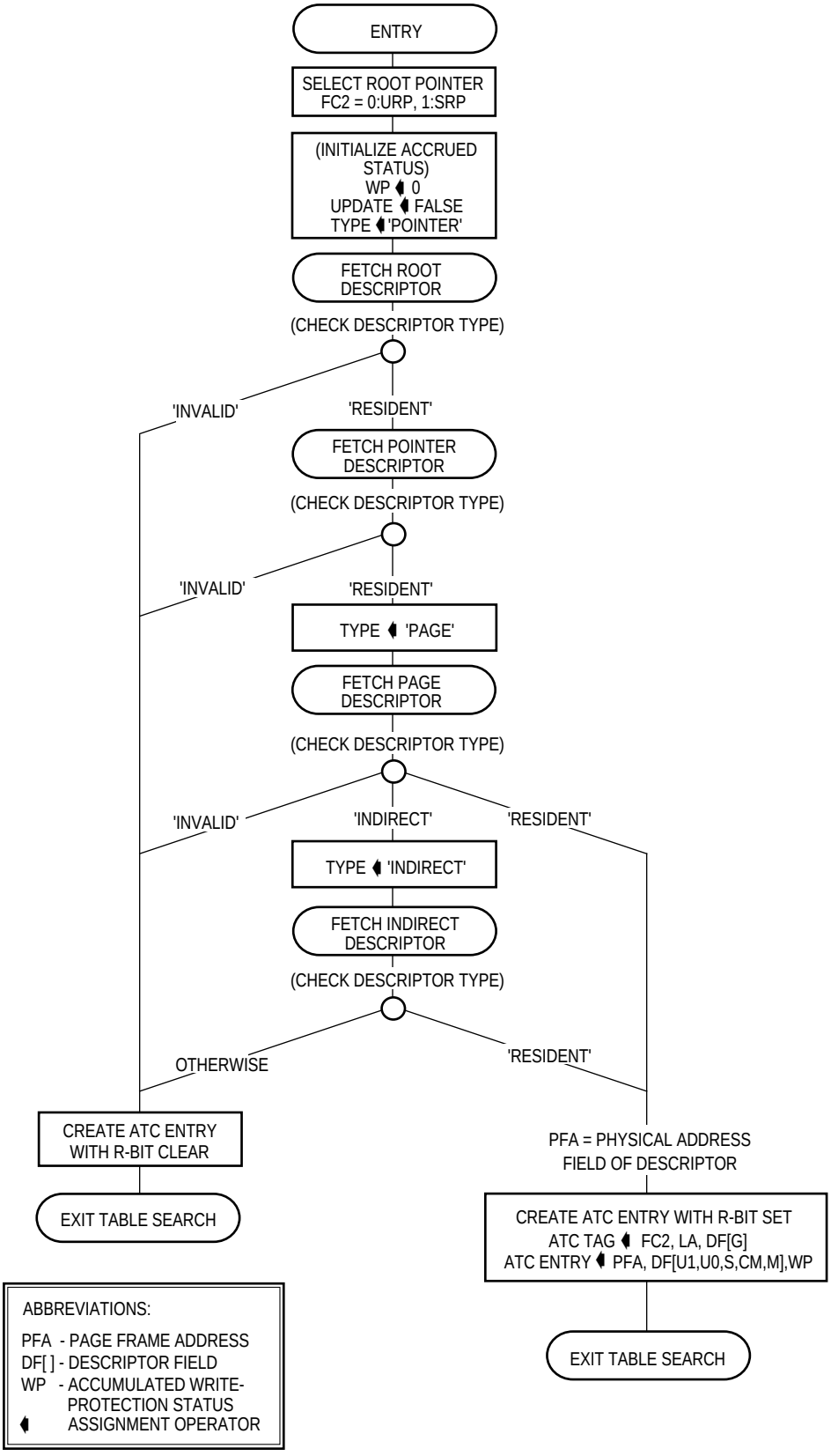


Figure 3-9. Detailed Flowchart of Table Search Operation

4.2 CACHE MANAGEMENT

Using the MOVEC instruction, the caches are individually enabled to access the 32-bit cache control register (CACR) illustrated in Figure 4-4. The CACR contains two enable bits that allow the instruction and data caches to be independently enabled or disabled. Setting one of these bits enables the associated cache without affecting the state of any lines within the cache. A hardware reset clears the CACR, disabling both caches; however, reset does not affect the tags, state information, and data within the caches. The CINV instruction must clear the caches before enabling them. It is not recommended that page descriptors be cached. Specifically, the M68040 does not support the caching of page descriptors in copyback mode with the bit pattern U = 0, M = 1, and R = 1 in a page descriptor. The M68040 table search algorithm will never leave this bit pattern for a page descriptor.

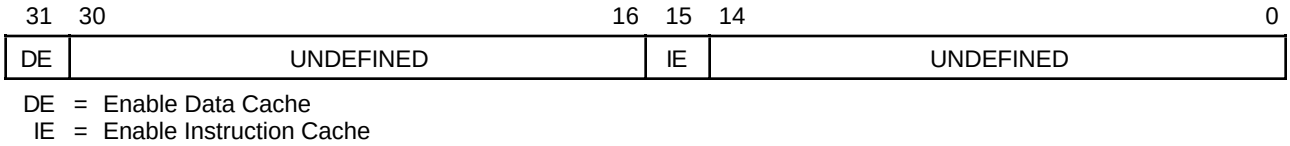


Figure 4-4. Cache Control Register

System hardware can assert the cache disable (CDIS) signal to dynamically disable both caches, regardless of the state of the enable bits in the CACR. The caches are disabled immediately after the current access completes. If CDIS is asserted during the access for the first half of a misaligned operand spanning two cache lines, the data cache is disabled for the second half of the operand. Accesses by the execution units bypass the caches while they are disabled and do not affect their contents (with the exception of CINV and CPUSH instructions). Disabling the caches with CDIS does not affect snoop operations. CDIS is intended primarily for use by in-circuit emulators to allow swapping between the tags and emulator memories.

Even if the instruction cache is disabled, the M68040 can cache instructions because of an internal cache line register. This happens for instruction loops that are completely resident within the first six bytes of a half-line. Thus, the cache line holding register can operate as a small cache. If a loop fits anywhere within the first three words of a half-line, then it becomes cached.

The CINV and CPUSH instructions support cache management in the supervisor mode. CINV allows selective invalidation of cache entries. CPUSH performs two operations: 1) any selected data cache lines containing dirty data are pushed to memory; 2) all selected cache lines are invalidated. This operation can be used to update a page in memory before swapping it out with snooping disabled or to push dirty data when changing a page caching mode to write-through. Because of the size of the caches, pushing pages or an entire cache incurs a significant time penalty. However, these instructions are interruptable to avoid large interrupt latencies. The state of the CDIS signal or the cache enable bits in the CACR does not affect the operation of CINV and CPUSH. Both instructions allow operation on a single cache line, all cache lines in a specific page, or an

Table 5-7 Signal Summary (Continued)

Signal Name	Mnemonic	Type	Active	Three-State
Transfer Start	TS	Input/Output	Low	Yes
Transfer Type	TT1, TT0	Input/Output	High	Yes
Test Clock	TCK	Input	—	—
Test Data Input	TDI	Input	High	—
Test Data Output	TDO	Output	High	Yes
Test Mode Select	TMS	Input	High	—
Test Reset	TRST	Input	Low	—
User-Programmable Attributes	UPA1, UPA0	Output	High	Yes
Power Supply	V_{CC}	Power	—	—

NOTES:
1. This signal is not available on the MC68LC040 and MC68EC040.
2. These signals are different on power-up for the MC68LC040 and MC68EC040.
3. This signal is not available on the MC68EC040.

All M68040 bidirectional pins include two boundary scan data cells, an input, and an output. One of five associated boundary scan control cells controls each bidirectional pin. If these cells contain a logic one, the associated bidirectional or three-state pin will be configured as an output and enabled. The cell captures the current value during the capture-DR state. All five control cells are reset (i.e., logic zero) in the test-logic-reset state. The five bidirectional/three-state control cells and their boundary scan register bit positions are as follows:

Cell Name	Bit
io.ab	150
io.db	151
io.2	154
io.1	155
io.0	156

Table 6-2 lists the 184 boundary scan bit definitions. The first column in the table defines the bit position in the boundary scan register. The second column references one of the three cell types. The third column lists the pin name for all pin-related cells. The fourth column lists the system pin type for convenience where TS-Output indicates a three-state output pin and I/O indicates a bidirectional pin. The last column lists the name of the associated control bit of the boundary scan register for three-state output and bidirectional pins. The boundary scan description language (BSDL) type for each cell can be found in note 1.

7.2 DATA TRANSFER MECHANISM

Figure 7-2 illustrates how the bus designates operands for transfers on a byte boundary system. The integer unit handles floating-point operands as a sequence of related long-word operands. These designations are used in the figures and descriptions that follow.

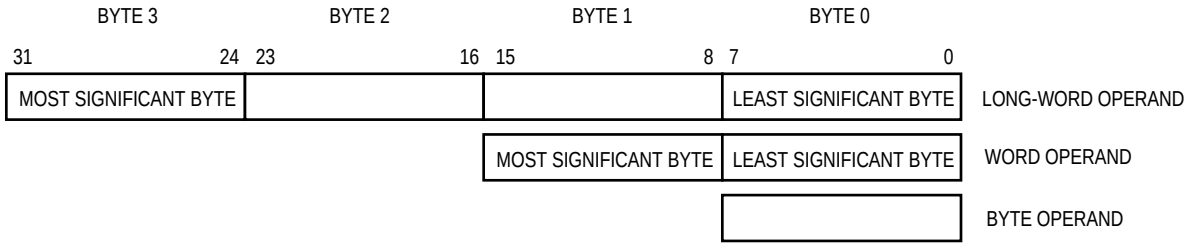


Figure 7-2. Internal Operand Representation

Figure 7-3 illustrates general multiplexing between an internal register and the external bus. The internal register connects to the external data bus through the internal data bus and multiplexer. The data multiplexer establishes the necessary connections for different combinations of address and data sizes.

Unlike the MC68020 and MC68030 processors, the M68040 does not support dynamic bus sizing and expects the referenced device to accept the requested access width. The MC68150 dynamic bus sizer is designed to allow the 32-bit M68040, MC68EC040, MC68LC040 bus to communicate bidirectionally with 32-, 16-, or 8-bit peripherals and memories. It dynamically recognizes the size of the selected peripheral or memory device and then reads or writes the appropriate data from that location. Refer to MC68150/D, *MC68150 Dynamic Bus Sizer*, for information on this device.

Blocks of memory that must be contiguous, such as for code storage or program stacks, must be 32 bits wide. Byte- and word-sized I/O ports that return an interrupt vector during interrupt acknowledge cycles must be mapped into the low-order 8 or 16 bits, respectively, of the data bus.

The multiplexer takes the four bytes of the 32-bit bus transfer and routes them to their required positions. For example, byte 0 would normally be routed to D31–D24, but it can also be routed to any other byte position supporting a misaligned data transfer. The same is true for any of the other operand bytes. The transfer size (SIZ0 and SIZ1) and byte offset (A1 and A0) signals determine the positioning of the bytes (see Table 7-1). The size indicated on the SIZx signals corresponds to the size of the operand transfer for the entire bus cycle. During an operand transfer, A31–A2 indicate the long-word base address for the first byte of the operand to be accessed; A1 and A0 indicate the byte offset from the base. For a burst-inhibited line transfer, A1 and A0 for each of the four accesses (the burst-inhibited line transfer and three long-word transfers) are copied from the lowest two bits of the access address used to initiate the line transfer.

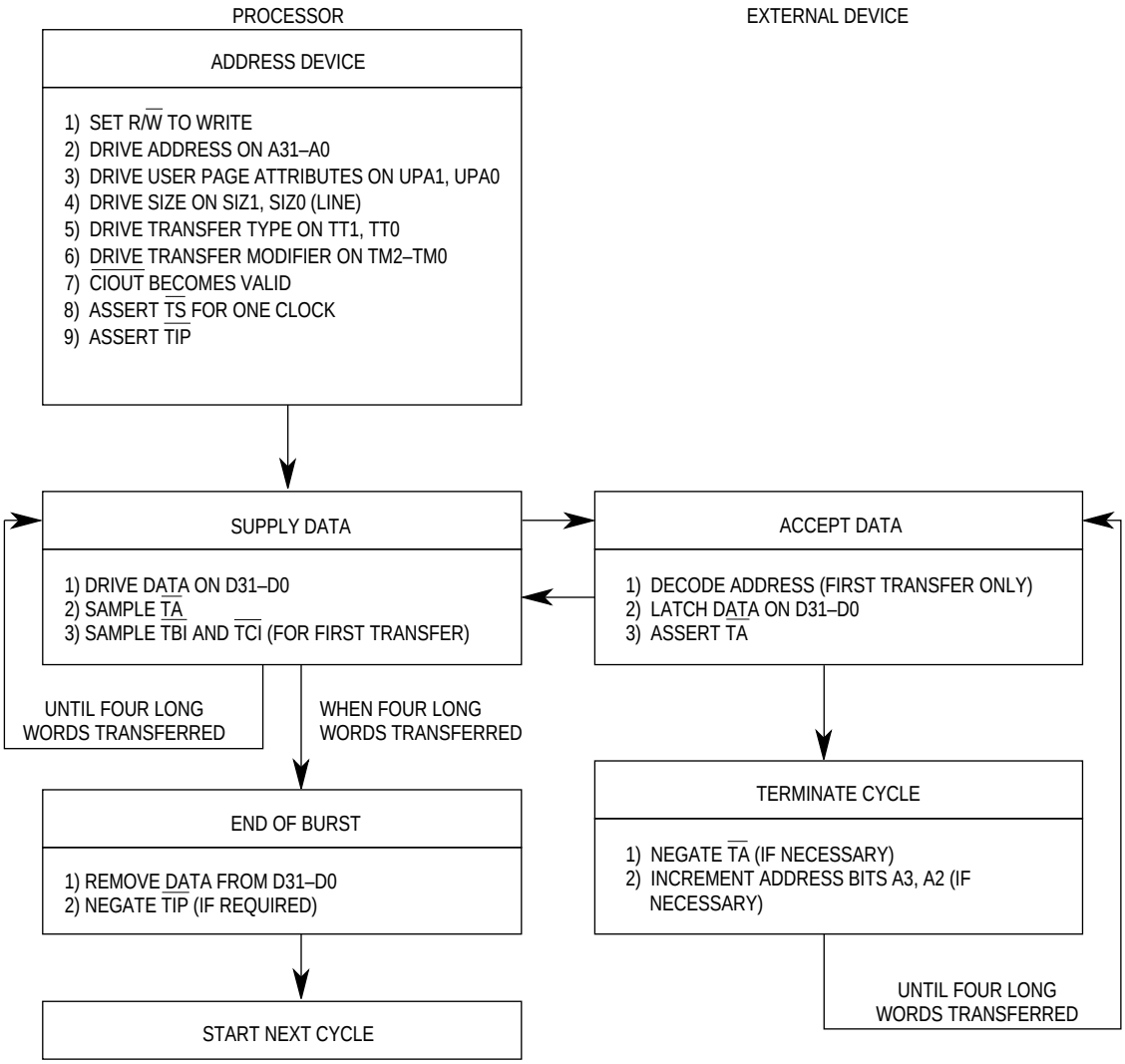
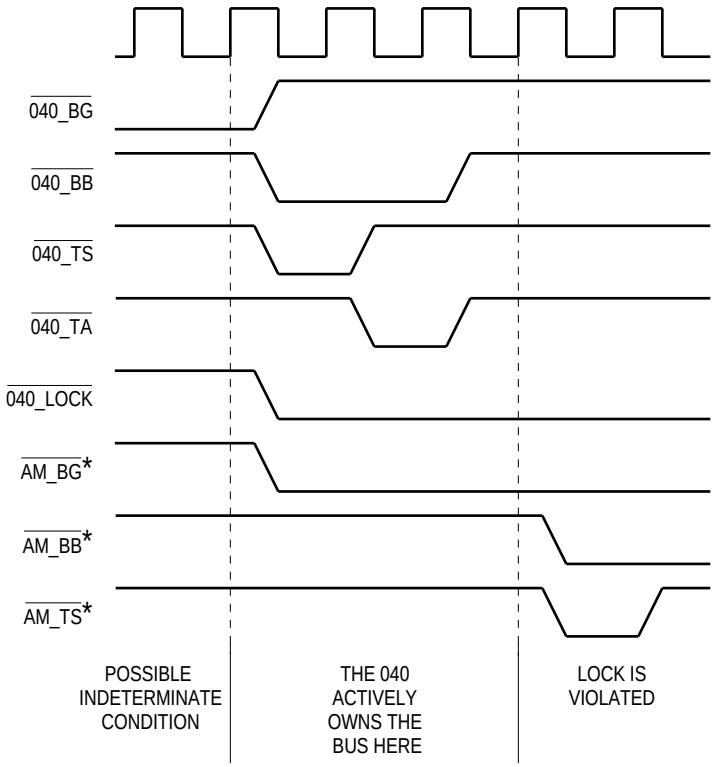


Figure 7-16. Line Write Transfer Flowchart



*AM indicates the alternate bus master.

Figure 7-31. Lock Violation Example

In addition to the indeterminate condition, the external arbiter's design needs to include the function of $\overline{BR}$. For example, in certain cases associated with conditional branches, the M68040 can assert $\overline{BR}$ to request the bus from an alternate bus master, then negate $\overline{BR}$ without using the bus, regardless of whether or not the external arbiter eventually asserts $\overline{BG}$. This situation happens when the M68040 attempts to prefetch an instruction for a conditional branch. To achieve maximum performance, the processor prefetches the instructions of both paths for a conditional branch. If the conditional branch results in a branch-not-taken, the previously issued branch-taken prefetch is then terminated since the prefetch is no longer needed. In an attempt to save time, the M68040 negates $\overline{BR}$. If $\overline{BG}$ takes too long to assert, the M68040 enters a disregard request condition.

The $\overline{BR}$ signal can be reasserted immediately for a different pending bus request, or it can stay negated indefinitely. If an external bus arbiter is designed to wait for the M68040 to assert $\overline{BB}$ before proceeding, then the system experiences an extended period of time in which bus arbitration is locked. Motorola recommends that an external bus arbiter not assume that there is a direct relationship between $\overline{BR}$ and $\overline{BB}$ or $\overline{BR}$ and $\overline{BG}$ signals.

Figure 7-32 illustrates an example of the processor requesting the bus from the external bus arbiter. During C1, the M68040 asserts $\overline{BR}$ to request the bus from the arbiter, which negates the alternate bus master's $\overline{BG}$ signal and grants the bus to the processor by asserting $\overline{BG}$ during C3. During C3, the alternate bus master completes its current access and relinquishes the bus by three-stating all bus signals. Typically, the $\overline{BB}$ and $\overline{TIP}$ signals

For processor resets after the initial power-on reset, $\overline{RSTI}$ should be asserted for at least 10 clock periods. Figure 7-45 illustrates timings associated with a reset when the processor is executing bus cycles. Note that $\overline{BB}$ and $\overline{TIP}$ (and $\overline{TA}$ if driven during a snooped access) are negated before transitioning to a three-state level.

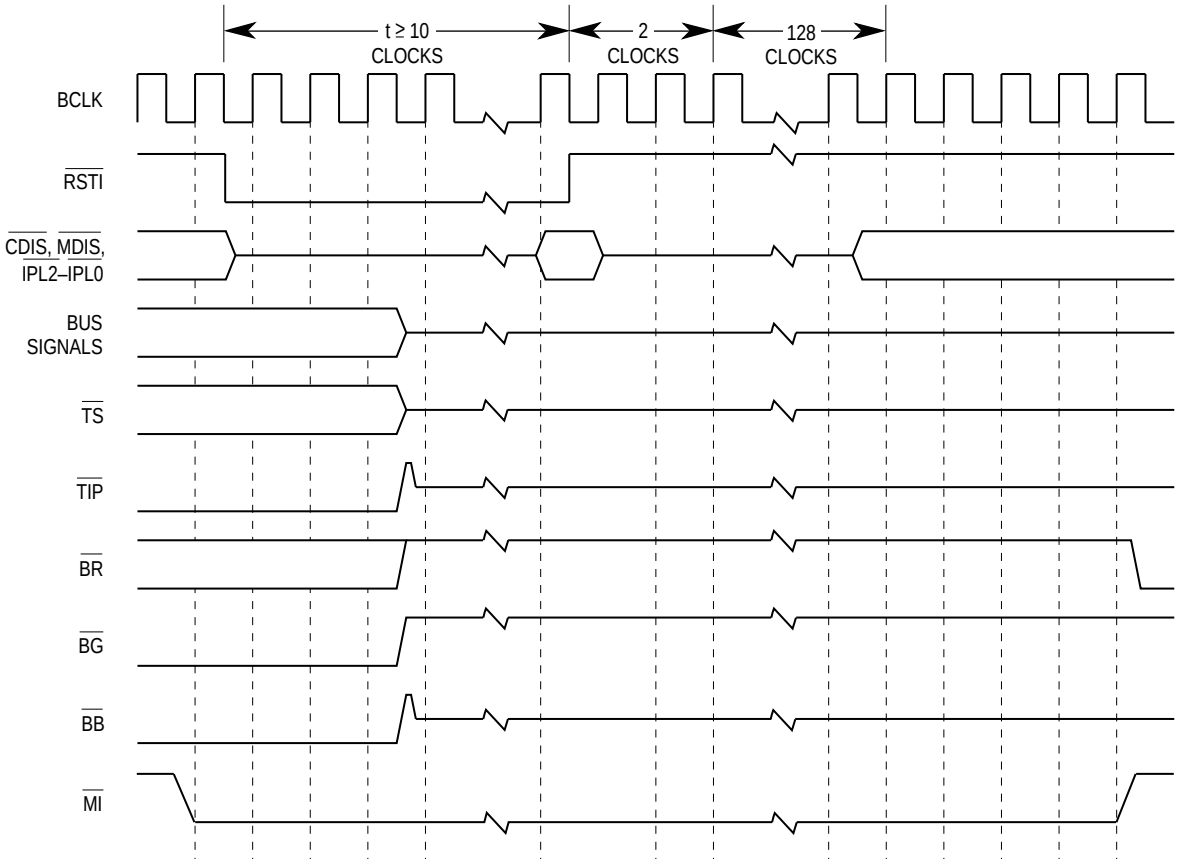


Figure 7-45. Normal Reset Timing

Resetting the processor causes any bus cycle in progress to terminate as if $\overline{TA}$ or $\overline{TEA}$ had been asserted. In addition, the processor initializes registers appropriately for a reset exception. **Section 8 Exception Processing** describes exception processing. When a RESET instruction is executed, the processor drives the reset out ($\overline{RSTO}$) signal for 512 BCLK cycles. In this case, the processor resets the external devices of the system, and the internal registers of the processor are unaffected. The external devices connected to the $\overline{RSTO}$ signal are reset at the completion of the RESET instruction. An $\overline{RSTI}$ signal that is asserted to the processor during execution of a RESET instruction immediately resets the processor and causes the $\overline{RSTO}$ signal to negate. $\overline{RSTO}$ can be logically ANDed with the external signal driving $\overline{RSTI}$ to derive a system reset signal that is asserted for both an external processor reset and execution of a RESET instruction.

Table 9-7. Floating-Point Condition Code Encodings

Data Type	N	Z	I	NAN
+ Normalized or Denormalized	0	0	0	0
– Normalized or Denormalized	1	0	0	0
+ 0	0	1	0	0
– 0	1	1	0	0
+ Infinity	0	0	1	0
– Infinity	1	0	1	0
+ NAN	0	0	0	1
– NAN	1	0	0	1

The inclusion of the NAN data type in the IEEE floating-point number system requires each conditional test to include the NAN condition code bit in its Boolean equation. Because a comparison of a NAN with any other data type is unordered (i.e., it is impossible to determine if a NAN is bigger or smaller than an in-range number), the compare instruction sets the NAN condition code bit when an unordered compare is attempted. All arithmetic instructions also set the FPCC NAN bit if the result of an operation is a NAN. The conditional instructions interpret the NAN condition code bit equal to one as the unordered condition.

The IEEE 754 standard defines four conditions: equal to (EQ), greater than (GT), less than (LT), and unordered (UN). In addition, the standard only requires the generation of the condition codes as a result of a floating-point compare operation. The FPU tests for these conditions and 28 others at the end of any operation affecting the condition codes. For purposes of the floating-point conditional branch, set byte on condition, decrement and branch on condition, and trap on condition instructions, the MC68040 logically combines the four FPCC bits to form 32 conditional tests. The 32 conditional tests are separated into two groups—16 that cause an exception if an unordered condition is present when the conditional test is attempted, IEEE nonaware tests, and 16 that do not cause an exception, IEEE aware tests. The set of IEEE nonaware tests is best used:

- when porting a program from a system that does not support the IEEE 754 standard to a conforming system or
- when generating high-level language code that does not support IEEE floating-point concepts (i.e., the unordered condition).

An unordered condition occurs when one or both of the operands in a floating-point compare operation is a NAN. The inclusion of the unordered condition in floating-point branches destroys the familiar trichotomy relationship (greater than, equal, less than) that exists for integers. For example, the opposite of floating-point branch greater than (FBGT) is not floating-point branch less than or equal (FBLE). Rather, the opposite condition is floating-point branch not greater than (FBNGT). If the result of the previous instruction was unordered, FBNGT is true; whereas, both FBGT and FBLE would be false since unordered fails both of these tests (and sets BSUN). Compiler programmers should be

10.4 MOVE INSTRUCTION TIMING

SOURCE	DESTINATION					
	Dn		(An)		(An)+	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1	1	1	1	1
(An)	1	1	1	1	2	1_L + 1
(An)+	1	1	2	1_L + 1	2	1_L + 1
–(An)	1	1	2	1_L + 1	2	1_L + 1
(d_{16},An)	1	1	2	1_L + 1	2	1_L + 1
(d_{16},PC)	3	2_L + 1	3	2_L + 1	3	2_L + 1
(xxx).W, (xxx).L	1	1	1	1	2	1_L + 1
#<xxx>	1	1	1	1	2	1_L + 1
(d_8,An,Xn)	3	3	4	4	5	5
(d_8,PC,Xn)	5	1_L + 4	5	1_L + 4	6	1_L + 5
(b_{16},BR,Xn)	7	1_L + 6	7	1_L + 6	8	1_L + 7
([bd,BR,Xn])	10	1_L + 9	10	1_L + 9	11	1_L + 10
([bd,BR,Xn],od)	11	1_L + 10	11	1_L + 10	12	1_L + 11
([bd,BR],Xn)	11	3_L + 8	11	3_L + 8	12	3_L + 9
([bd,BR],Xn,od)	12	3_L + 9	12	3_L + 9	13	3_L + 10
	–(An)		(d_{16},An)		(xxx).W, (xxx).L	
Dn	1	1	1	1	1	1
(An)	2	1_L + 1	2	1_L + 1	1	1
(An)+	2	1_L + 1	2	1_L + 1	2	1_L + 1
–(An)	2	1_L + 1	2	1_L + 1	2	1_L + 1
(d_{16},An)	2	1_L + 1	2	1_L + 1	2	1_L + 1
(d_{16},PC)	3	2_L + 1	4	3_L + 1	4	3_L + 1
(xxx).W, (xxx).L	2	1_L + 1	2	1_L + 1	2	1_L + 1
#<xxx>	2	1_L + 1	2	1_L + 1	2	1_L + 1
(d_8,An,Xn)	5	5	5	5	5	5
(d_8,PC,Xn)	6	1_L + 5	6	1_L + 5	6	1_L + 5
(b_{16},BR,Xn)	8	1_L + 7	8	1_L + 7	8	1_L + 7
([bd,BR,Xn])	11	1_L + 10	11	1_L + 10	11	1_L + 10
([bd,BR,Xn],od)	12	1_L + 11	12	1_L + 11	12	1_L + 11
([bd,BR],Xn)	12	3_L + 9	12	3_L + 9	12	3_L + 9
([bd,BR],Xn,od)	13	3_L + 10	13	3_L + 10	13	3_L + 10

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	BCHG, BCLR, BSET[a]		BFCHG, BFCLR, BFSET[b,c]		BFEXTS, BFEXTU[b,d]	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	3/4	3/4[e]	6/7[e]	1/2[e]	4/5[e]
An	—	—	—	—	—	—
(An)	1	3/4	9	2_L + 8	9	2_L + 7
(An)+	1	3/4	—	—	—	—
–(An)	1	3/4	—	—	—	—
(d_{16},An)	2/1	1_L + 3/4	9	2_L + 8	9	2_L + 7
(d_{16},PC)	—	—	—	—	10	3_L + 7
(xxx).W, (xxx).L	2/1	1_L + 3/4	9	2_L + 8	9	2_L + 7
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	3	5/6	10	11	10	10
(d_8,PC,Xn)	—	—	—	—	11	1_L + 10
(BR,Xn)	7	1_L + 8/1_L + 9	13	1_L + 13	13	1_L + 12
(bd,BR,Xn)	8	1_L + 9/1_L + 10	14	1_L + 14	14	1_L + 13
([bd,BR,Xn])	10	1_L + 11/1_L + 12	16	1_L + 16	16	1_L + 15
([bd,BR,Xn],od)	11	1_L + 12/1_L + 13	17	1_L + 17	17	1_L + 16
([bd,BR],Xn)	11	3_L + 10/3_L + 11	17	3_L + 15	17	3_L + 14
([bd,BR],Xn,od)	12	3_L + 11/3_L + 12	18	3_L + 16	18	3_L + 15

NOTES:
a. Bit instruction <ea> calculate and execute times T1/T2 apply to #<xxx>/Dn bit numbers.
b. This instruction interlocks the <ea> calculate and execute stages.
c. If the bit field spans a long-word boundary, add ten and nine clocks to the <ea> calculate and execute times, respectively. Two memory addresses are accessed in this case.
d. If the bit field spans a long-word boundary, add two clocks to the execute time. Two memory addresses are accessed in this case.
e. Immediate count specified for both width and offset and width and/or offset specified in register, respectively.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

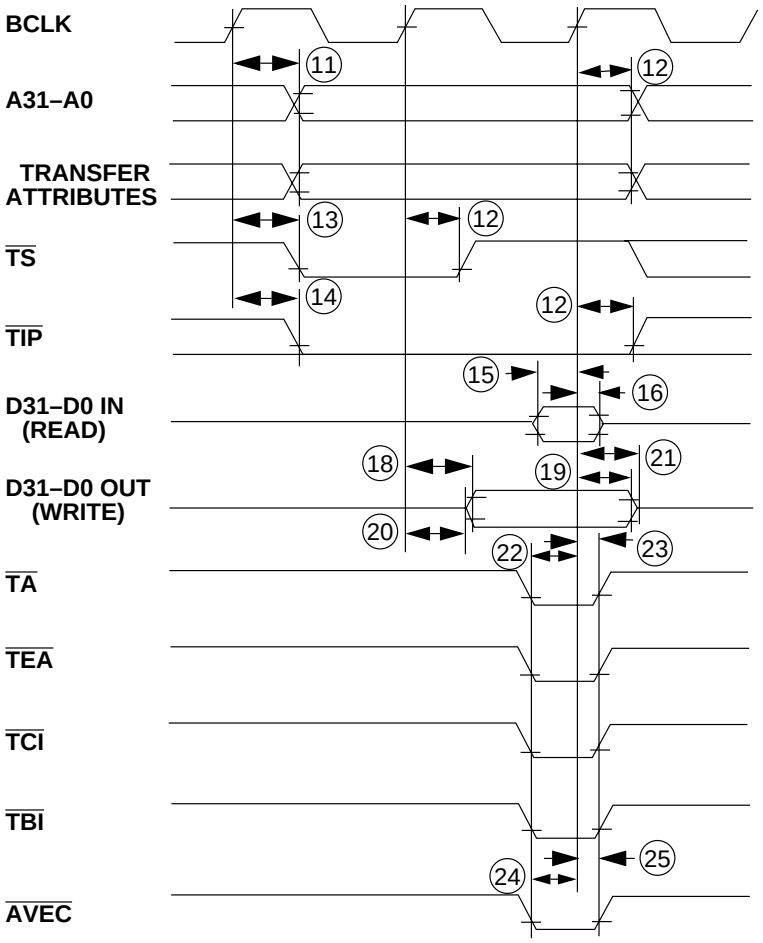
Addressing Mode	CHK2* (<ea>, Rn)		CLR		CMP	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	—	—	1	1	1	1
An	—	—	—	—	1	1
(An)	11	2_L + 9	1	1	1	1
(An)+	—	—	1	1	1	1
–(An)	—	—	1	1	1	1
(d_{16},An)	11	2_L + 9	1	1	1	1
(d_{16},PC)	12	3_L + 9	—	—	3	2_L + 1
(xxx).W, (xxx).L	11	2_L + 9	1	1	1	1
#<xxx>	—	—	—	—	1	1
(d_8,An,Xn)	13	1_L + 12	3	3	3	3
(d_8,PC,Xn)	14	2_L + 12	—	—	5	1_L + 4
(BR,Xn)	15	2_L + 13	6	1_L + 5	6	1_L + 5
(bd,BR,Xn)	16	2_L + 14	7	1_L + 6	7	1_L + 6
([bd,BR,Xn])	19	2_L + 17	9	1_L + 8	9	1_L + 8
([bd,BR,Xn],od)	20	2_L + 18	10	1_L + 9	10	1_L + 9
([bd,BR],Xn)	20	4_L + 16	10	3_L + 7	10	3_L + 7
([bd,BR],Xn,od)	21	4_L + 17	11	3_L + 8	11	3_L + 8

*This instruction interlocks the <ea> calculate and execute stages. Timing for Dn within bounds, UB > LB. For UB < LB, add three clocks to <ea> calculate and execute times. For Rn = An, add one clock to <ea> calculate and execute times.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	NEG, NEGX, NOT		PEA		ROL, ROR	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1	—	—	1	3/4*
An	—	—	—	—	—	—
(An)	1	1	2	1_L + 1	1	3
(An)+	1	1	—	—	1	3
–(An)	1	1	—	—	1	3
(d_{16},An)	1	1	2	1_L + 1	1	3
(d_{16},PC)	—	—	4	3_L + 1	—	—
(xxx).W, (xxx).L	1	1	2	1_L + 1	1	3
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	3	3	4	1_L + 3	3	5
(d_8,PC,Xn)	—	—	6	2_L + 4	—	—
(BR,Xn)	6	1_L + 5	7	2_L + 5	6	1_L + 7
(bd,BR,Xn)	7	1_L + 6	8	2_L + 6	7	1_L + 8
([bd,BR,Xn])	9	1_L + 8	10	2_L + 8	9	1_L + 10
([bd,BR,Xn],od)	10	1_L + 9	11	2_L + 9	10	1_L + 11
([bd,BR],Xn)	10	3_L + 7	11	4_L + 7	10	3_L + 9
([bd,BR],Xn,od)	11	3_L + 8	12	4_L + 8	11	3_L + 10

*Immediate count specified for shift count/shift count specified in register, respectively.



NOTE: Transfer Attribute Signals = UPAx, SIZx, TTx, TMx, TLNx, R/$\overline{W}$, $\overline{LOCK}$, $\overline{LOCKE}$, and $\overline{CIOUT}$

Figure A-5. Read/Write Timing

INDEX

–A–

–B–