Motorola - MC68040FE25A Datasheet

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	Obsolete
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=mc68040fe25a

TABLE OF CONTENTS (Continued)

the logical addresses of the currently executing process. Portions of translation tables can be dynamically allocated as the process requires additional memory.

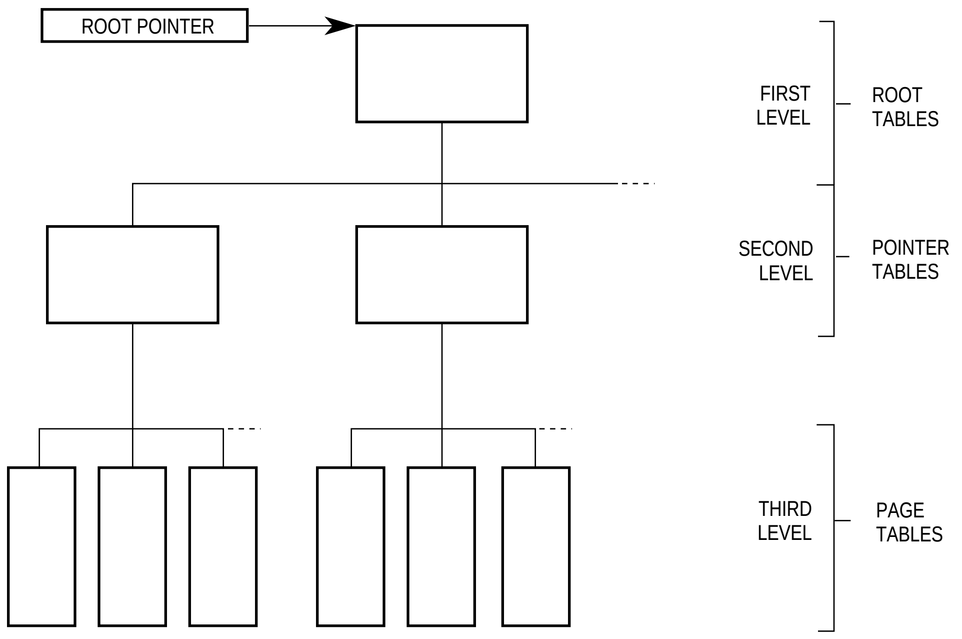


Figure 3-7. Translation Table Structure

The current privilege mode determines the use of the URP or SRP for translation of the access. The root pointer contains the base address of the translation table's root-level table. The translation table consists of tables of descriptors. The table descriptors of the root- and pointer-levels can be either resident or invalid. The page descriptors of the page-level table can be resident, indirect, or invalid. A page descriptor defines the physical address of a page frame in memory that corresponds to the logical address of a page. An indirect descriptor, which contains a pointer to the actual page descriptor, can be used when two or more logical addresses access a single page descriptor.

The table search uses logical addresses to access the translation tables. Figure 3-8 illustrates a logical address format, which is segmented into four fields: root index (RI), pointer index (PI), page index (PGI), and page offset. The first three fields extracted from the logical address index the base address for each table level. The seven bits of the logical address RI field are multiplied by 4 or shifted to the left by two bits. This sum is concatenated with the upper 23 bits of the appropriate root pointer (URP or SRP) to yield the physical address of a root-level table descriptor. Each of the 128 root-level table descriptors corresponds to a 32-Mbyte block of memory and points to the base of a pointer-level table.

Descriptor Address

This 30-bit field, which contains the physical address of a page descriptor, is only used in indirect descriptors.

G—Global

When this bit is set, it indicates the entry is global. PFLUSH instruction variants that specify nonglobal entries do not invalidate global entries, even when all other selection criteria are satisfied. If these PFLUSH variants are not used, then system software can use this bit.

M—Modified

This bit identifies a modified page. The M68040 sets the M-bit in the corresponding page descriptor before a write operation to a page for which the M-bit is clear, except for write-protect or supervisor violations. The read portion of a read-modify-write access is considered a write for updating purposes. The M68040 never clears this bit.

PDT—Page Descriptor Type

This field identifies the descriptor as an invalid descriptor, a page descriptor for a resident page, or an indirect pointer to another page descriptor.

00 =	Invalid This code indicates that the descriptor is invalid. An invalid descriptor can represent a nonresident page or a logical address range that is out of bounds. All other bits in the descriptor are ignored. When an invalid descriptor is encountered, an ATC entry is created for the logical address with the resident bit in the MMUSR clear.
01 or 11 =	Resident These codes indicate that the page is resident.
10 =	Indirect This code indicates that the descriptor is an indirect descriptor. Bits 31–2 contain the physical address of the page descriptor. This encoding is invalid for a page descriptor pointed to by an indirect descriptor.

Physical Address

This 20-bit field contains the physical base address of a page in memory. The logical address supplies the low-order bits of the address required to index into the page. When the page size is 8-Kbyte, the least significant bit of this field is not used.

S—Supervisor Protected

This bit identifies a page as supervisor only. Only programs operating in the supervisor mode are allowed to access the portion of the logical address space mapped by this descriptor when the S-bit is set. If the bit is clear, both supervisor and user accesses are allowed.

4.7.1 Instruction Cache

The IU uses the instruction cache to store instruction prefetches as it requests them. Instruction prefetches are normally requested from sequential memory locations except when a change of program flow occurs (e.g., a branch taken) or when an instruction that can modify the status register (SR) is executed, in which case the instruction pipe is automatically flushed and refilled. The instruction cache supports a line-based protocol that allows individual cache lines to be in either the invalid or valid states.

For instruction prefetch requests that hit in the cache, the half-line selected by physical address bit 3 is multiplexed onto the internal instruction data bus. When an access misses in the cache, the cache controller requests the line containing the required data from memory and places it in the cache. If available, an invalid line is selected and updated with the tag and data from memory. The line state then changes from invalid to valid by setting the V-bit. If all lines in the set are already valid, a pseudo-random replacement algorithm is used to select one of the four cache lines replacing the tag and data contents of the line with the new line information. Figure 4-5 illustrates the instruction-cache line state transitions resulting from processor and snoop controller accesses. Transitions are labeled with a capital letter, indicating the previous state, followed by a number indicating the specific case listed in Table 4-3.

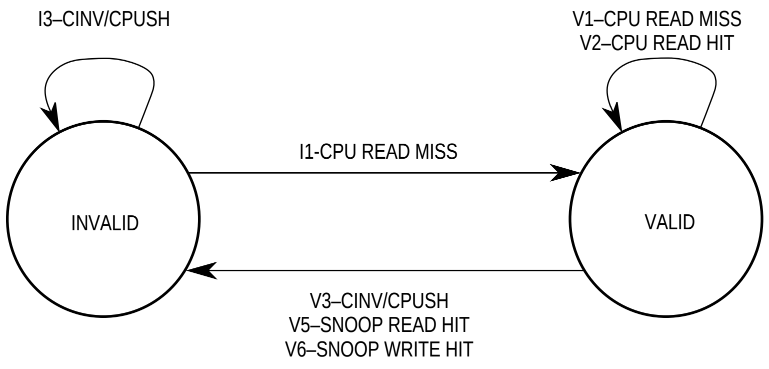
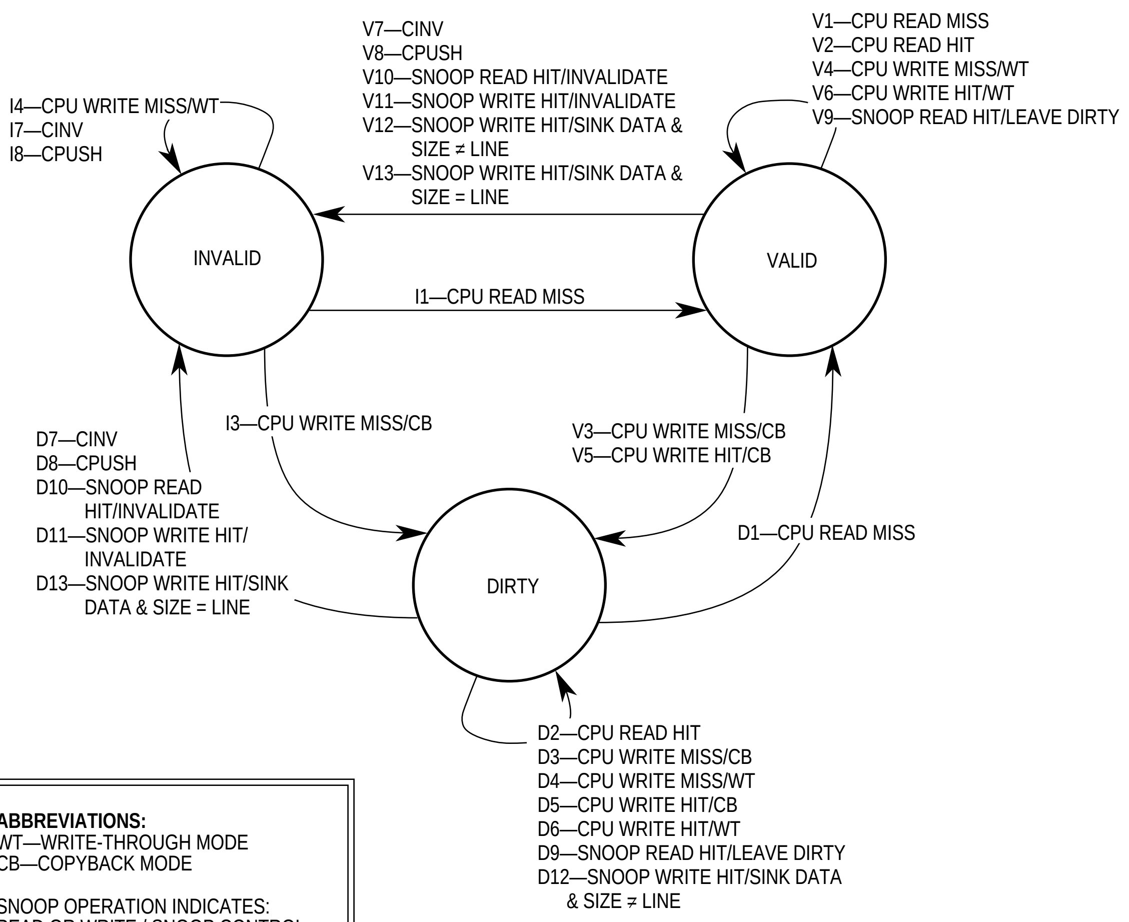


Figure 4-5. Instruction-Cache Line State Diagram


V7—CINV
V8—CPUSH
V10—SNOOP READ HIT/INVALIDATE
V11—SNOOP WRITE HIT/INVALIDATE
V12—SNOOP WRITE HIT/SINK DATA & SIZE ≠ LINE
V13—SNOOP WRITE HIT/SINK DATA & SIZE = LINE
V1—CPU READ MISS
V2—CPU READ HIT
V4—CPU WRITE MISS/WT
V6—CPU WRITE HIT/WT
V9—SNOOP READ HIT/LEAVE DIRTY
I4—CPU WRITE MISS/WT
I7—CINV
I8—CPUSH
INVALID
VALID
I1—CPU READ MISS
I3—CPU WRITE MISS/CB
V3—CPU WRITE MISS/CB
V5—CPU WRITE HIT/CB
D7—CINV
D8—CPUSH
D10—SNOOP READ HIT/INVALIDATE
D11—SNOOP WRITE HIT/ INVALIDATE
D13—SNOOP WRITE HIT/SINK DATA & SIZE = LINE
D1—CPU READ MISS
DIRTY
D2—CPU READ HIT
D3—CPU WRITE MISS/CB
D4—CPU WRITE MISS/WT
D5—CPU WRITE HIT/CB
D6—CPU WRITE HIT/WT
D9—SNOOP READ HIT/LEAVE DIRTY
D12—SNOOP WRITE HIT/SINK DATA & SIZE ≠ LINE

ABBREVIATIONS:
WT—WRITE-THROUGH MODE
CB—COPYBACK MODE

SNOOP OPERATION INDICATES:
READ OR WRITE / SNOOP CONTROL ENCODING

Figure 4-6. Data-Cache Line State Diagram

```
            DLE:       in          bit;
            PCLK:      in          bit;
            BCLK:      in          bit;
            IPL:       in          bit_vector(0 to 2);
            RSTI:      in          bit;
            CDIS:      in          bit;
            MDIS:      in          bit;
            EGND:      linkage bit_vector(1 to 23);
            EVDD:      linkage     bit_vector(1 to 12);
            IGND:      linkage     bit_vector(1 to 12);
            IVDD:      linkage     bit_vector(1 to 7);
            CGND:      linkage     bit_vector(1 to 2);
            CVDD:      linkage     bit_vector(1 to 6);
            PGND:      linkage     bit_vector(1 to 3);
            PVDD:      linkage     bit_vector(1 to 2)
            );


      use STD_1149_1_1990.all;
      attribute PIN_MAP of MC68040 : entity is PHYSICAL_PIN_MAP;



   —18x18 PGA Pin Map

   constant PGA_18x18 : PIN_MAP_STRING :=
            "TDI:       S3,                                                        "  &
            "TDO:       T2,                                                        "  &
            "TMS:       S5,                                                        "  &
            "TCK:       S4,                                                        "  &
            "TRST:      T3,                                                        "  &
            "RSTO:      R3,                                                        "  &
            "IPEND:     S1,                                                        "  &
            "CIOUT:     R1,                                                        "  &
            "UPA:       (Q3, Q1),                                                  "  &
            "TT:        (P3, P2),                                                  "  &
            "A:         (L18, K18, J17, J18, H18, G18, G16, F18, E18, F16, P1,  N3,   "  &
            "            N1,  M1,  L1,  K1,  K2,  J1,  H1,  J2,  G1,  F1,  E1,  G3,    "  &
            "            D1,  F3,  E2,  C1,  E3,  B1,  D3,  A1),                  "  &
            "D:         (C3,  B3,  C4,  A2,  A3,  A4,  A5,  A6,  B7,  A7,  A8,  A9,   "  &
            "            A10, A11, A12, A13, B11, A14, B12, A15, A16, A17, B16, C15,  "  &
            "            A18, C16, B18, D16, C18, E16, E17, D18),                 "  &
            "LOCKE:     R18,                                                       "  &
            "LOCK:      S18,                                                       "  &
            "R_W:       N16,                                                       "  &
            "TLN:       (Q18, P18),                                                "  &
            "TM:        (N18, M18, K17),                                           "  &
            "SIZ:       (P17, P16),                                                "  &
            "MI:        Q16,                                                       "  &
            "BR:        T18,                                                       "  &
            "TS:        R16,                                                       "  &
            "BB:        T17,                                                       "  &
            "TIP:       R15,                                                       "  &
            "PST:       (T15, S14, R14, T16),                                      "  &
            "TA:        T14,                                                       "  &
            "TEA:       S13,                                                       "  &
            "BG:        T13,                                                       "  &
            "SC:        (T12, S12),                                                "  &
            "TBI:       S11,                                                       "  &
            "AVEC:      T11,                                                       "  &
```

```
num    cell      port       function    safe   ccell   dsval   rslt

"0     (BC_2,    RSTO,      output2,    X),                                 "  &
"1     (BC_2,    IPEND,     output2,    X),                                 "  &
"2     (BC_2,    CIOUT,     output3,    X,     156,    0,      Z),          "  &     —156 = io.0
"3     (BC_2,    UPA(0),    output3,    X,     156,    0,      Z),          "  &
"4     (BC_2,    UPA(1),    output3,    X,     156,    0,      Z),          "  &
"5     (BC_2,    TT(0),     output3,    X,     156,    0,      Z),          "  &
"6     (BC_4,    TT(0),     input,      X),                                 "  &
"7     (BC_2,    TT(1),     output3,    X,     156,    0,      Z),          "  &
"8     (BC_4,    TT(1),     input,      X),                                 "  &
"9     (BC_2,    A(10),     output3,    X,     150,    0,      Z),          "  &     —150 = io.ab
"10    (BC_4,    A(10),     input,      X),                                 "  &
"11    (BC_2,    A(11),     output3,    X,     150,    0,      Z),          "  &
"12    (BC_4,    A(11),     input,      X),                                 "  &
"13    (BC_2,    A(12),     output3,    X,     150,    0,      Z),          "  &
"14    (BC_4,    A(12),     input,      X),                                 "  &
"15    (BC_2,    A(13),     output3,    X,     150,    0,      Z),          "  &
"16    (BC_4,    A(13),     input,      X),                                 "  &
"17    (BC_2,    A(14),     output3,    X,     150,    0,      Z),          "  &
"18    (BC_4,    A(14),     input,      X),                                 "  &
"19    (BC_2,    A(15),     output3,    X,     150,    0,      Z),          "  &
"20    (BC_4,    A(15),     input,      X),                                 "  &
"21    (BC_2,    A(16),     output3,    X,     150,    0,      Z),          "  &
"22    (BC_4,    A(16),     input,      X),                                 "  &
"23    (BC_2,    A(17),     output3,    X,     150,    0,      Z),          "  &
"24    (BC_4,    A(17),     input,      X),                                 "  &
"25    (BC_2,    A(18),     output3,    X,     150,    0,      Z),          "  &
"26    (BC_4,    A(18),     input,      X),                                 "  &
"27    (BC_2,    A(19),     output3,    X,     150,    0,      Z),          "  &
"28    (BC_4,    A(19),     input,      X),                                 "  &
"29    (BC_2,    A(20),     output3,    X,     150,    0,      Z),          "  &
"30    (BC_4,    A(20),     input,      X),                                 "  &
"31    (BC_2,    A(21),     output3,    X,     150,    0,      Z),          "  &
"32    (BC_4,    A(21),     input,      X),                                 "  &
"33    (BC_2,    A(22),     output3,    X,     150,    0,      Z),          "  &
"34    (BC_4,    A(22),     input,      X),                                 "  &
"35    (BC_2,    A(23),     output3,    X,     150,    0,      Z),          "  &
"36    (BC_4,    A(23),     input,      X),                                 "  &
"37    (BC_2,    A(24),     output3,    X,     150,    0,      Z),          "  &
"38    (BC_4,    A(24),     input,      X),                                 "  &
"39    (BC_2,    A(25),     output3,    X,     150,    0,      Z),          "  &
"40    (BC_4,    A(25),     input,      X),                                 "  &
"41    (BC_2,    A(26),     output3,    X,     150,    0,      Z),          "  &
"42    (BC_4,    A(26),     input,      X),                                 "  &
"43    (BC_2,    A(27),     output3,    X,     150,    0,      Z),          "  &
"44    (BC_4,    A(27),     input,      X),                                 "  &
"45    (BC_2,    A(28),     output3,    X,     150,    0,      Z),          "  &
"46    (BC_4,    A(28),     input,      X),                                 "  &
"47    (BC_2,    A(29),     output3,    X,     150,    0,      Z),          "  &
"48    (BC_4,    A(29),     input,      X),                                 "  &
"49    (BC_2,    A(30),     output3,    X,     150,    0,      Z),          "  &
"50    (BC_4,    A(30),     input,      X),                                 "  &
"51    (BC_2,    A(31),     output3,    X,     150,    0,      Z),          "  &
"52    (BC_4,    A(31),     input,      X),                                 "  &
"53    (BC_2,    D(0),      output3,    X,     151,    0,      Z),          "  &     — 151 = io.db
"54    (BC_2,    D(1),      output3,    X,     151,    0,      Z),          "  &
"55    (BC_2,    D(2),      output3,    X,     151,    0,      Z),          "  &
"56    (BC_2,    D(3),      output3,    X,     151,    0,      Z),          "  &
```

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.

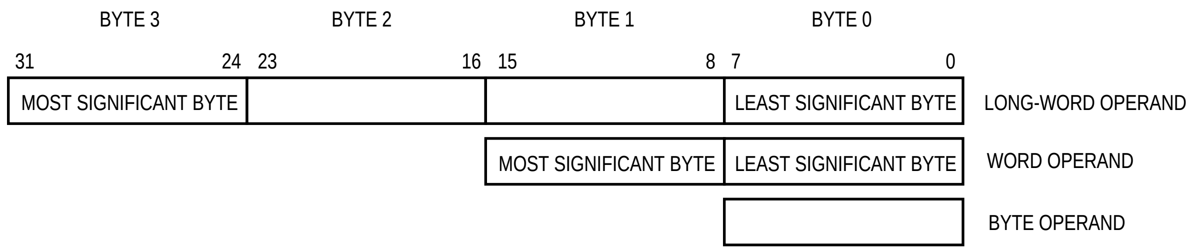


Figure 7-2. Internal Operand Representation

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.

The interrupt acknowledge bus cycle is a read transfer. It differs from a normal read cycle in the following respects:

1. TT1 and TT0 = $3 to indicate an acknowledged bus cycle.
2. Address signals A31–A0 are set to all ones ($FFFFFFFF).
3. TM2–TM0 are set to the interrupt request level (the inverted values of $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$).

The responding device places the vector number on the data bus during the interrupt acknowledge bus cycle, and the cycle is terminated normally with $\overline{\text{TA}}$. Figures 7-21 and 7-22 illustrate a flowchart and functional timing diagram for an interrupt acknowledge cycle terminated with $\overline{\text{TA}}$.

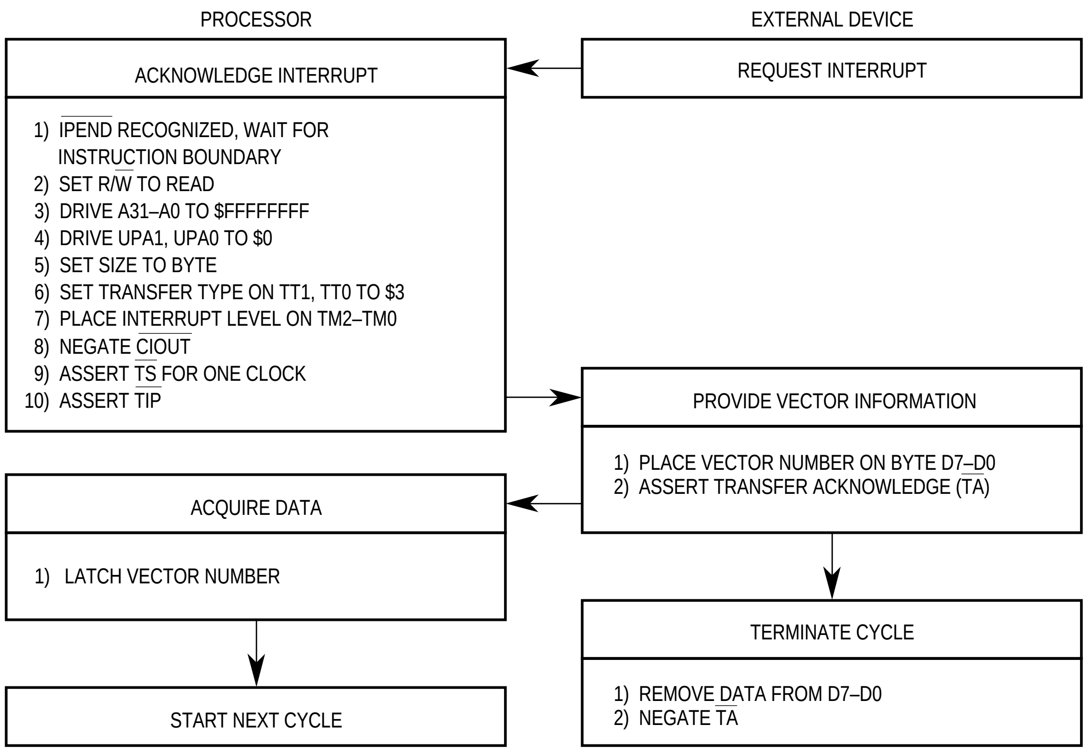

PROCESSOR
EXTERNAL DEVICE
ACKNOWLEDGE INTERRUPT
1) IPEND RECOGNIZED, WAIT FOR INSTRUCTION BOUNDARY
2) SET R/W TO READ
3) DRIVE A31–A0 TO $FFFFFFFF
4) DRIVE UPA1, UPA0 TO $0
5) SET SIZE TO BYTE
6) SET TRANSFER TYPE ON TT1, TT0 TO $3
7) PLACE INTERRUPT LEVEL ON TM2–TM0
8) NEGATE CIOUT
9) ASSERT TS FOR ONE CLOCK
10) ASSERT TIP
REQUEST INTERRUPT
PROVIDE VECTOR INFORMATION
1) PLACE VECTOR NUMBER ON BYTE D7–D0
2) ASSERT TRANSFER ACKNOWLEDGE (TA)
ACQUIRE DATA
1) LATCH VECTOR NUMBER
TERMINATE CYCLE
1) REMOVE DATA FROM D7–D0
2) NEGATE TA
START NEXT CYCLE

Figure 7-21. Interrupt Acknowledge Bus Cycle Flowchart

generates the vector number, which is the sum of the interrupt priority level plus 24 ($18). There are seven distinct autovectors that can be used, corresponding to the seven levels of interrupts available with $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$ signals. Figure 7-23 illustrates a functional timing diagram for an autovector operation.

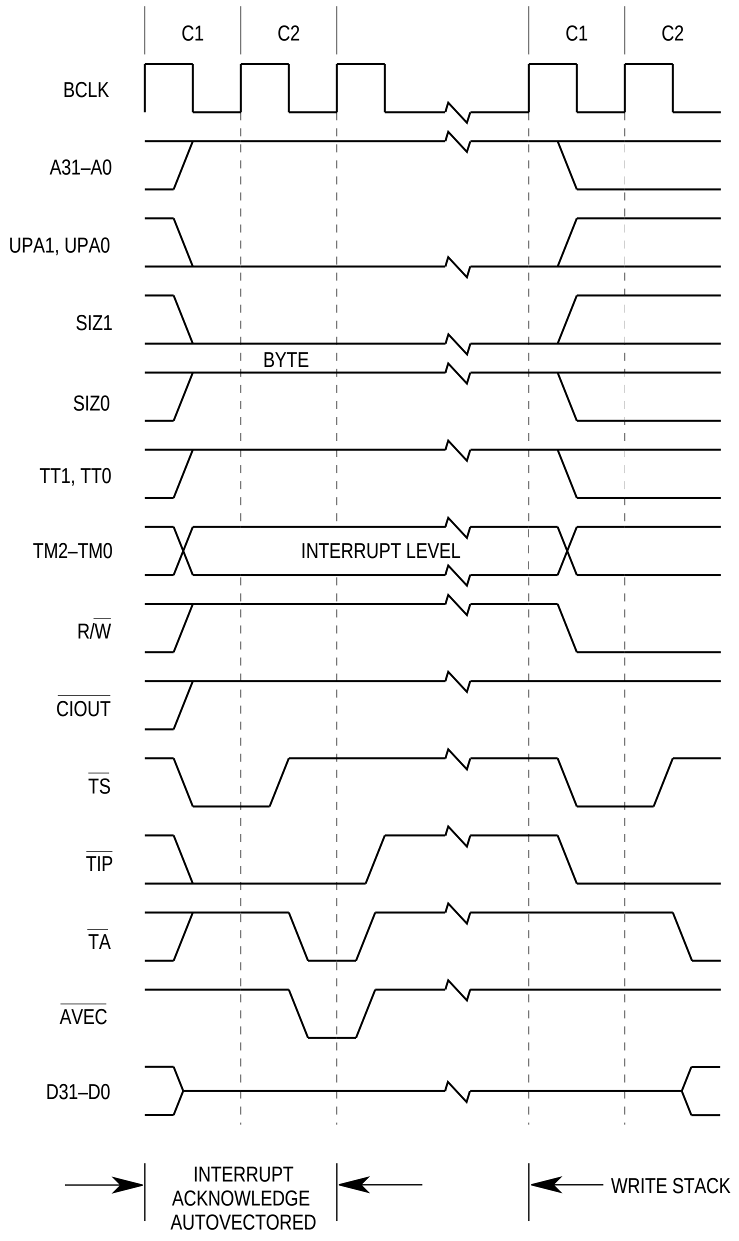


Figure 7-23. Autovector Interrupt Acknowledge Bus Cycle Timing

7.5.1.3 SPURIOUS INTERRUPT ACKNOWLEDGE BUS CYCLE. When a device does not respond to an interrupt acknowledge bus cycle with $\overline{\text{TA}}$, or $\overline{\text{AVEC}}$ and $\overline{\text{TA}}$, the external logic typically returns the transfer error acknowledge signal ($\overline{\text{TEA}}$). In this case, the M68040 automatically generates the spurious interrupt vector number 24 ($18) instead of the interrupt vector number. If $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ are both asserted, the processor retries the cycle.

To properly control termination of a bus cycle for a bus error or retry condition, $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ must be asserted and negated for the same rising edge of BCLK. Table 7-5 lists the control signal combinations and the resulting bus cycle terminations. Bus error and retry terminations during burst cycles operate as described in **7.4.2 Line Read Transfers** and **7.4.4 Line Write Transfers**.

Table 7-5. $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ Assertion Results

Case No.	$\overline{\text{TA}}$	$\overline{\text{TEA}}$	Result
1	High	Low	Bus Error—Terminate and Take Bus Error Exception, Possibly Deferred
2	Low	Low	Retry Operation—Terminate and Retry
3	Low	High	Normal Cycle Terminate and Continue
4	High	High	Insert Wait States

7.6.1 Bus Errors

The system hardware can use the $\overline{\text{TEA}}$ signal to abort the current bus cycle when a fault is detected. A bus error is recognized during a bus cycle when $\overline{\text{TA}}$ is negated and $\overline{\text{TEA}}$ is asserted. When the processor recognizes a bus error condition for an access, the access is terminated immediately. A line access that has $\overline{\text{TEA}}$ asserted for one of the four long-word transfers aborts without completing the remaining transfers, regardless of whether the line transfer uses a burst or burst-inhibited access.

When $\overline{\text{TEA}}$ is asserted to terminate a bus cycle, the M68040 can enter access error exception processing immediately following the bus cycle, or it can defer processing the exception. The instruction prefetch mechanism requests instruction words from the instruction memory unit before it is ready to execute them. If a bus error occurs on an instruction fetch, the processor does not take the exception until it attempts to use the instruction. Should an intervening instruction cause a branch or should a task switch occur, the access error exception for the unused access does not occur. Similarly, if a bus error is detected on the second, third, or fourth long-word transfer for a line read access, an access error exception is taken only if the execution unit is specifically requesting that long word. Otherwise, the line is not placed in the cache, and the processor repeats the line access when another access references the line. If a misaligned operand spans two long words in a line, a bus error on either the first or second transfer for the line causes exception processing to begin immediately. A bus error termination for any write accesses or for read accesses that reference data specifically requested by the execution unit causes the processor to begin exception processing immediately. Refer to **Section 8 Exception Processing** for details of access error exception processing.

When a bus error terminates an access, the contents of the corresponding cache can be affected in different ways, depending on the type of access. For a cache line read to replace a valid instruction or data cache line, the cache line being filled is invalidated before the bus cycle begins and remains invalid if the replacement line access is terminated with a bus error. If a dirty data cache line is being replaced and a bus error occurs during the replacement line read, the dirty line is restored from an internal push

7.6.3 Double Bus Fault

A double bus fault occurs when an access or address error occurs during the exception processing sequence—e.g., the processor attempts to stack several words containing information about the state of the machine while processing an access error exception. If a bus error occurs during the stacking operation, the second error is considered a double bus fault.

The M68040 indicates a double bus fault condition by continuously driving PST3–PST0 with an encoded value of $5 until the processor is reset. Only an external reset operation can restart a halted processor. While the processor is halted, negating $\overline{\text{BR}}$ and forcing all outputs to a high-impedance state releases the external bus.

A second access or address error that occurs during execution of an exception handler or later, does not cause a double bus fault. A bus cycle that is retried does not constitute a bus error or contribute to a double bus fault. The processor continues to retry the same bus cycle as long as external hardware requests it.

7.7 BUS SYNCHRONIZATION

The M68040 integer unit generates access requests to the instruction and data memory units to support integer and floating-point operations. Both the <ea> fetch and write-back stages of the integer unit pipeline perform accesses to the data memory unit, with effective address fetches assigned a higher priority. This priority allows data read and write accesses to occur out of order, with a memory write access potentially delayed for many clocks while allowing read accesses generated by later instructions to complete. The processor detects a read access that references earlier data waiting to be written (address collisions) and allows the corresponding write access to complete. A given sequence of read accesses or write accesses is completed in order, and reordering only occurs with writes relative to reads. Figure 2-1 in **Section 2 Integer Unit** illustrates the integer pipeline stages.

Besides address collisions, the instruction restart model used for exception processing in the M68040 causes another potential problem. After the operand fetch for an instruction, an exception that causes the instruction to be aborted can occur, resulting in another access for the operand after the instruction restarts. For example, an exception could occur after a read access of an I/O device's status register. The exception causes the instruction to be aborted and the register to be read again. If the first read accesses clears the status bits, the status information is lost, and the instruction obtains incorrect data.

Designating the memory page containing the address of the device as serialized noncachable prevents multiple out-of-order accesses to devices sensitive to such accesses. When the data memory unit detects an attempt to read an operand from a page designated as serialized noncachable, it allows all pending write accesses to complete before beginning the external read access. The definition of a page as noncachable versus serialized noncachable only affects read accesses. When a write operation reaches the integer unit's write-back stage, all previous instructions have completed. When a read access to a serialized noncachable page begins, only a bus error exception

If a floating-point exception is pending from a previous floating-point instruction, a pre-instruction exception is taken. After the appropriate exception handler is executed, the conditional instruction is restarted. When the FPU pipeline is idle (all previous floating-point instructions have completed) and no exceptions are pending, the processor evaluates the conditional predicate and checks for a BSUN exception before executing the conditional instruction.

9.7.1.1 MASKABLE EXCEPTION CONDITIONS. A BSUN exception occurs if the conditional predicate is one of the IEEE nonaware branches and the FPCC NAN bit is set. When the processor detects this condition, it sets the BSUN bit in the FPSR EXC byte.

a. If the user BSUN exception handler is disabled, the floating-point condition is evaluated as if it were the equivalent IEEE aware conditional predicate. No exceptions are taken.

b. If the user BSUN exception handler is enabled, the processor takes a floating-point pre-instruction exception. A $0 stack frame is saved, and vector number 48 is generated to access the BSUN exception vector. The BSUN entry in the processor's vector table points to the M68040FPSP BSUN exception handler.

For MC68881/MC68882 compatibility, the M68040FPSP updates the FPIAR by copying the PC value in the pre-instruction stack frame to the FPIAR. The M68040FPSP BSUN exception handler restores the FPU to its exceptional state, cleans up the stack to the state prior to the M68040FPSP BSUN exception handler's execution, and continues instruction execution at the user BSUN exception handler. No parameters are passed to the user BSUN exception handler since the M68040FPSP BSUN exception handler provides the illusion that it never existed.

The user BSUN exception handler must execute an FSAVE as its first floating-point instruction. FSAVE allows other floating-point instructions to execute without reporting the BSUN exception again, although none of the state frame values are useful in the execution of the user BSUN exception handler. The BSUN exception is unique in that the exception is taken before the conditional predicate is evaluated. If the user BSUN exception handler does not set the PC to the instruction following the one that caused BSUN exception when returning, the exception is executed again. Therefore, it is the responsibility of the user BSUN exception handler to prevent the conditional instruction from taking the BSUN exception again. There are four ways to prevent taking the exception again:

1. Incrementing the stored PC in the stack bypasses the conditional instruction. This technique applies to situations where a fall-through is desired. Note that accurate calculation of the PC increment requires detailed knowledge of the size of the conditional instruction being bypassed.
2. Clearing the NAN bit prevents the exception from being taken again. However, this alone cannot deterministically control the result's indication (true or false) that would be returned when the conditional instruction reexecutes.
3. Disabling the BSUN bit also prevents the exception from being taken again. Like the second method, this method cannot control the result indication (true or false) that would be returned when the conditional instruction reexecutes.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

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

NOTES:

a. This instruction interlocks the <ea> calculate and execute stages.

b. If the bit field spans a long-word boundary, add two clocks to the execute time. Two memory addresses are accessed in this case.

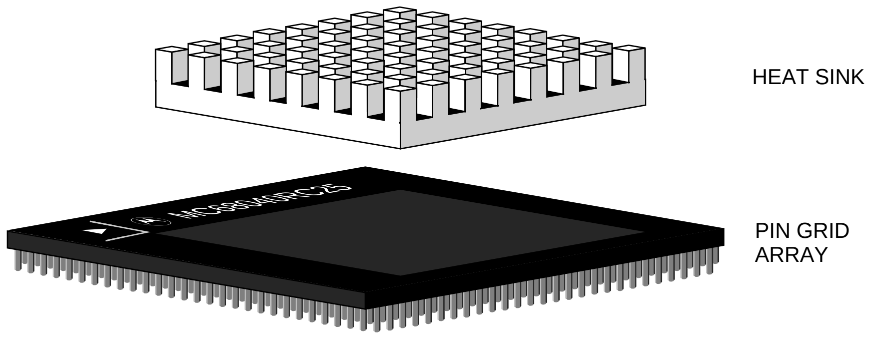
c. If the bit field spans a long-word boundary, add seven clocks to both the <ea> calculate and execute times. Two memory addresses are accessed in this case.

d. If the bit field spans a long-word boundary, add ten and nine clocks to both the <ea> calculate and execute times, respectively. Two memory addresses are accessed in this case.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	MOVES <ea>,An*		MOVES <ea>,Dn*		MOVES Rn,<ea>*	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	—	—	—	—	—	—
An	—	—	—	—	—	—
(An)	28	4_L + 24	20	4_L + 19	13	4_L + 9
(An)+	28	4_L + 24	20	4_L + 19	13	4_L + 9
–(An)	17	2_L + 15	11	12	11	2_L + 9
$(d_{16}$,An)	29	4_L + 24	21	4_L + 19	14	4_L + 9
$(d_{16}$,PC)	—	—	—	—	—	—
(xxx).W, (xxx).L	17	2_L + 15	11	4_L + 10	11	2_L + 9
#<xxx>	—	—	—	—	—	—
$(d_8$,An,Xn)	29	1_L + 27	21	1_L + 22	14	1_L + 12
$(d_8$,PC,Xn)	—	—	—	—	—	—
(BR,Xn)	21	2_L + 19	15	2_L + 14	15	2_L + 13
(bd,BR,Xn)	22	2_L + 20	16	2_L + 15	16	2_L + 14
([bd,BR,Xn])	35	2_L + 32	26	2_L + 27	21	2_L + 17
([bd,BR,Xn],od)	31	2_L + 29	23	2_L + 24	20	2_L + 18
([bd,BR],Xn)	36	4_L + 31	27	4_L + 26	21	4_L + 16
([bd,BR],Xn,od)	32	4_L + 28	24	4_L + 23	21	4_L + 17

*Times listed are typical. This instruction interlocks the <ea> calculate and execute stages and synchronizes some portions of the processor before execution.



NOTE: Do not cover up microprocessor markings with an adhesive mounted heat sink.

Figure 11-10. Heat Sink with Adhesive

All pin-fin heat sinks tested were made from extrusion aluminum products. The planar face of the heat-sink matting to the package should have a good degree of planarity; if it has any curvature, the curvature should be convex at the central region of the heat-sink surface to provide intimate physical contact to the PGA surface. This heat sinks meet this criteria. Nonplanar, concave curvature in the central regions of the heat sink results in poor thermal contact to the package.

Although there are several ways to attach a heat sink to the package, it is easiest to use a demountable heat-sink attachment called “E-Z attach for PGA packages” (see Figure 11-33). A steel spring clamps the heat sink and the package to a plastic frame. Besides the height of the heat sink and plastic frame, no additional height is added to the package. The interface between the ceramic package and the aluminum heat sink was evaluated for both dry and wet interfaces in still air. The thermal grease reduced the θ_{CA} quite significantly (about 2.5 °C/W) in still air. An attachment with thermal grease provided about the same thermal performance as if a thermal epoxy had been used.

a pending unimplemented floating-point instruction exception. The access error exception handler then completes the write-backs in software, and exception processing for the unimplemented floating-point instruction exception begins immediately after return from the access error handler.

3. The processor begins exception processing for the unimplemented floating-point instruction by making an internal copy of the current SR. The processor then enters the supervisor mode and clears the trace bits (T1 and T0). It creates a format $4 stack frame and saves the internal copy of the SR, PC, vector offset, calculated effective address, and PC value of the faulted instruction in the stack frame.

   The effective address field of the format $4 stack frame contains the calculated effective address of the operand for the faulted floating-point instruction using the addressing mode in which the effective address is calculated. For immediate and register direct addressing modes, this field is $0. The saved PC value is the logical address of the instruction that follows the unimplemented floating-point instruction. This value will be restored during RTE execution. The vector offset format number ($4) is used for this eight-word stack frame. Note that an MC68040 cannot correctly handle a stack format $4. The PC of the faulted instruction contains a long-word PC of the floating-point instruction that caused the trap to occur. The information is provided so that the instruction is available for software emulation of floating-point instructions. The processor generates exception vector number 11 for the unimplemented F-line instruction exception vector, fetches the address of the F-line exception handler from the exception vector table, and begins execution of the handler after prefetching instructions to fill the pipeline. Refer to **Section 8 Exception Processing** for details about exception processing.

A.5.2 MC68LC040 Stack Frames

When the processor executes an RTE instruction, it examines the stack frame on top of the active supervisor stack to determine if it is a valid frame and what type of context restoration it requires. The MC68LC040 provides five different stack frames for exception processing and allows for an MC68040-specific stack frame. The set of frames includes four- and six-word stack frames, a four-word throwaway stack frame, an access error stack frame, and a new eight-word unimplemented floating-point stack frame. The stack frame that the MC68040 can generate and the MC68LC040 can process is the floating-point post-instruction stack frame. Refer to **Section 8 Exception Processing** for details about exception stack frames.

Table 12-1. Eight-Word Stack Frame (Format $4)

Stack Frames	Exception Types	Stacked PC Points To

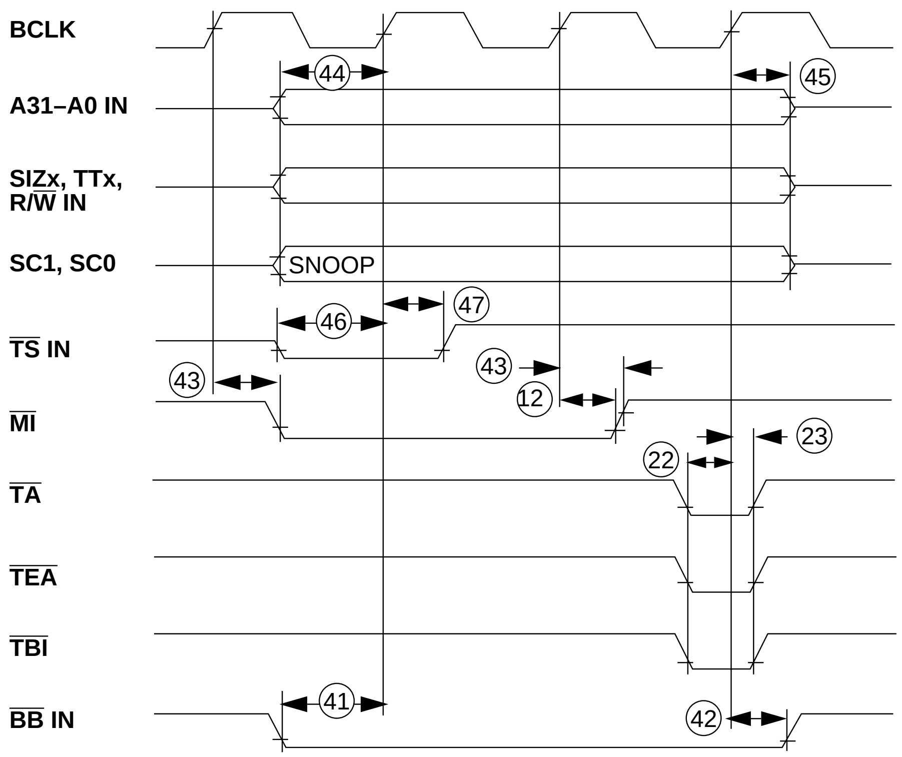


Figure B-11. Snoop Miss Timing

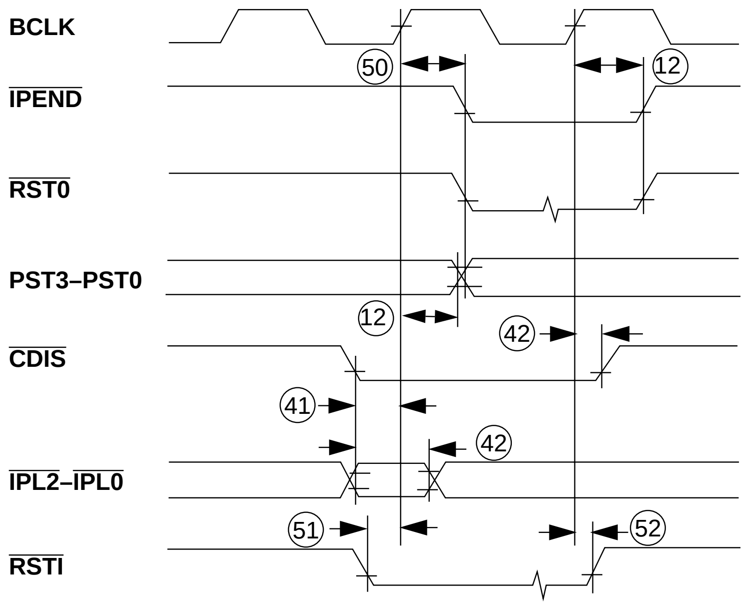


Figure B-12. Other Signal Timing

C.7.3 DC Electrical Specifications (V_{CC} = 3.3 Vdc ±10 %)

Characteristic	Symbol	Min	Max	Unit
Input High Voltage	V_{IH}	2	5.5	V
Input Low Voltage	V_{IL}	GND	0.8	V
Overshoot	—	—	TBD	V
Input Leakage Current @ 0.5/2.4 V During Normal Operation Only $\overline{AVEC}$, BCLK, $\overline{BG}$, $\overline{CDIS}$, $\overline{MDIS}$[1], $\overline{IPLx}$, $\overline{RSTI}$, SCx, $\overline{TBI}$, TLNx, $\overline{TCI}$, TCK, $\overline{TEA}$	I_{in}	TBD	TBD	µA
Hi-Z (Off-State) Leakage Current @ 0.5/2.4 V During Normal Operation An, $\overline{BB}$, $\overline{CIOUT}$, Dn, $\overline{LOCK}$, $\overline{LOCKE}$, R/$\overline{W}$, SIZx, $\overline{TA}$, TDO, $\overline{TIP}$, TMx, TLNx, $\overline{TS}$, TTx, UPAx	I_{TSI}	TBD	TBD	µA
Signal Low Input Current, V_{IL} = 0.8 V TMS, TDI	I_{IL}	TBD	TBD	mA
Signal High Input Current, V_{IH} = 2.0 V TMS, TDI	I_{IH}	TBD	TBD	mA
Output High Voltage I_{OH} = 5ma	V_{OH}	2.4	—	V
Output Low Voltage I_{OL} = 5ma	V_{OL}	—	0.5	V
Capacitance2, V_{in} = 0 V, f = 1 MHz	C_{in}	—	TBD	pF

NOTES:
1. There is no $\overline{MDIS}$ on the MC68EC040V.
2. Capacitance is periodically sampled rather than 100% tested.

C.7.4 Power Dissipation

	25 MHz	33 MHz
Worst Case (V_{CC} = 3.6 V , T_A = 0°C)		
MC68040V	TBD	2 W
MC68EC040V	TBD	2 W
LPSTOP Mode - No output loads, not driving bus		
MC68040V	TBD	TBD
MC68EC040V	TBD	TBD
Typical Values (V_{CC} = 3.3 V, T_J = TBD°C)* - Normal Operation		
MC68040V	TBD	1.5 W
MC68EC040V	TBD	1.5 W

*This information is for system reliability purposes.

C.7.7 Input AC Timing Specifications (See Figures C-13 to C-21)

PRELIMINARY

Num	Characteristic	0–16.67 MHz		25 MHz		33 MHz		Unit
		Min	Max	Min	Max	Min	Max	
15	Data-In Valid to BCLK (Setup)	5	—	5	—	4	—	ns
16	BCLK to Data-In Invalid (Hold)	4	—	4	—	4	—	ns
17	BCLK to Data-In High Impedance (Read Followed by Write)	—	49	—	49	—	36.5	ns
22a	$\overline{\text{TA}}$ Valid to BCLK (Setup)	10	—	10	—	10	—	ns
22b	$\overline{\text{TEA}}$ Valid to BCLK (Setup)	10	—	10	—	10	—	ns
22c	$\overline{\text{TCI}}$ Valid to BCLK (Setup)	10	—	10	—	10	—	ns
22d	$\overline{\text{TBI}}$ Valid to BCLK (Setup)	11	—	11	—	10	—	ns
23	BCLK to $\overline{\text{TA}}$, $\overline{\text{TEA}}$, $\overline{\text{TCI}}$, $\overline{\text{TBI}}$ Invalid (Hold)	2	—	2	—	2	—	ns
24	$\overline{\text{AVEC}}$ Valid to BCLK (Setup)	5	—	5	—	5	—	ns
25	BCLK to $\overline{\text{AVEC}}$ Invalid (Hold)	2	—	2	—	2	—	ns
41a	$\overline{\text{BB}}$ Valid to BCLK (Setup)	7	—	7	—	7	—	ns
41b	$\overline{\text{BG}}$ Valid to BCLK (Setup)	8	—	8	—	7	—	ns
41c	$\overline{\text{CDIS}}$, $\overline{\text{MDIS}}$* Valid to BCLK (Setup)	10	—	10	—	8	—	ns
41d	$\overline{\text{IPLx}}$ Valid to BCLK (Setup)	4	—	4	—	3	—	ns
42	BCLK to $\overline{\text{BB}}$, $\overline{\text{BG}}$, $\overline{\text{CDIS}}$, $\overline{\text{MDIS}}$*, $\overline{\text{IPLx}}$ Invalid (Hold)	2	—	2	—	2	—	ns
44a	Address Valid to BCLK (Setup)	8	—	8	—	7	—	ns
44b	SIZx Valid to BCLK (Setup)	12	—	12	—	8	—	ns
44c	TTx Valid to BCLK (Setup)	6	—	6	—	8.5	—	ns
44d	R/$\overline{\text{W}}$ Valid to BCLK (Setup)	6	—	6	—	5	—	ns
44e	SCx Valid to BCLK (Setup)	10	—	10	—	11	—	ns
45	BCLK to Address SIZx, TTx, R/$\overline{\text{W}}$, SCx Invalid (Hold)	2	—	2	—	2	—	ns
46	$\overline{\text{TS}}$ Valid to BCLK (Setup)	5	—	5	—	9	—	ns
47	BCLK to $\overline{\text{TS}}$ Invalid (Hold)	2	—	2	—	2	—	ns
49	BCLK to $\overline{\text{BB}}$ High Impedance (Processor Assumes Bus Mastership)	—	9	—	9	—	9	ns
51	$\overline{\text{RSTI}}$ Valid to BCLK	5	—	5	—	4	—	ns
52	BCLK to $\overline{\text{RSTI}}$ Invalid	2	—	2	—	2	—	ns
B	$\overline{\text{LFO}}$ change to valid $\overline{\text{IPLx}}$, $\overline{\text{RSTI}}$ (setup)	5	—	5	—	5		ns
D	$\overline{\text{IPEND}}$ valid to $\overline{\text{IPLx}}$ invalid (Hold)	0	—	0	—	0	—	ns
V	$\overline{\text{RSTI}}$ pulse width, leaving LPSTOP mode	10	—	10	—	10	—	ns
Z	$\overline{\text{IPLx}}$, $\overline{\text{RSTI}}$ valid to $\overline{\text{LFO}}$ change (Hold)	500	—	500	—	500	—	ns

NOTE: *Not on the MC68EC040V.