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	EC000
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	12MHz
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	68-LCC (J-Lead)
Supplier Device Package	68-PLCC (24.21x24.21)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68hc001ei12

LIST OF ILLUSTRATIONS (Concluded)

SECTION 2
INTRODUCTION

The section provide a brief introduction to the M68000 microprocessors (MPUs). Detailed information on the programming model, data types, addressing modes, data organization and instruction set can be found in M68000PM/AD, *M68000 Programmer's Reference Manual*. All the processors are identical from the programmer's viewpoint, except that the MC68000 can directly access 16 Mbytes (24-bit address) and the MC68008 can directly access 1 Mbyte (20-bit address on 48-pin version or 22-bit address on 52-pin version). The MC68010, which also uses a 24-bit address, has much in common with the other devices; however, it supports additional instructions and registers and provides full virtual machine/memory capability. Unless noted, all information pertains to all the M68000 MPUs.

2.1 PROGRAMMER'S MODEL

All the microprocessors executes instructions in one of two modes—user mode or supervisor mode. The user mode provides the execution environment for the majority of application programs. The supervisor mode, which allows some additional instructions and privileges, is used by the operating system and other system software.

2.1.1 User' Programmer's Model

The user programmer's model (see Figure 2-1) is common to all M68000 MPUs. The user programmer's model, contains 16, 32-bit, general-purpose registers (D0–D7, A0–A7), a 32-bit program counter, and an 8-bit condition code register. The first eight registers (D0–D7) are used as data registers for byte (8-bit), word (16-bit), and long-word (32-bit) operations. The second set of seven registers (A0–A6) and the user stack pointer (USP) can be used as software stack pointers and base address registers. In addition, the address registers can be used for word and long-word operations. All of the 16 registers can be used as index registers.

When a data register is used as either a source or a destination operand, only the appropriate low-order portion is changed; the remaining high-order portion is neither used nor changed.

2.3.2 Address Registers

Each address register (and the stack pointer) is 32 bits wide and holds a full, 32-bit address. Address registers do not support byte-sized operands. Therefore, when an address register is used as a source operand, either the low-order word or the entire long-word operand is used, depending upon the operation size. When an address register is used as the destination operand, the entire register is affected, regardless of the operation size. If the operation size is word, operands are sign-extended to 32 bits before the operation is performed.

2.4 DATA ORGANIZATION IN MEMORY

Bytes are individually addressable. As shown in Figure 2-5, the high-order byte of a word has the same address as the word. The low-order byte has an odd address, one count higher. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long-word operand is located at address n (n even), then the second word of that operand is located at address n+2.

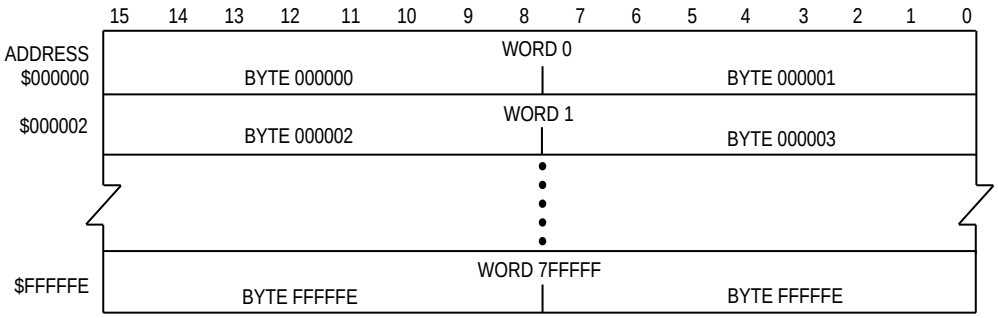


Figure 2-5. Word Organization in Memory

The data types supported by the M68000 MPUs are bit data, integer data of 8, 16, and 32 bits, 32-bit addresses, and binary-coded-decimal data. Each data type is stored in memory as shown in Figure 2-6. The numbers indicate the order of accessing the data from the processor. For the MC68008 with its 8-bit bus, the appearance of data in memory is identical to the all the M68000 MPUs. The organization of data in the memory of the MC68008 is shown in Figure 2-7.

3.6 SYSTEM CONTROL

The system control inputs are used to reset the processor, to halt the processor, and to signal a bus error to the processor. The outputs reset the external devices in the system and signal a processor error halt to those devices. The three system control signals are described in the following paragraphs.

Bus Error ($\overline{BERR}$)

This input signal indicates a problem in the current bus cycle. The problem may be the following:

1. No response from a device.
2. No interrupt vector number returned.
3. An illegal access request rejected by a memory management unit.
4. Some other application-dependent error.

Either the processor retries the bus cycle or performs exception processing, as determined by interaction between the bus error signal and the halt signal.

Reset ($\overline{RESET}$)

The external assertion of this bidirectional signal along with the assertion of $\overline{HALT}$ starts a system initialization sequence by resetting the processor. The processor assertion of $\overline{RESET}$ (from executing a RESET instruction) resets all external devices of a system without affecting the internal state of the processor. To reset both the processor and the external devices, the $\overline{RESET}$ and $\overline{HALT}$ input signals must be asserted at the same time.

Halt ($\overline{HALT}$)

An input to this bidirectional signal causes the processor to stop bus activity at the completion of the current bus cycle. This operation places all control signals in the inactive state and places all three-state lines in the high-impedance state (refer to Table 3-4).

When the processor has stopped executing instructions (in the case of a double bus fault condition, for example), the $\overline{HALT}$ line is driven by the processor to indicate the condition to external devices.

Mode (MODE) (MC68HC001/68EC000)

The MODE input selects between the 8-bit and 16-bit operating modes. If this input is grounded at reset, the processor will come out of reset in the 8-bit mode. If this input is tied high or floating at reset, the processor will come out of reset in the 16-bit mode. This input should be changed only at reset and must be stable two clocks after RESET is negated. Changing this input during normal operation may produce unpredictable results.

The descriptions of the eight states of a write cycle are as follows:

STATE 0 The write cycle starts in S0. The processor places valid function codes on FC2–FC0 and drives R/$\overline{W}$ high (if a preceding write cycle has left R/$\overline{W}$ low).

STATE 1 Entering S1, the processor drives a valid address on the address bus.

STATE 2 On the rising edge of S2, the processor asserts $\overline{AS}$ and drives R/$\overline{W}$ low.

STATE 3 During S3, the data bus is driven out of the high-impedance state as the data to be written is placed on the bus.

STATE 4 At the rising edge of S4, the processor asserts $\overline{LDS}$, or $\overline{DS}$. The processor waits for a cycle termination signal ($\overline{DTACK}$ or $\overline{BERR}$) or $\overline{VPA}$, an M6800 peripheral signal. When $\overline{VPA}$ is asserted during S4, the cycle becomes a peripheral cycle (refer to **Appendix B M6800 Peripheral Interface**). If neither termination signal is asserted before the falling edge at the end of S4, the processor inserts wait states (full clock cycles) until either $\overline{DTACK}$ or $\overline{BERR}$ is asserted.

STATE 5 During S5, no bus signals are altered.

STATE 6 During S6, no bus signals are altered.

STATE 7 On the falling edge of the clock entering S7, the processor negates $\overline{AS}$, $\overline{LDS}$, and $\overline{DS}$. As the clock rises at the end of S7, the processor places the address and data buses in the high-impedance state, and drives R/$\overline{W}$ high. The device negates $\overline{DTACK}$ or $\overline{BERR}$ at this time.

4.1.3 Read-Modify-Write Cycle.

The read-modify-write cycle performs a read operation, modifies the data in the arithmetic logic unit, and writes the data back to the same address. The address strobe ($\overline{AS}$) remains asserted throughout the entire cycle, making the cycle indivisible. The test and set (TAS) instruction uses this cycle to provide a signaling capability without deadlock between processors in a multiprocessing environment. The TAS instruction (the only instruction that uses the read-modify-write cycle) only operates on bytes. Thus, all read-modify-write cycles are byte operations. Figure 4-5 and 4-6 illustrate the read-modify-write cycle operation.

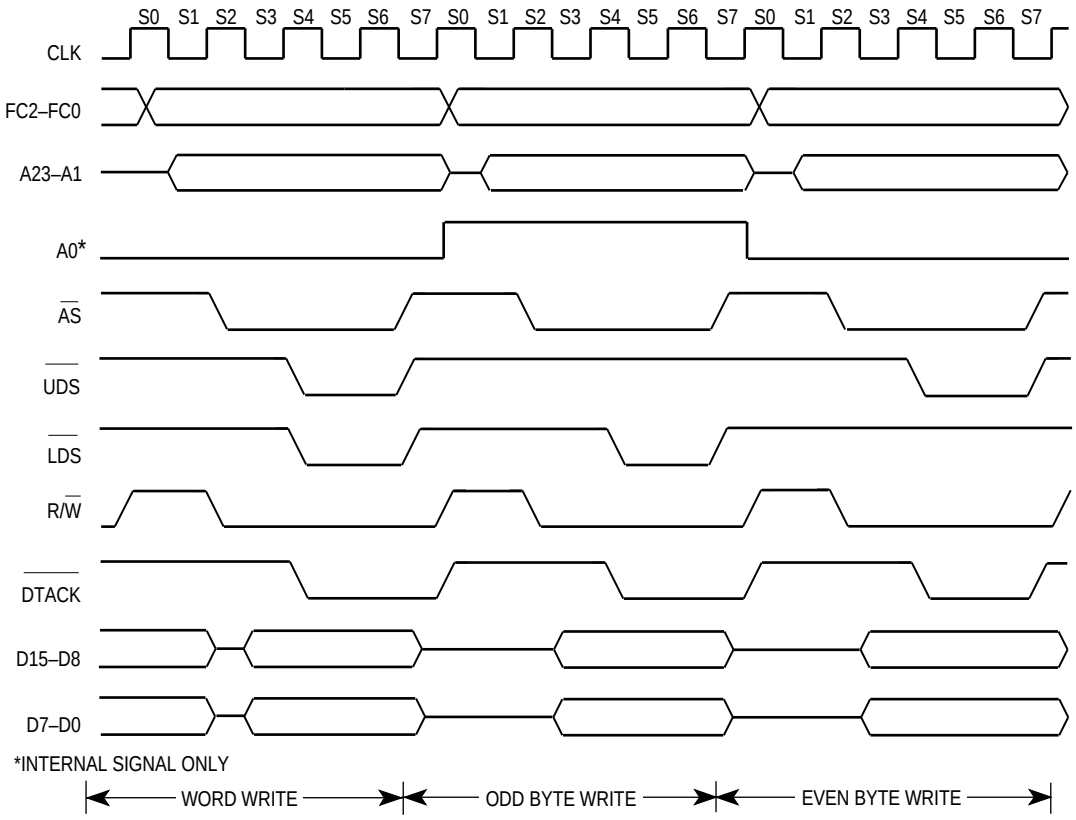


Figure 5-7. Word and Byte Write-Cycle Timing Diagram

The descriptions of the eight states of a write cycle are as follows:

STATE 0 The write cycle starts in S0. The processor places valid function codes on FC2–FC0 and drives R/$\overline{W}$ high (if a preceding write cycle has left R/$\overline{W}$ low).

STATE 1 Entering S1, the processor drives a valid address on the address bus.

STATE 2 On the rising edge of S2, the processor asserts $\overline{AS}$ and drives R/$\overline{W}$ low.

STATE 3 During S3, the data bus is driven out of the high-impedance state as the data to be written is placed on the bus.

STATE 4 At the rising edge of S4, the processor asserts $\overline{UDS}$, or $\overline{LDS}$. The processor waits for a cycle termination signal ($\overline{DTACK}$ or $\overline{BERR}$) or $\overline{VPA}$, an M6800 peripheral signal. When $\overline{VPA}$ is asserted during S4, the cycle becomes a peripheral cycle (refer to **Appendix B M6800 Peripheral Interface**. If neither termination signal is asserted before the falling edge at the end of S4, the processor inserts wait states (full clock cycles) until either $\overline{DTACK}$ or $\overline{BERR}$ is asserted.

STATE 5 During S5, no bus signals are altered.

STATE 6 During S6, no bus signals are altered.

The breakpoint acknowledge cycle is performed by the MC68010 to provide an indication to hardware that a software breakpoint is being executed when the processor executes a breakpoint (BKPT) instruction. The processor neither accepts nor sends data during this cycle, which is otherwise similar to a read cycle. The cycle is terminated by either $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, or as an M6800 peripheral cycle when $\overline{\text{VPA}}$ is asserted, and the processor continues illegal instruction exception processing. Figure 5-12 illustrates the timing diagram for the breakpoint acknowledge cycle.

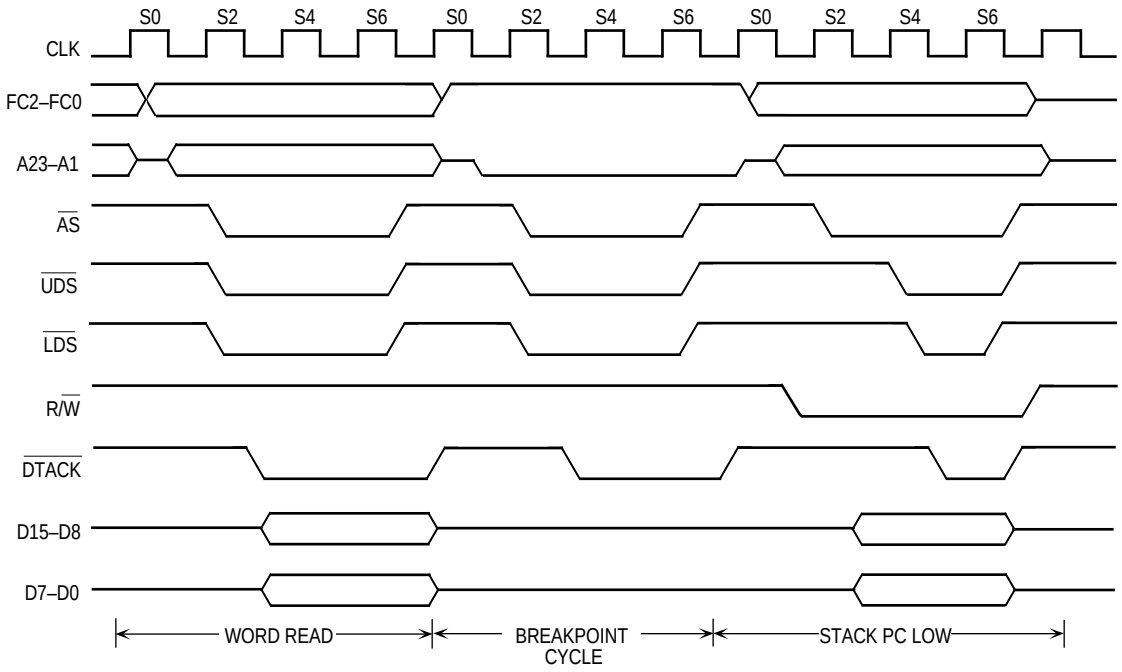


Figure 5-12. Breakpoint Acknowledge Cycle Timing Diagram

5.2 BUS ARBITRATION

Bus arbitration is a technique used by bus master devices to request, to be granted, and to acknowledge bus mastership. Bus arbitration consists of the following:

1. Asserting a bus mastership request
2. Receiving a grant indicating that the bus is available at the end of the current cycle
3. Acknowledging that mastership has been assumed

There are two ways to arbitrate the bus, 3-wire and 2-wire bus arbitration. The MC68000, MC68HC000, MC68EC000, MC68HC001, MC68008, and MC68010 can do 2-wire bus arbitration. The MC68000, MC68HC000, MC68HC001, and MC68010 can do 3-wire bus arbitration. Figures 5-13 and 5-15 show 3-wire bus arbitration and Figures 5-14 and 5-16 show 2-wire bus arbitration. Bus arbitration on all microprocessors, except the 48-pin MC68008 and MC68EC000, $\overline{\text{BGACK}}$ must be pulled high for 2-wire bus arbitration.

bus request signal. When no acknowledge is received before the bus request signal is negated, the processor continues the use of the bus.

5.2.2 Receiving The Bus Grant

The processor asserts $\overline{BG}$ as soon as possible. Normally, this process immediately follows internal synchronization, except when the processor has made an internal decision to execute the next bus cycle but has not yet asserted $\overline{AS}$ for that cycle. In this case, $\overline{BG}$ is delayed until $\overline{AS}$ is asserted to indicate to external devices that a bus cycle is in progress.

$\overline{BG}$ can be routed through a daisy-chained network or through a specific priority-encoded network. Any method of external arbitration that observes the protocol can be used.

5.2.3 Acknowledgment Of Mastership (3-Wire Bus Arbitration Only)

Upon receiving $\overline{BG}$, the requesting device waits until $\overline{AS}$, $\overline{DTACK}$, and $\overline{BGACK}$ are negated before asserting $\overline{BGACK}$. The negation of $\overline{AS}$ indicates that the previous bus master has completed its cycle. (No device is allowed to assume bus mastership while $\overline{AS}$ is asserted.) The negation of $\overline{BGACK}$ indicates that the previous master has released the bus. The negation of $\overline{DTACK}$ indicates that the previous slave has terminated the connection to the previous master. (In some applications, $\overline{DTACK}$ might not be included in this function; general-purpose devices would be connected using $\overline{AS}$ only.) When $\overline{BGACK}$ is asserted, the asserting device is bus master until it negates $\overline{BGACK}$. $\overline{BGACK}$ should not be negated until after the bus cycle(s) is complete. A device relinquishes control of the bus by negating $\overline{BGACK}$.

The bus request from the granted device should be negated after $\overline{BGACK}$ is asserted. If another bus request is pending, $\overline{BG}$ is reasserted within a few clocks, as described in **5.3 Bus Arbitration Control**. The processor does not perform any external bus cycles before reasserting $\overline{BG}$.

5.3 BUS ARBITRATION CONTROL

All asynchronous bus arbitration signals to the processor are synchronized before being used internally. As shown in Figure 5-17, synchronization requires a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (#47, defined in **Section 10 Electrical Characteristic**) has been met. The input asynchronous signal is sampled on the falling edge of the clock and is valid internally after the next falling edge.

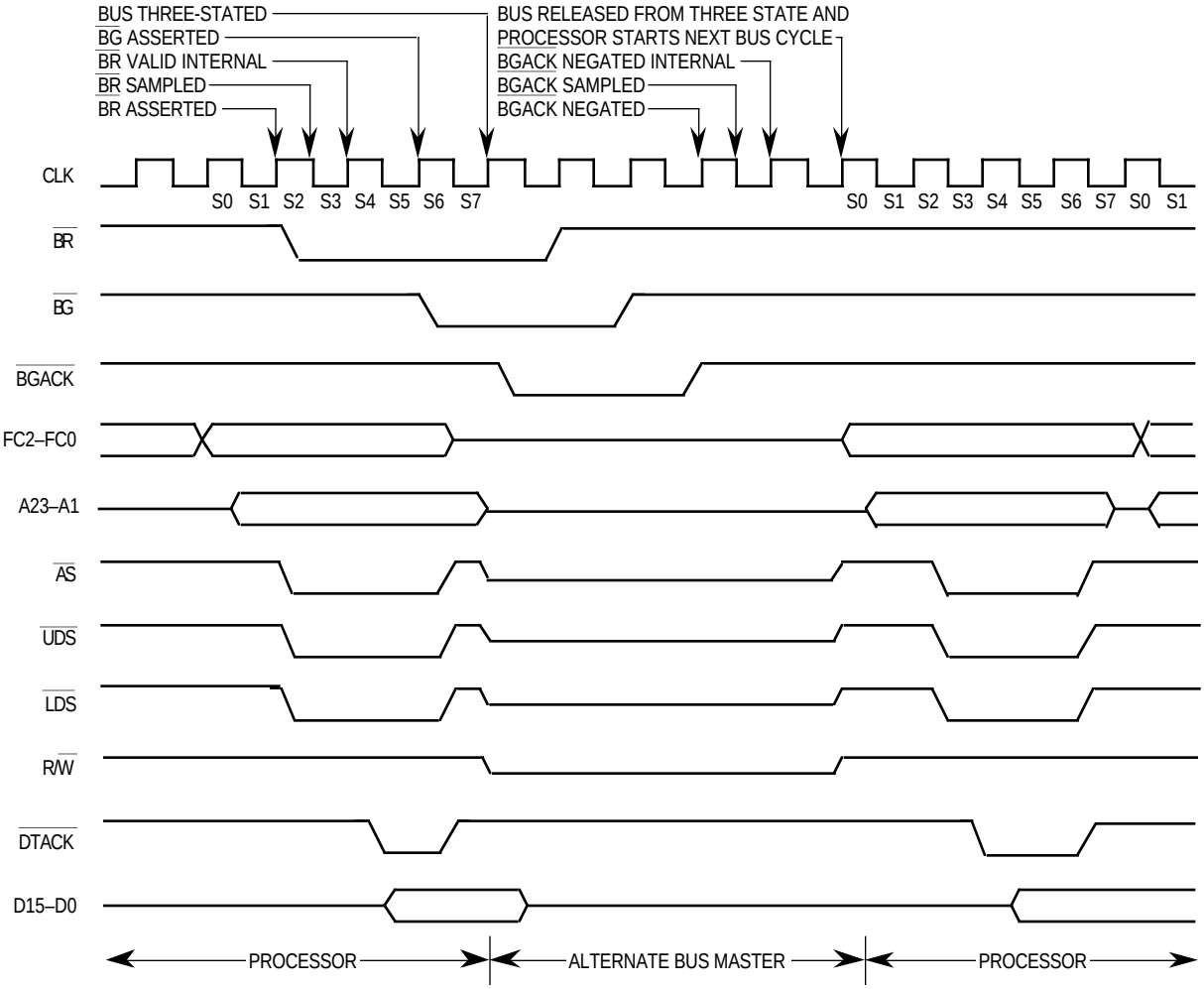


Figure 5-19. 3-Wire Bus Arbitration Timing Diagram—Processor Active

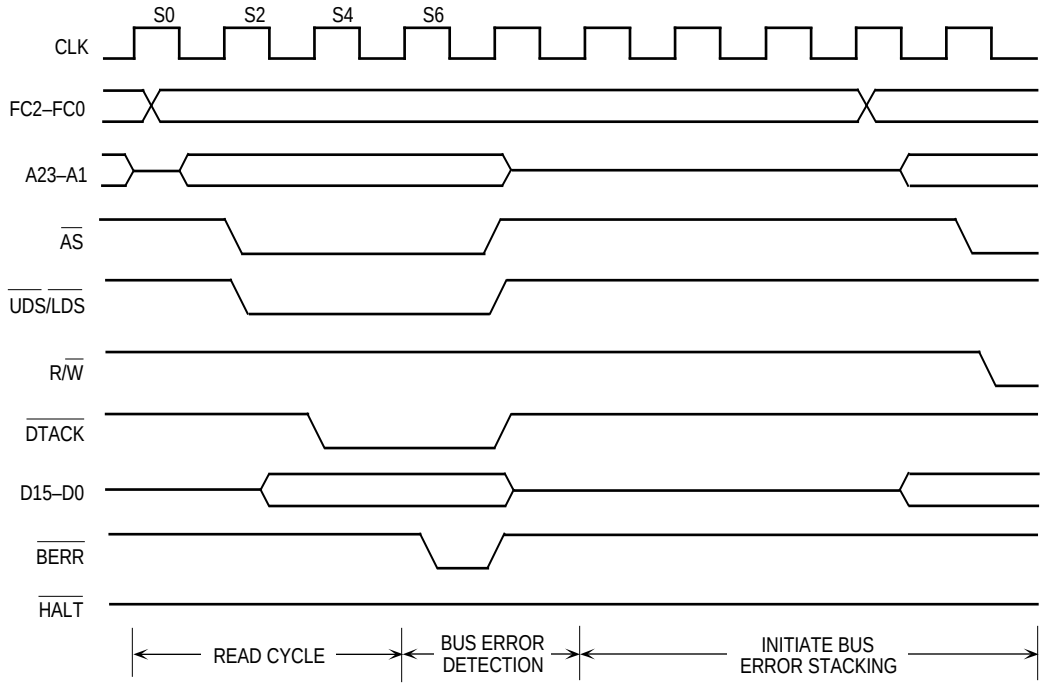


Figure 5-26. Delayed Bus Error Timing Diagram (MC68010)

After the aborted bus cycle is terminated and $\overline{\text{BERR}}$ is negated, the processor enters exception processing for the bus error exception. During the exception processing sequence, the following information is placed on the supervisor stack:

1. Status register
2. Program counter (two words, which may be up to five words past the instruction being executed)
3. Error information

The first two items are identical to the information stacked by any other exception. The error information differs for the MC68010. The MC68000, MC68HC000, MC68HC001, MC68EC000, and MC68008 stack bus error information to help determine and to correct the error. The MC68010 stacks the frame format and the vector offset followed by 22 words of internal register information. The return from exception (RTE) instruction restores the internal register information so that the MC68010 can continue execution of the instruction after the error handler routine completes.

After the processor has placed the required information on the stack, the bus error exception vector is read from vector table entry 2 (offset $08) and placed in the program counter. The processor resumes execution at the address in the vector, which is the first instruction in the bus error handler routine.

Table 8-6. Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
CLR	Byte, Word	**4**(1/0)	**8**(1/1)+
	Long	**6**(1/0)	**12**(1/2)+
NBCD	Byte	**6**(1/0)	**8**(1/1)+
NEG	Byte, Word	**4**(1/0)	**8**(1/1)+
	Long	**6**(1/0)	**12**(1/2)+
NEGX	Byte, Word	**4**(1/0)	**8**(1/1)+
	Long	**6**(1/0)	**12**(1/2)+
NOT	Byte, Word	**4**(1/0)	**8**(1/1)+
	Long	**6**(1/0)	**12**(1/2)+
Scc	Byte, False	**4**(1/0)	**8**(1/1)+
	Byte, True	**6**(1/0)	**8**(1/1)+
TAS	Byte	**4**(1/0)	**14**(2/1)+
TST	Byte, Word	**4**(1/0)	**4**(1/0)+
	Long	**4**(1/0)	**4**(1/0)+

+Add effective address calculation time.

8.6 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Table 8-7 lists the timing data for the shift and rotate instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 8-7. Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	**6+2n**(1/0)	**8**(1/1)+
	Long	**8+2n**(1/0)	—
LSR, LSL	Byte, Word	**6+2n**(1/0)	**8**(1/1)+
	Long	**8+2n**(1/0)	—
ROR, ROL	Byte, Word	**6+2n**(1/0)	**8**(1/1)+
	Long	**8+2n**(1/0)	—
ROXR, ROXL	Byte, Word	**6+2n**(1/0)	**8**(1/1)+
	Long	**8+2n**(1/0)	—

+Add effective address calculation time for word operands.
n is the shift count.

8.9 JMP, JSR, LEA, PEA, AND MOVEM INSTRUCTION EXECUTION TIMES

Table 8-10 lists the timing data for the jump (JMP), jump to subroutine (JSR), load effective address (LEA), push effective address (PEA), and move multiple registers (MOVEM) instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

Table 8-10. JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Instruction	Size	(An)	(An)+	–(An)	(d_{16},An)	(d_8,An,Xn)+	(xxx).W	(xxx).L	(d_{16} PC)	(d_8, PC, Xn)*
JMP	—	**8**(2/0)	—	—	**10** (2/0)	**14** (3/0)	**10** (2/0)	**12** (3/0)	**10** (2/0)	**14** (3/0)
JSR	—	**16** (2/2)	—	—	**18** (2/2)	**22** (2/2)	**18** (2/2)	**20** (3/2)	**18** (2/2)	**22** (2/2)
LEA	—	4(1/0)	—	—	**8**(2/0)	**12** (2/0)	**8**(2/0)	**12** (3/0)	**8**(2/0)	**12** (2/0)
PEA	—	**12** (1/2)	—	—	**16** (2/2)	**20** (2/2)	**16** (2/2)	**20** (3/2)	**16** (2/2)	**20** (2/2)
MOVEM M → R	Word	**12+4n** (3+n/0)	**12+4n** (3+n/0)	—	**16+4n** (4+n/0)	**18+4n** (4+n/0)	**16+4n** (4+n/0)	**20+4n** (5+n/0)	**16+4n** (4n/0)	**18+4n** (4+n/0)
	Long	**12+8n** (3+2n/0)	**12+8n** (3+n/0)	—	**16+8n** (4+2n/0)	**18+8n** (4+2n/0)	**16+8n** (4+2n/0)	**20+8n** (5+2n/0)	**16+8n** (4+2n/0)	**18+8n** (4+2n/0)
MOVEM R → M	Word	**8+4n** (2/n)	—	**8+4n** (2/n)	**12+4n** (3/n)	**14+4n** (3/n)	**12+4n** (3/n)	**16+4n** (4/n)	— —	— —
	Long	**8+8n** (2/2n)	— —	**8+8n** (2/2n)	**12+8n** (3/2n)	**14+8n** (3/2n)	**12+8n** (3/2n)	**16+8n** (4/2n)	— —	— —

n is the number of registers to move.
*The size of the index register (Xn) does not affect the instruction's execution time.

8.10 MULTIPRECISION INSTRUCTION EXECUTION TIMES

Table 8-11 lists the timing data for multiprecision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

The following notation applies in Table 8-11:

Dn — Data register operand
M — Memory operand

Table 9-11. Single Operand Instruction Loop Mode Execution Times

		Loop Continued			Loop Terminated					
		Valid Count, cc False			Valid Count, cc True			Expired Count		
Instruction	Size	(An)	(An)+	–(An)	(An)	(An)+	–(An)	(An)	(An)+	–(An)
CLR	Byte, Word	**10**(0/1)	**10**(0/1)	**12**(0/1)	**18**(2/1)	**18**(2/1)	**20**(2/0)	**16**(2/1)	**16**(2/1)	**18**(2/1)
	Long	**14**(0/2)	**14**(0/2)	**16**(0/2)	**22**(2/2)	**22**(2/2)	**24**(2/2)	**20**(2/2)	**20**(2/2)	**22**(2/2)
NBCD	Byte	**18**(1/1)	**18**(1/1)	**20**(1/1)	**24**(3/1)	**24**(3/1)	**26**(3/1)	**22**(3/1)	**22**(3/1)	**24**(3/1)
NEG	Byte, Word	**16**(1/1)	**16**(1/1)	**18**(2/2)	**22**(3/1)	**22**(3/1)	**24**(3/1)	**20**(3/1)	**20**(3/1)	**22**(3/1)
	Long	**24**(2/2)	**24**(2/2)	**26**(2/2)	**30**(4/2)	**30**(4/2)	**32**(4/2)	**28**(4/2)	**28**(4/2)	**30**(4/2)
NEGX	Byte, Word	**16**(1/1)	**16**(1/1)	**18**(2/2)	**22**(3/1)	**22**(3/1)	**24**(3/1)	**20**(3/1)	**20**(3/1)	**22**(3/1)
	Long	**24**(2/2)	**24**(2/2)	**26**(2/2)	**30**(4/2)	**30**(4/2)	**32**(4/2)	**28**(4/2)	**28**(4/2)	**30**(4/2)
NOT	Byte, Word	**16**(1/1)	**16**(1/1)	**18**(2/2)	**22**(3/1)	**22**(3/1)	**24**(3/1)	**20**(3/1)	**20**(3/1)	**22**(3/1)
	Long	**24**(2/2)	**24**(2/2)	**26**(2/2)	**30**(4/2)	**30**(4/2)	**32**(4/2)	**28**(4/2)	**28**(4/2)	**30**(4/2)
TST	Byte, Word	**12**(1/0)	**12**(1/0)	**14**(1/0)	**18**(3/0)	**18**(3/0)	**20**(3/0)	**16**(3/0)	**16**(3/0)	**18**(3/0)
	Long	**18**(2/0)	**18**(2/0)	**20**(2/0)	**24**(4/0)	**24**(4/0)	**26**(4/0)	**20**(4/0)	**20**(4/0)	**22**(4/0)

9.6 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Tables 9-12 and 9-13 list the timing data for the shift and rotate instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 9-12. Shift/Rotate Instruction Execution Times

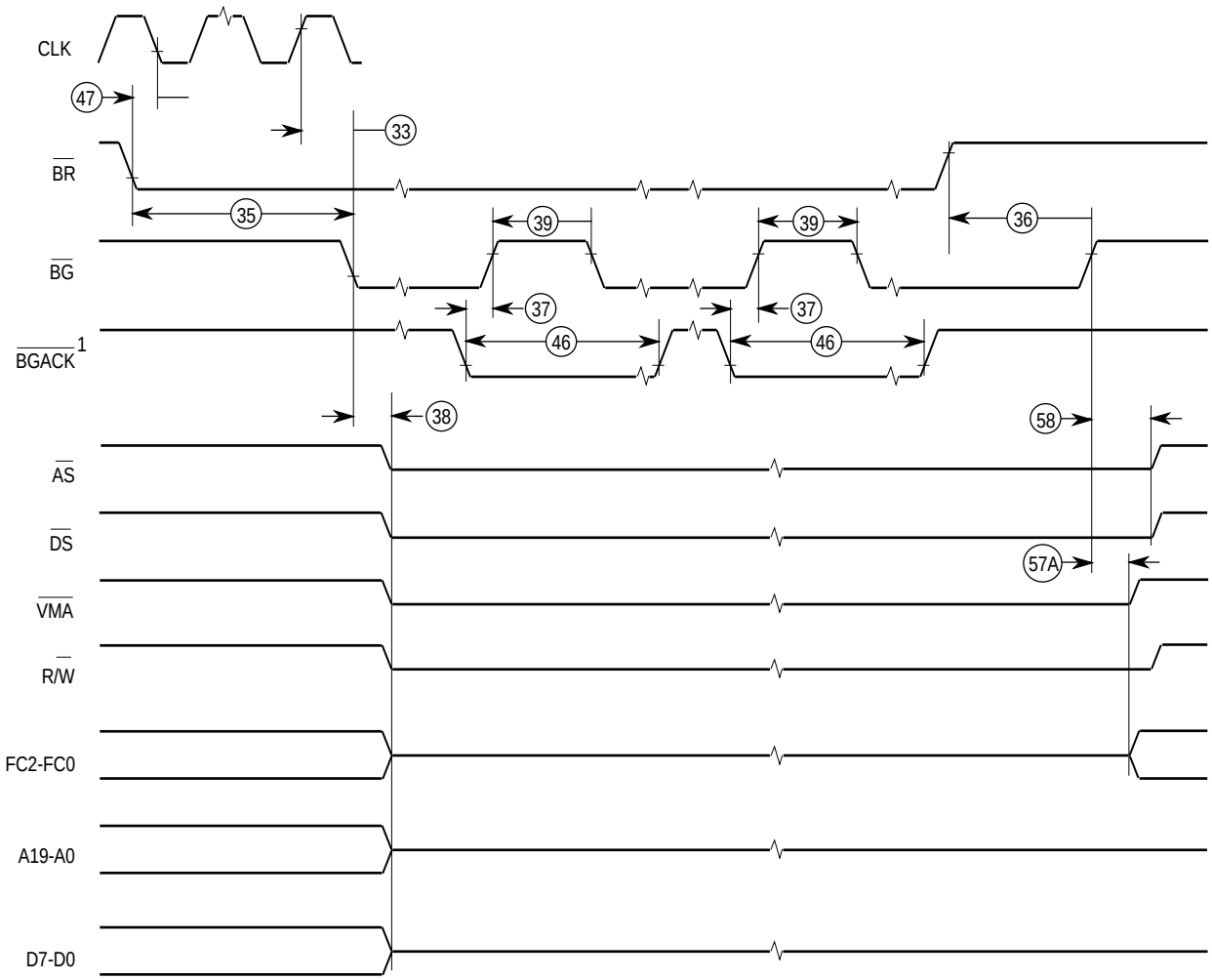
Instruction	Size	Register	Memory*
ASR, ASL	Byte, Word	**6+2n**(1/0)	**8**(1/1)+
	Long	**8+2n**(1/0)	—
LSR, LSL	Byte, Word	**6+2n**(1/0)	**8**(1/1)+
	Long	**8+2n**(1/0)	—
ROR, ROL	Byte, Word	**6+2n**(1/0)	**8**(1/1)+
	Long	**8+2n**(1/0)	—
ROXR, ROXL	Byte, Word	**6+2n**(1/0)	**8**(1/1)+
	Long	**8+2n**(1/0)	—

+Add effective address calculation time.
n is the shift or rotate count.
* Word only.

Table 9-18. Miscellaneous Instruction Execution Times

Instruction	Size	Register	Memory	Register→ Destination**	Source**→ Register
ANDI to CCR	—	**16**(2/0)	—	—	—
ANDI to SR	—	**16**(2/0)	—	—	—
CHK	—	**8**(1/0)+	—	—	—
EORI to CCR	—	**16**(2/0)	—	—	—
EORI to SR	—	**16**(2/0)	—	—	—
EXG	—	**6**(1/0)	—	—	—
EXT	Word	**4**(1/0)	—	—	—
	Long	**4**(1/0)	—	—	—
LINK	—	**16**(2/2)	—	—	—
MOVE from CCR	—	**4**(1/0)	**8**(1/1)+*	—	
MOVE to CCR	—	**12**(2/0)	**12**(2/0)+	—	—
MOVE from SR	—	**4**(1/0)	**8**(1/1)+*	—	—
MOVE to SR	—	**12**(2/0)	**12**(2/0)+	—	—
MOVE from USP	—	**6**(1/0)	—	—	—
MOVE to USP	—	**6**(1/0)	—	—	—
MOVEC	—	—	—	**10**(2/0)	**12**(2/0)
MOVEP	Word	—	—	**16**(2/2)	**16**(4/0)
	Long	—	—	**24**(2/4)	**24**(6/0)
NOP	—	**4**(1/0)	—	—	—
ORI to CCR	—	**16**(2/0)	—	—	—
ORI to SR	—	**16**(2/0)	—	—	—
RESET	—	**130**(1/0)	—	—	—
RTD	—	**16**(4/0)	—	—	—
RTE	Short	**24**(6/0)	—	—	—
	Long, Retry Read	**112**(27/10)	—	—	—
	Long, Retry Write	**112**(26/1)	—	—	—
	Long, No Retry	**110**(26/0)	—	—	—
RTR	—	**20**(5/0)	—	—	—
RTS	—	**16**(4/0)	—	—	—
STOP	—	**4**(0/0)	—	—	—
SWAP	—	**4**(1/0)	—	—	—
TRAPV	—	**4**(1/0)	—	—	—
UNLK	—	**12**(3/0)	—	—	—

+Add effective address calculation time.
+Use nonfetching effective address calculation time.
**Source or destination is a memory location for the MOVEP instruction and a control register for the MOVEC instruction.



NOTES: Waveform measurements for all inputs and outputs are specified at: logic high 2.0 V, logic low = 0.8 V.
1. MC68008 52-Pin Version only.

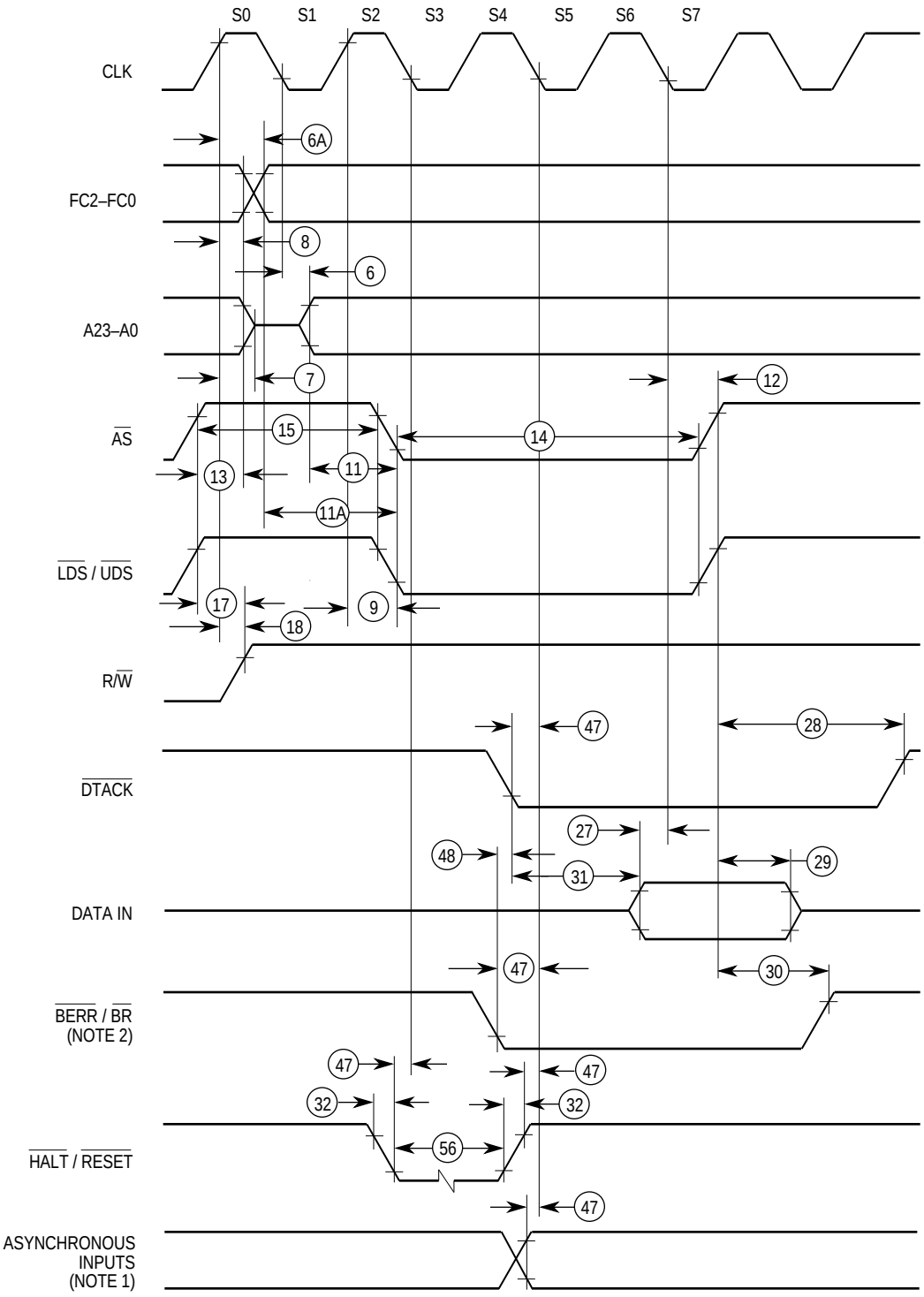
Figure 10-11. Bus Arbitration Timing — Multiple Bus Request
(Applies To All Processors Except The MC68EC000)

10.13 MC68EC000 DC ELECTRICAL SPECIFICATIONS (VCC=5.0 VDC ± 5;PC; GND=0 VDC; $T_A = T_L$ TO T_H)

Characteristic		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	GND–0.3	0.8	V
Input Leakage Current @5.25 V	$\overline{BERR}$, $\overline{BR}$, $\overline{DTACK}$, CLK, $\overline{IPL2}$–$\overline{IPL0}$, $\overline{AVEC}$ MODE, $\overline{HALT}$, $\overline{RESET}$	I_{in}	— —	2.5 20	µA
Three-State (Off State) Input Current @2.4 V/0.4 V	AS, A23–A0, D15–D0, FC2–FC0, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$	I_{TSI}	—	20	µA
Output High Voltage (I_{OH}=–400 µA)	AS, A23–A0, $\overline{BG}$, D15–D0, FC2–FC0, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$	V_{OH}	VCC –0.75	—	V
Output Low Voltage		V_{OL}			V
(IOL = 1.6 mA)	$\overline{HALT}$		—	0.5	
(IOL = 3.2 mA)	A23–A0, $\overline{BG}$, FC2–FC0		—	0.5	
(IOL = 5.0 mA)	$\overline{RESET}$		—	0.5	
(IOL = 5.3 mA)	$\overline{AS}$, D15–D0, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$		—	0.5	
Current Dissipation*	f=8 MHz	I_D	—	25	mA
	f=10 MHz		—	30	
	f=12.5 MHz		—	35	
	f=16.67 MHz		—	50	
	f= 20 MHz		—	70	
Power Dissipation	f=8 MHz	P_D	—	0.13	W
	f=10 MHz		—	0.16	
	f=12.5 MHz		—	0.19	
	f=16.67 MHz		—	0.26	
	f=20 MHz		—	0.38	
Capacitance (Vin=0 V, TA=25°C, Frequency=1 MHz)**		C_{in}	—	20.0	pF
Load Capacitance	$\overline{HALT}$	C_L	—	70	pF
	All Others		—	130	

*Currents listed are with no loading.

**Capacitance is periodically sampled rather than 100% tested.



NOTES:
1. Setup time for the asynchronous inputs IPL2–IPL0 and AVEC (#47) guarantees their recognition at the next falling edge of the clock.
2. BR need fall at this time only to insure being recognized at the end of the bus cycle.
3. Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall is linear between 0.8 V and 2.0 V.

Figure 10-12. MC68EC000 Read Cycle Timing Diagram

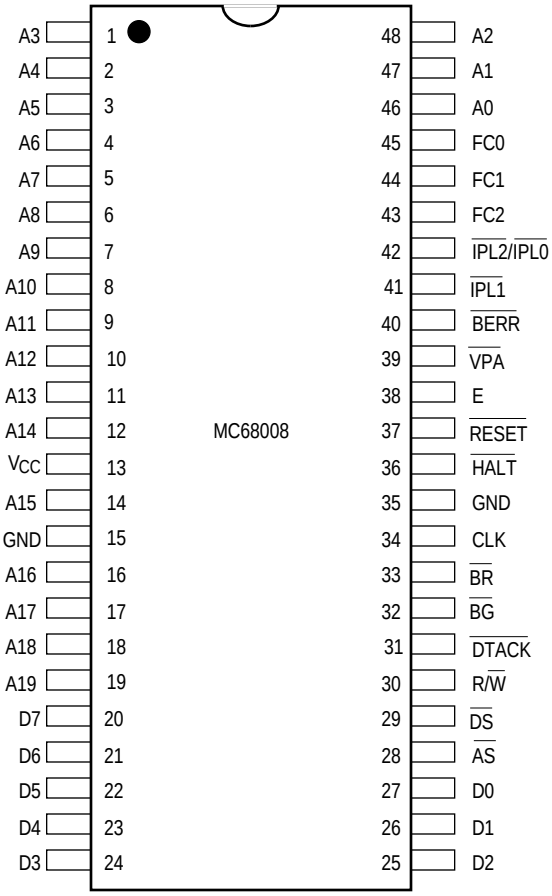


Figure 11-5. 48-Pin Dual In Line

APPENDIX A
MC68010 LOOP MODE OPERATION

In the loop mode of the MC68010, a single instruction is executed repeatedly under control of the test condition, decrement, and branch (DBcc) instruction without any instruction fetch bus cycles. The execution of a single-instruction loop without fetching an instruction provides a highly efficient means of repeating an instruction because the only bus cycles required are those that read and write the operands.

The DBcc instruction uses three operands: a loop counter, a branch condition, and a branch displacement. When this instruction is executed in the loop mode, the value in the low-order word of the register specified as the loop counter is decremented by one and compared to minus one. If the result after decrementing the value is equal to minus one, the result is placed in the loop counter, and the next instruction in sequence is executed. Otherwise, the condition code register is checked against the specified branch condition. If the branch condition is true, the result is discarded, and the next instruction in sequence is executed. When the count is not equal to minus one and the branch condition is false, the branch displacement is added to the value in the program counter, and the instruction at the resulting address is executed.

Figure A-1 shows the source code of a program fragment containing a loop that executes in the loop mode in the MC68010. The program moves a block of data at address SOURCE to a block starting at address DEST. The number of words in the block is labeled LENGTH. If any word in the block at address SOURCE contains zero, the move operation stops, and the program performs whatever processing follows this program fragment.

```
        LEA     SOURCE, A0      Load A Pointer To Source Data
        LEA     DEST, A1        Load A Pointer To Destination
        MOVE.W  #LENGTH, D0     Load The Counter Register
LOOP    MOVE.W  (A0);pl, (A1)+  Loop To Move The Block Of Data
        DBEQ    D0, LOOP        Stop If Data Word Is Zero
```

Figure A-1. DBcc Loop Mode Program Example

The first load effective address (LEA) instruction loads the address labeled SOURCE into address register A0. The second instruction, also an LEA instruction, loads the address labeled DEST into address register A1. Next, a move data from source to destination (MOVE) instruction moves the number of words into data register D0, the loop counter. The last two instructions, a MOVE and a test equal, decrement, and branch (DBEQ), form the loop that moves the block of data. The bus activity required to execute these instructions consists of the following cycles:

asserted in S2. If the bus cycle is a write cycle, the read/write (R/$\overline{W}$) signal is driven low (for write) during S2. In state 3 (S3) of a write cycle, the write data is placed on the data bus, and in state 4 (S4), the data strobes are asserted to indicate valid data on the bus. Next, the processor inserts wait states until it recognizes the assertion of $\overline{VPA}$.

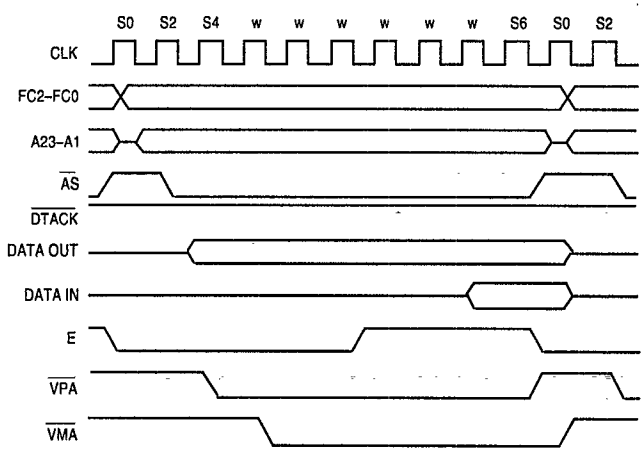


Figure B-4 M6800 Peripheral Timing—Best Case

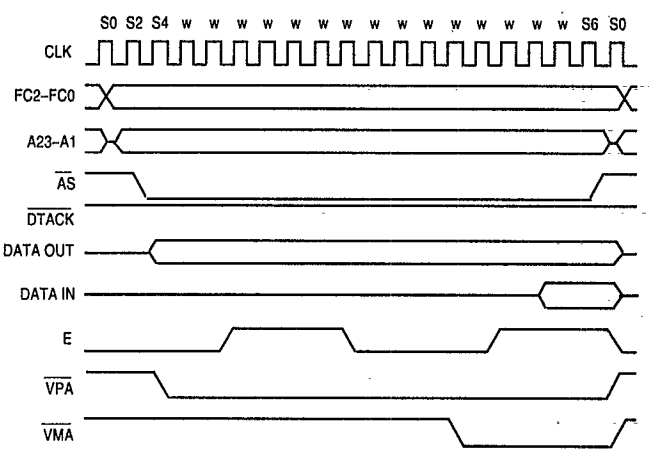


Figure B-5. M6800 Peripheral Timing—Worst Case

The $\overline{VPA}$ input indicates to the processor that the address on the bus is the address of an M6800 device (or an area reserved for M6800 devices) and that the bus should conform to the phase-two transfer characteristics of the M6800 bus. $\overline{VPA}$ is derived by decoding the address bus, conditioned by the address strobe ($\overline{AS}$).

B