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/product-detail/nxp-semiconductors/mc68hc000ei12

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	Section 4 8-Bit Bus Operations	
4.1	Data Transfer Operations	4-1
4.1.1	Read Operations	4-1
4.1.2	Write Cycle	4-3
4.1.3	Read-Modify-Write Cycle	4-5
4.2	Other Bus Operations	4-8
	Section 5 16-Bit Bus Operations	
5.1	Data Transfer Operations	5-1
5.1.1	Read Operations	5-1
5.1.2	Write Cycle	5-4
5.1.3	Read-Modify-Write Cycle	5-7
5.1.4	CPU Space Cycle	5-9
5.2	Bus Arbitration	5-11
5.2.1	Requesting The Bus	5-14
5.2.2	Receiving The Bus Grant	5-15
5.2.3	Acknowledgment of Mastership (3-Wire Arbitration Only)	5-15
5.3	Bus Arbitration Control	5-15
5.4	Bus Error and Halt Operation	5-23
5.4.1	Bus Error Operation	5-24
5.4.2	Retrying The Bus Cycle	5-26
5.4.3	Halt Operation	5-27
5.4.4	Double Bus Fault	5-28
5.5	Reset Operation	5-29
5.6	The Relationship of $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$	5-30
5.7	Asynchronous Operation	5-32
5.8	Synchronous Operation	5-35
	Section 6 Exception Processing	
6.1	Privilege Modes	6-1
6.1.1	Supervisor Mode	6-2
6.1.2	User Mode	6-2
6.1.3	Privilege Mode Changes	6-2
6.1.4	Reference Classification	6-3
6.2	Exception Processing	6-4
6.2.1	Exception Vectors	6-4
6.2.2	Kinds Of Exceptions	6-5
6.2.3	Multiple Exceptions	6-8

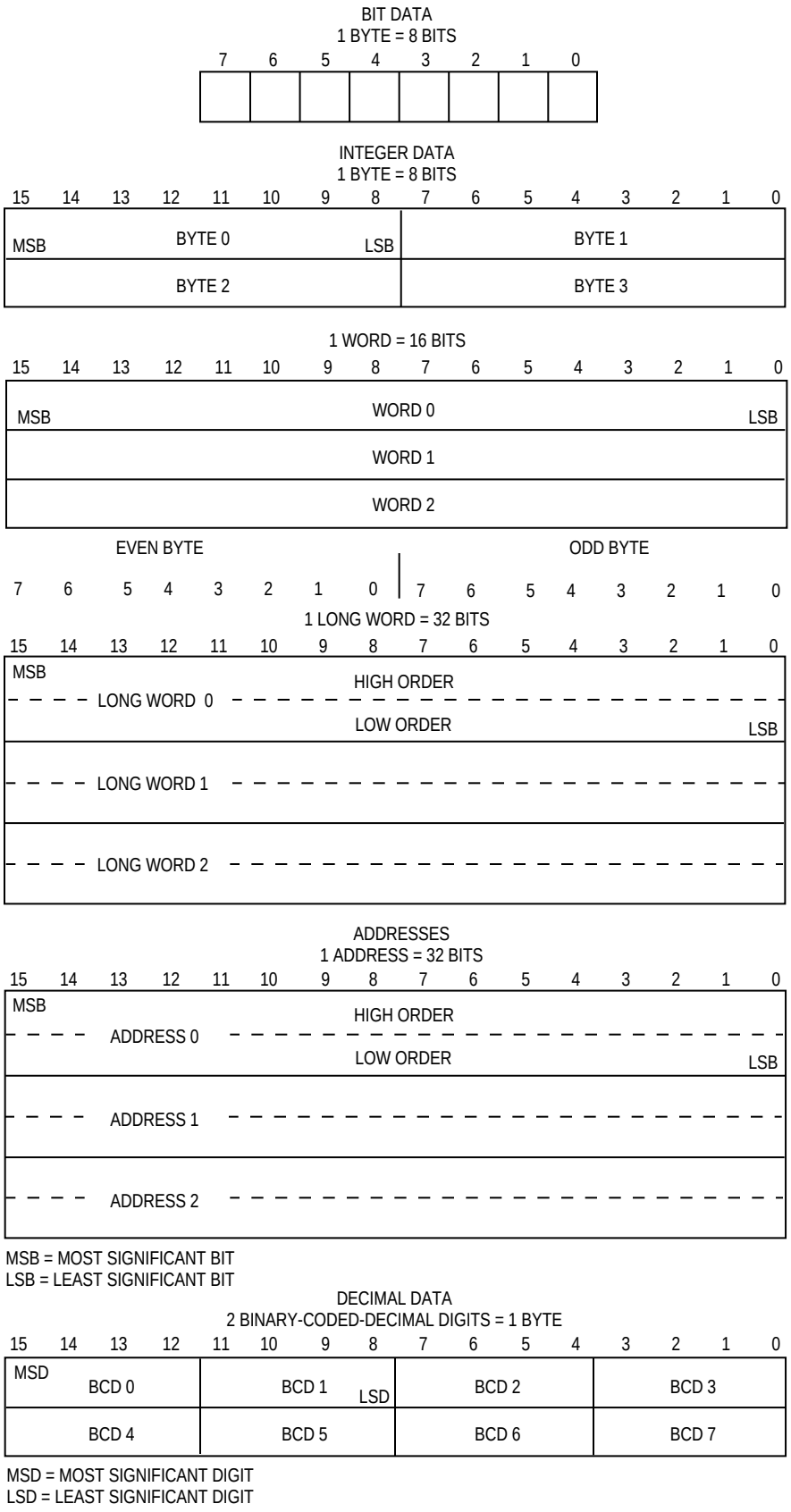


Figure 2-6. Data Organization in Memory

SECTION 5
16-BIT BUS OPERATION

The following paragraphs describe control signal and bus operation for 16-bit bus operations during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation. The 16-bit bus operation devices are the MC68000, MC68HC000, MC68010, and the MC68HC001 and MC68EC000 in 16-bit mode. The MC68HC001 and MC68EC000 select 16-bit mode by pulling mode high or leave it floating during reset.

5.1 DATA TRANSFER OPERATIONS

Transfer of data between devices involves the following signals:

1. Address bus A1 through highest numbered address line
2. Data bus D0 through D15
3. Control signals

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cases, the bus master must deskew all signals it issues at both the start and end of a bus cycle. In addition, the bus master must deskew the acknowledge and data signals from the slave device.

The following paragraphs describe the read, write, read-modify-write, and CPU space cycles. The indivisible read-modify-write cycle implements interlocked multiprocessor communications. A CPU space cycle is a special processor cycle.

5.1.1 Read Cycle

During a read cycle, the processor receives either one or two bytes of data from the memory or from a peripheral device. If the instruction specifies a word or long-word operation, the MC68000, MC68HC000, MC68HC001, MC68EC000, or MC68010 processor reads both upper and lower bytes simultaneously by asserting both upper and lower data strobes. When the instruction specifies byte operation, the processor uses the internal A0 bit to determine which byte to read and issues the appropriate data strobe. When A0 equals zero, the upper data strobe is issued; when A0 equals one, the lower data strobe is issued. When the data is received, the processor internally positions the byte appropriately.

The word read-cycle flowchart is shown in Figure 5-1 and the byte read-cycle flowchart is shown in Figure 5-2. The read and write cycle timing is shown in Figure 5-3 and the word and byte read-cycle timing diagram is shown in Figure 5-4.

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.

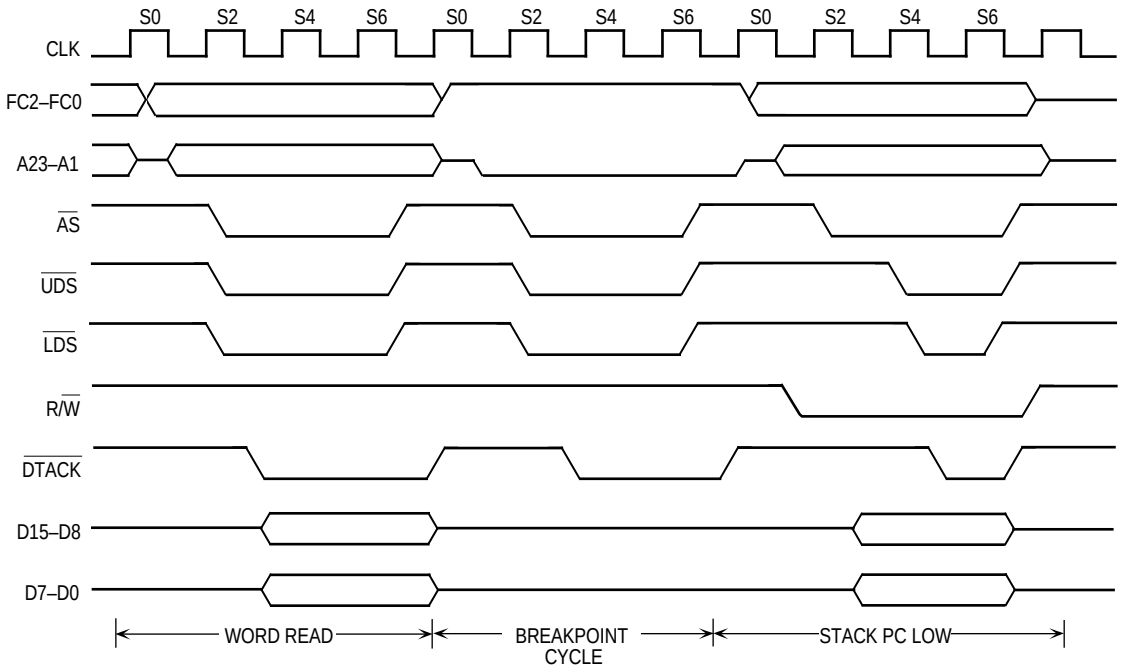


Figure 5-12. Breakpoint Acknowledge Cycle Timing Diagram

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.

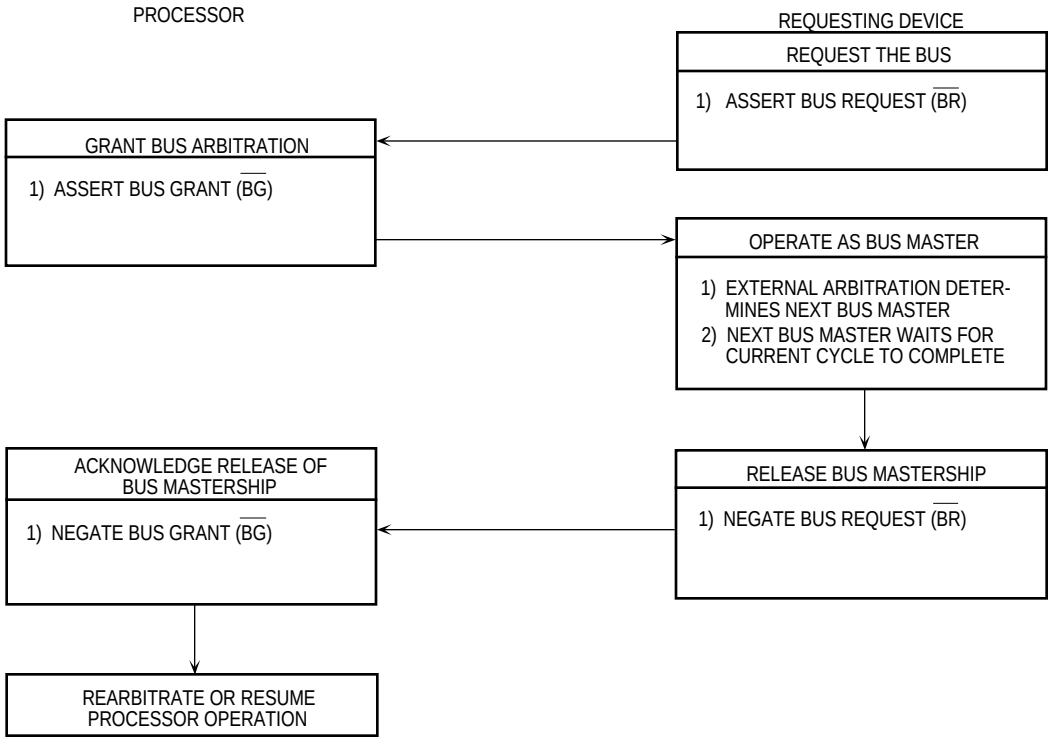


Figure 5-14. 2-Wire Bus Arbitration Cycle Flowchart

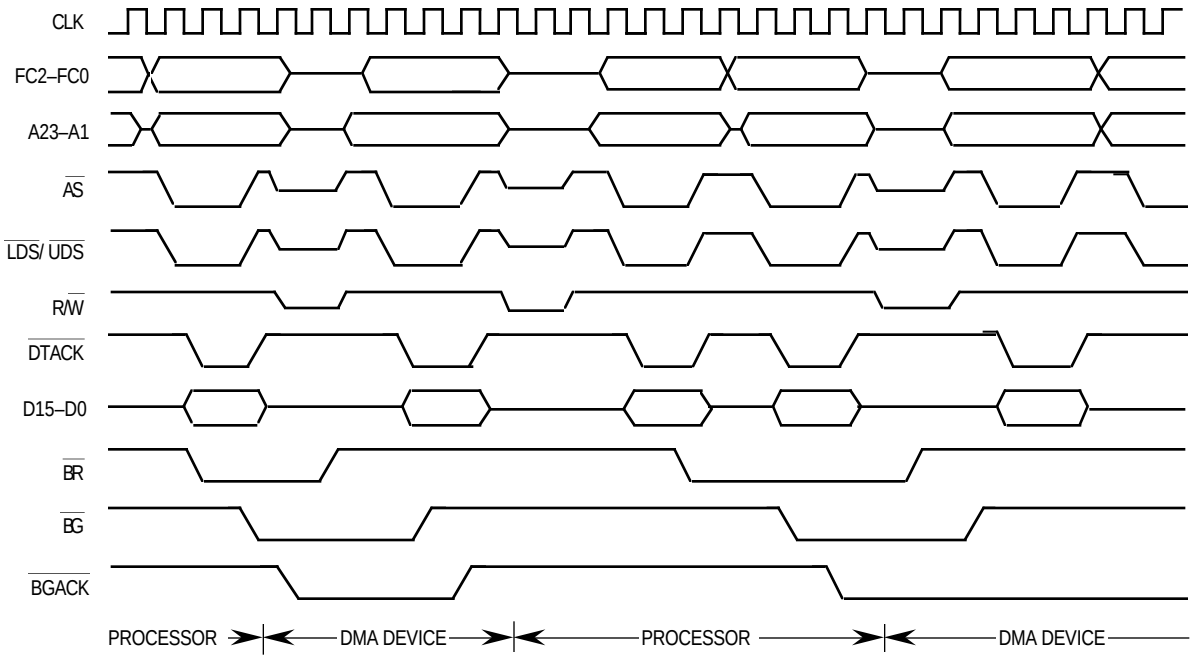


Figure 5-15. 3-Wire Bus Arbitration Timing Diagram (Not Applicable to 48-Pin MC68008 or MC68EC000)

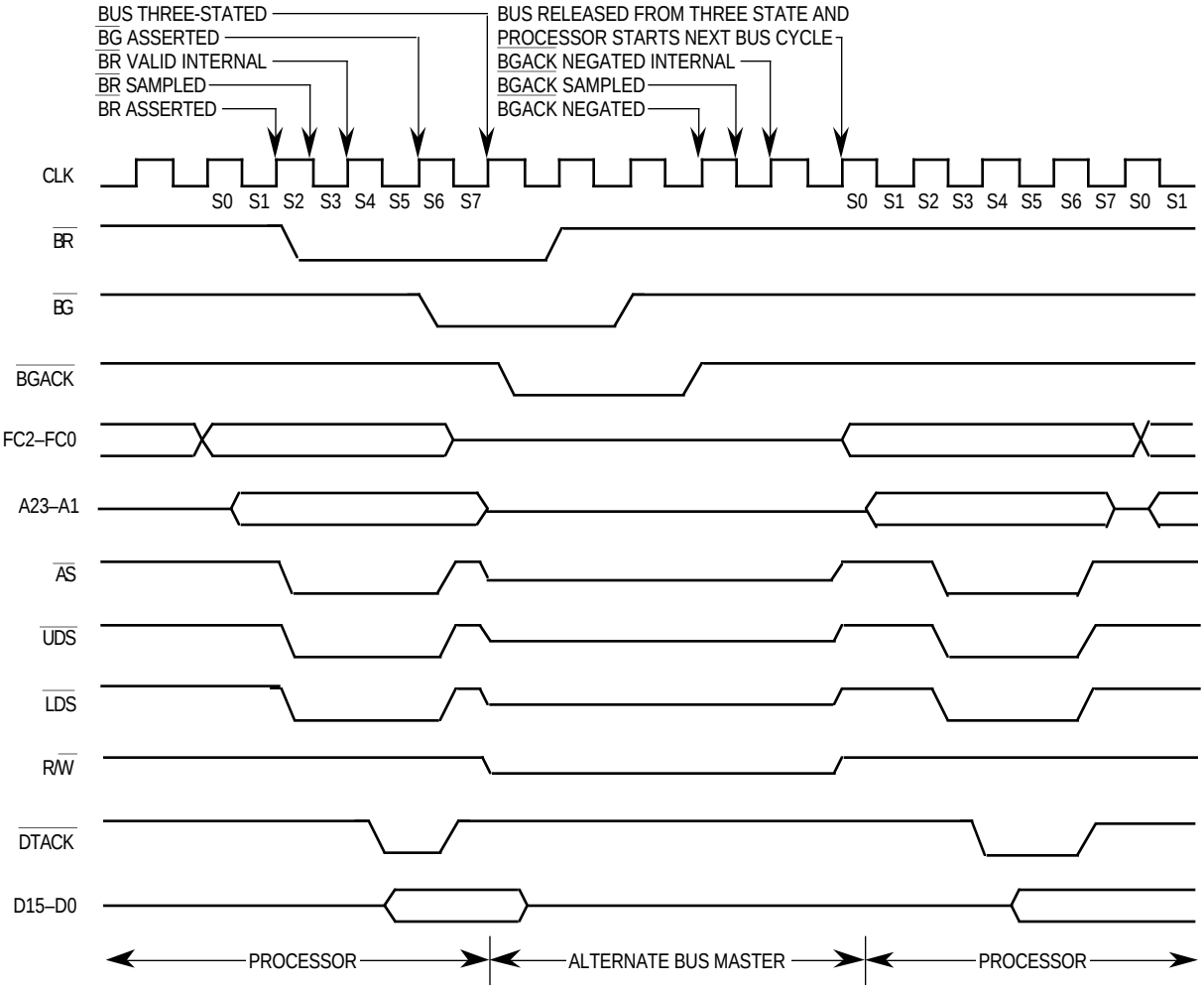


Figure 5-19. 3-Wire Bus Arbitration Timing Diagram—Processor Active

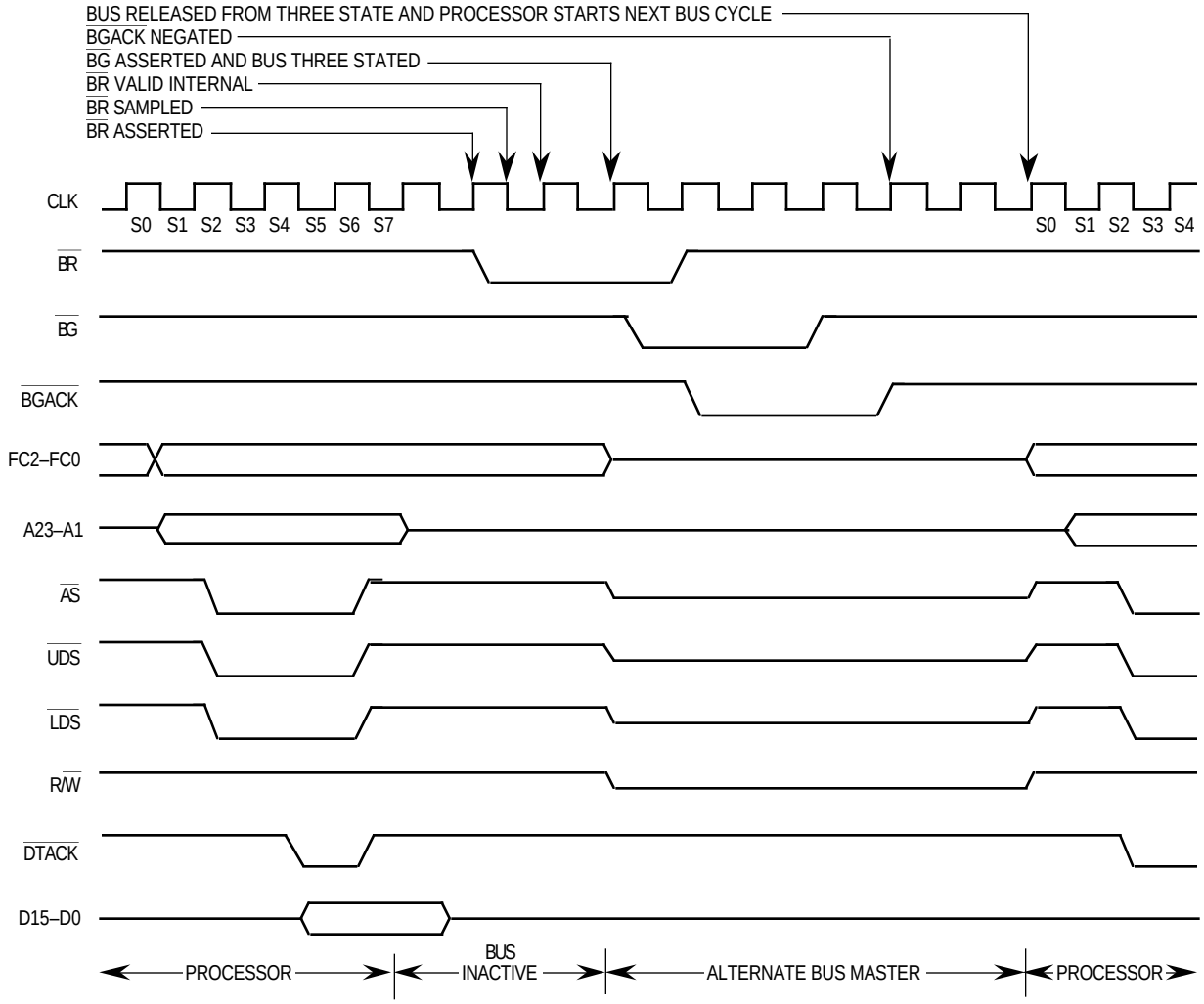


Figure 5-20. 3-Wire Bus Arbitration Timing Diagram—Bus Inactive

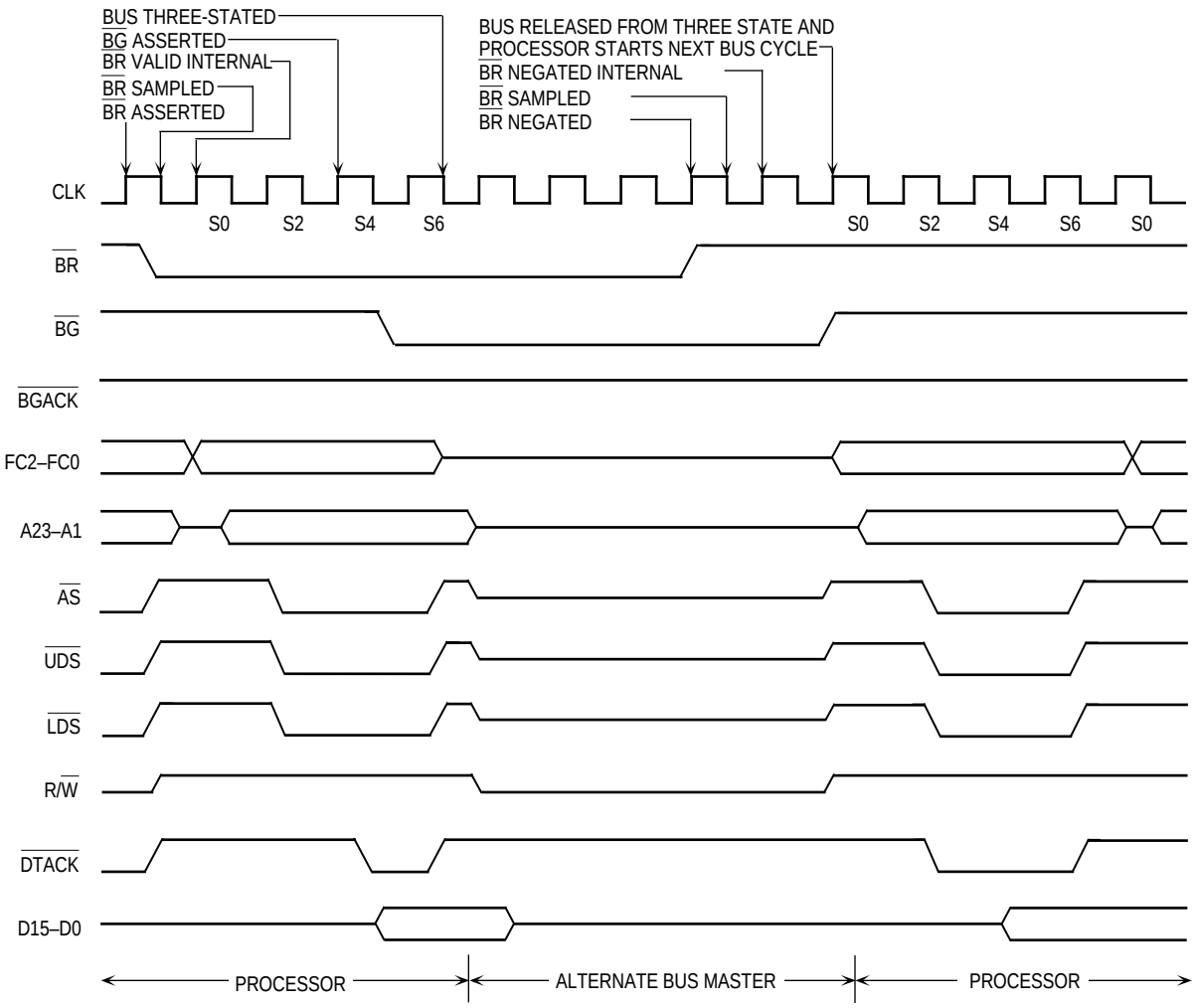


Figure 5-24. 2-Wire Bus Arbitration Timing Diagram—Special Case

5.4. BUS ERROR AND HALT OPERATION

In a bus architecture that requires a handshake from an external device, such as the asynchronous bus used in the M68000 Family, the handshake may not always occur. A bus error input is provided to terminate a bus cycle in error when the expected signal is not asserted. Different systems and different devices within the same system require different maximum-response times. External circuitry can be provided to assert the bus error signal after the appropriate delay following the assertion of address strobe.

In a virtual memory system, the bus error signal can be used to indicate either a page fault or a bus timeout. An external memory management unit asserts bus error when the page that contains the required data is not resident in memory. The processor suspends execution of the current instruction while the page is loaded into memory. The MC68010 pushes enough information on the stack to be able to resume execution of the instruction following return from the bus error exception handler.

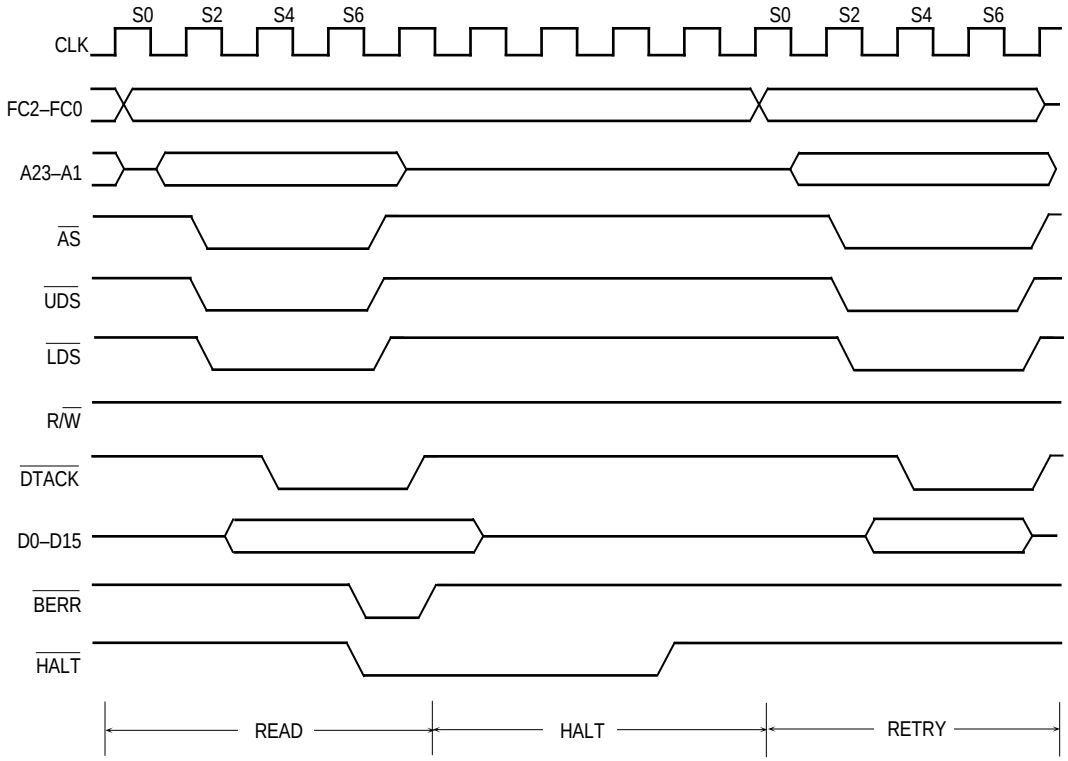


Figure 5-29. Halt Operation Timing Diagram

While the processor is halted, the address bus and the data bus signals are placed in the high-impedance state. Bus arbitration is performed as usual. Should a bus error occur while $\overline{\text{HALT}}$ is asserted, the processor performs the retry operation previously described.

The single-step mode is derived from correctly timed transitions of $\overline{\text{HALT}}$. $\overline{\text{HALT}}$ is negated to allow the processor to begin a bus cycle, then asserted to enter the halt mode when the cycle completes. The single-step mode proceeds through a program one bus cycle at a time for debugging purposes. The halt operation and the hardware trace capability allow tracing of either bus cycles or instructions one at a time. These capabilities and a software debugging package provide total debugging flexibility.

5.4.4 Double Bus Fault

When a bus error exception occurs, the processor begins exception processing by stacking information on the supervisor stack. If another bus error occurs during exception processing (i.e., before execution of another instruction begins) the processor halts and asserts $\overline{\text{HALT}}$. This is called a double bus fault. Only an external reset operation can restart a processor halted due to a double bus fault.

A retry operation does not initiate exception processing; a bus error during a retry operation does not cause a double bus fault. The processor can continue to retry a bus cycle indefinitely if external hardware requests.

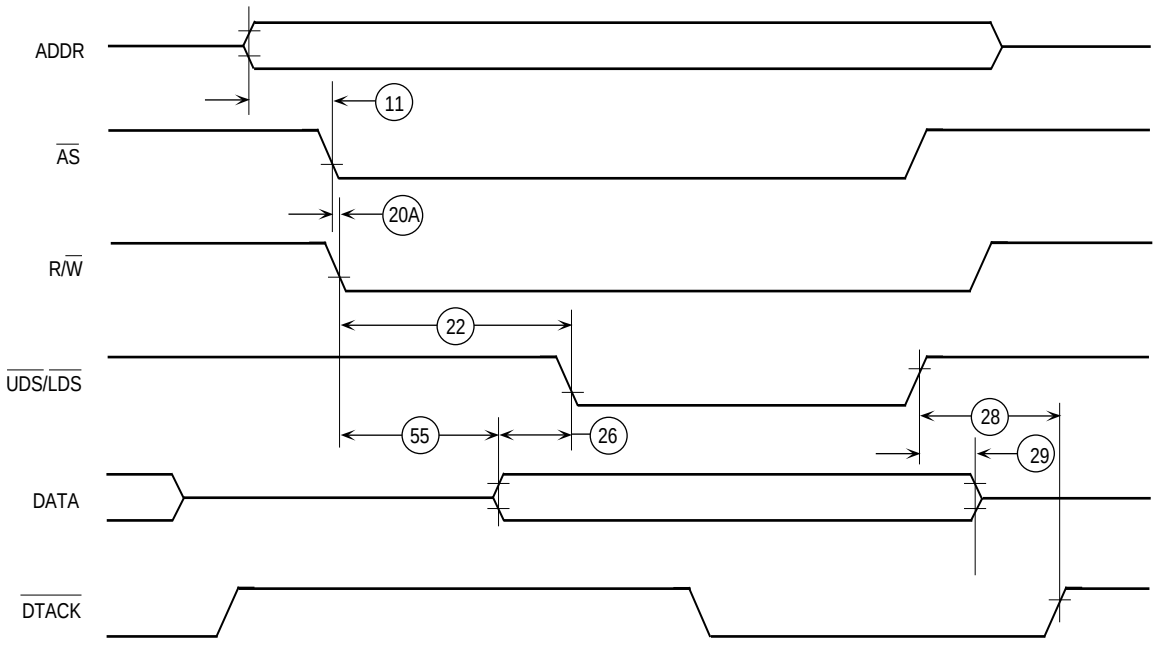


Figure 5-34. Pseudo-Asynchronous Write Cycle

In the MC68010, the BERR signal can be delayed after the assertion of DTACK. Specification #48 is the maximum time between assertion of DTACK and assertion of BERR. If this maximum delay is exceeded, operation of the processor may be erratic.

5.8 SYNCHRONOUS OPERATION

In some systems, external devices use the system clock to generate DTACK and other asynchronous input signals. This synchronous operation provides a closely coupled design with maximum performance, appropriate for frequently accessed parts of the system. For example, memory can operate in the synchronous mode, but peripheral devices operate asynchronously. For a synchronous device, the designer uses explicit timing information shown in **Section 10 Electrical Characteristics**. These specifications define the state of all bus signals relative to a specific state of the processor clock.

The standard M68000 bus cycle consists of four clock periods (eight bus cycle states) and, optionally, an integral number of clock cycles inserted as wait states. Wait states are inserted as required to allow sufficient response time for the external device. The following state-by-state description of the bus cycle differs from those descriptions in **5.1.1 READ CYCLE** and **5.1.2 WRITE CYCLE** by including information about the important timing parameters that apply in the bus cycle states.

STATE 0 The bus cycle starts in S0, during which the clock is high. At the rising edge of S0, the function code for the access is driven externally. Parameter #6A defines the delay from this rising edge until the function codes are valid. Also, the R/W signal is driven high; parameter #18 defines the delay from the same rising edge to the transition of R/W. The minimum value for parameter #18 applies to a read cycle preceded by a write cycle; this value

After the execution of the instruction is complete and before the start of the next instruction, exception processing for a trace begins. A copy is made of the status register. The transition to supervisor mode is made, and the T bit of the status register is turned off, disabling further tracing. The vector number is generated to reference the trace exception vector, and the current program counter and the copy of the status register are saved on the supervisor stack. On the MC68010, the format/offset word is also saved on the supervisor stack. The saved value of the program counter is the address of the next instruction. Instruction execution commences at the address contained in the trace exception vector.

6.3.9 Bus Error

A bus error exception occurs when the external logic requests that a bus error be processed by an exception. The current bus cycle is aborted. The current processor activity, whether instruction or exception processing, is terminated, and the processor immediately begins exception processing. The bus error facility is identical on the all processors; however, the stack frame produced on the MC68010 contains more information. The larger stack frame supports instruction continuation, which supports virtual memory on the MC68010 processor.

6.3.9.1 BUS ERROR. Exception processing for a bus error follows the usual sequence of steps. The status register is copied, the supervisor mode is entered, and tracing is turned off. The vector number is generated to refer to the bus error vector. Since the processor is fetching the instruction or an operand when the error occurs, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are saved. The value saved for the program counter is advanced 2–10 bytes beyond the address of the first word of the instruction that made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. In addition to the usual information, the processor saves its internal copy of the first word of the instruction being processed and the address being accessed by the aborted bus cycle. Specific information about the access is also saved: type of access (read or write), processor activity (processing an instruction), and function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a group 2 exception; the processor is not processing an instruction if it is processing a group 0 or a group 1 exception. Figure 6-7 illustrates how this information is organized on the supervisor stack. If a bus error occurs during the last step of exception processing, while either reading the exception vector or fetching the instruction, the value of the program counter is the address of the exception vector. Although this information is not generally sufficient to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, an address error, or a reset, the processor halts and all processing ceases. This halt simplifies the detection of a catastrophic system failure, since the processor removes itself from the system to

shown in Figure 6-9. If the bus cycle is a read, the data at the fault address should be written to the images of the data input buffer, instruction input buffer, or both according to the data fetch (DF) and instruction fetch (IF) bits.* In addition, for read-modify-write cycles, the status register image must be properly set to reflect the read data if the fault occurred during the read portion of the cycle and the write operation (i.e., setting the most significant bit of the memory location) must also be performed. These operations are required because the entire read-modify-write cycle is assumed to have been completed by software. Once the cycle has been completed by software, the rerun (RR) bit in the special status word is set to indicate to the processor that it should not rerun the cycle when the RTE instruction is executed. If the RR bit is set when an RTE instruction executes, the MC68010 reads all the information from the stack, as usual.

15	14	13	12	11	10	9	8	7 — 3	2 — 0
RR	*	IF	DF	RM	HB	BY	RW	*	FC2–FC0

RR — Rerun flag; 0=processor rerun (default), 1=software rerun
IF — Instruction fetch to the instruction input buffer
DF — Data fetch to the data input buffer
RM — Read-modify-write cycle
HB — High-byte transfer from the data output buffer or to the data input buffer
BY — Byte-transfer flag; HB selects the high or low byte of the transfer register. If BY is clear, the transfer is word.
RW — Read/write flag; 0=write, 1=read
FC — The function code used during the faulted access
* — These bits are reserved for future use by Motorola and will be zero when written by the MC68010.

Figure 6-9. Special Status Word Format

6.3.10 Address Error

An address error exception occurs when the processor attempts to access a word or long-word operand or an instruction at an odd address. An address error is similar to an internally generated bus error. The bus cycle is aborted, and the processor ceases current processing and begins exception processing. The exception processing sequence is the same as that for a bus error, including the information to be stacked, except that the vector number refers to the address error vector. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted.

On the MC68010, the address error exception stacks the same information stacked by a bus error exception. Therefore, the RTE instruction can be used to continue execution of the suspended instruction. However, if the RR flag is not set, the fault address is used when the cycle is retried, and another address error exception occurs. Therefore, the user must be certain that the proper corrections have been made to the stack image and user registers before attempting to continue the instruction. With proper software handling, the address error exception handler could emulate word or long-word accesses to odd addresses if desired.

*If the faulted access was a byte operation, the data should be moved from or to the least significant byte of the data output or input buffer images, unless the high-byte transfer (HB) bit is set. This condition occurs if a MOVEP instruction caused the fault during transfer of bits 8–15 of a word or long word or bits 24–31 of a long word.

Table 7-7. Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
CLR	Byte Word Long	**8**(2/0) **8**(2/0) **10**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
NBCD	Byte	**10**(2/0)	**12**(2/1)+
NEG	Byte Word Long	**8**(2/0) **8**(2/0) **10**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
NEGX	Byte Word Long	**8**(2/0) **8**(2/0) **10**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
NOT	Byte Word Long	**8**(2/0) **8**(2/0) **10**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
Scc	Byte, False Byte, True	**8**(2/0) **10**(2/0)	**12**(2/1)+ **12**(2/1)+
TAS	Byte	**8**(2/0)	**14**(2/1)+
TST	Byte Word Long	**8**(2/0) **8**(2/0) **8**(2/0)	**8**(2/0)+ **8**(2/0)+ **8**(2/0)+

+Add effective address calculation time.

7.6 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Table 7-8 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 7-8. Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory
ASR, ASL	Byte Word Long	**10+2n**(2/0) **10+2n**(2/0) **12+n2**(2/0)	— **16**(2/2)+ —
LSR, LSL	Byte Word Long	**10+2n**(2/0) **10+2n**(2/0) **12+n2**(2/0)	— **16**(2/2)+ —
ROR, ROL	Byte Word Long	**10+2n**(2/0) **10+2n**(2/0) **12+n2**(2/0)	— **16**(2/2)+ —
ROXR, ROXL	Byte Word Long	**10+2n**(2/0) **10+2n**(2/0) **12+n2**(2/0)	— **16**(2/2)+ —

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

Table 8-3. Move Long Instruction Execution Times

Source	Destination								
	Dn	**An**	**(An)**	**(An)+**	**–(An)**	**(d_{16}, An)**	**(d_8, An, Xn)***	**(xxx).W**	**(xxx).L**
Dn	**4**(1/0)	**4**(1/0)	**12**(1/2)	**12**(1/2)	**12**(1/2)	**16**(2/2)	**18**(2/2)	**16**(2/2)	**20**(3/2)
An	**4**(1/0)	**4**(1/0)	**12**(1/2)	**12**(1/2)	**12**(1/2)	**16**(2/2)	**18**(2/2)	**16**(2/2)	**20**(3/2)
(An)	**12**(3/0)	**12**(3/0)	**20**(3/2)	**20**(3/2)	**20**(3/2)	**24**(4/2)	**26**(4/2)	**24**(4/2)	**28**(5/2)
(An)+	**12**(3/0)	**12**(3/0)	**20**(3/2)	**20**(3/2)	**20**(3/2)	**24**(4/2)	**26**(4/2)	**24**(4/2)	**28**(5/2)
–(An)	**14**(3/0)	**14**(3/0)	**22**(3/2)	**22**(3/2)	**22**(3/2)	**26**(4/2)	**28**(4/2)	**26**(4/2)	**30**(5/2)
(d_{16}, An)	**16**(4/0)	**16**(4/0)	**24**(4/2)	**24**(4/2)	**24**(4/2)	**28**(5/2)	**30**(5/2)	**28**(5/2)	**32**(6/2)
(d_8, An, Xn)*	**18**(4/0)	**18**(4/0)	**26**(4/2)	**26**(4/2)	**26**(4/2)	**30**(5/2)	**32**(5/2)	**30**(5/2)	**34**(6/2)
(xxx).W	**16**(4/0)	**16**(4/0)	**24**(4/2)	**24**(4/2)	**24**(4/2)	**28**(5/2)	**30**(5/2)	**28**(5/2)	**32**(6/2)
(xxx).L	**20**(5/0)	**20**(5/0)	**28**(5/2)	**28**(5/2)	**28**(5/2)	**32**(6/2)	**34**(6/2)	**32**(6/2)	**36**(7/2)
(d, PC)	**16**(4/0)	**16**(4/0)	**24**(4/2)	**24**(4/2)	**24**(4/2)	**28**(5/2)	**30**(5/2)	**28**(5/2)	**32**(5/2)
(d, PC, Xn)*	**18**(4/0)	**18**(4/0)	**26**(4/2)	**26**(4/2)	**26**(4/2)	**30**(5/2)	**32**(5/2)	**30**(5/2)	**34**(6/2)
#<data>	**12**(3/0)	**12**(3/0)	**20**(3/2)	**20**(3/2)	**20**(3/2)	**24**(4/2)	**26**(4/2)	**24**(4/2)	**28**(5/2)

*The size of the index register (Xn) does not affect execution time.

8.3 STANDARD INSTRUCTION EXECUTION TIMES

The numbers of clock periods shown in Table 8-4 indicate the times required to perform the operations, store the results, and read the next instruction. 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 (+).

In Table 8-4, the following notation applies:

An — Address register operand
Dn — Data register operand
ea — An operand specified by an effective address
M — Memory effective address operand

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.

9.1 OPERAND EFFECTIVE ADDRESS CALCULATION TIMES

Table 9-1 lists the numbers of clock periods required to compute the effective addresses for instructions. The totals include fetching any extension words, computing the address, and fetching the memory operand. The total number of clock periods, the number of read cycles, and the number of write cycles (zero for all effective address calculations) are shown in the previously described format.

Table 9-1. Effective Address Calculation Times

Addressing Mode		Byte, Word		Long	
		Fetch	No Fetch	Fetch	No Fetch
	Register				
Dn	Data Register Direct	**0**(0/0)	—	**0**(0/0)	—
An	Address Register Direct	**0**(0/0)	—	**0**(0/0)	—
	Memory				
(An)	Address Register Indirect	**4**(1/0)	**2**(0/0)	**8**(2/0)	**2**(0/0)
(An)+	Address Register Indirect with Postincrement	**4**(1/0)	**4**(0/0)	**8**(2/0)	**4**(0/0)
–(An)	Address Register Indirect with Predecrement	**6**(1/0)	**4**(0/0)	**10**(2/0)	**4**(0/0)
(d_{16}, An)	Address Register Indirect with Displacement	**8**(2/0)	**4**(0/0)	**12**(3/0)	**4**(1/0)
(d_8, An, Xn)*	Address Register Indirect with Index	**10**(2/0)	**8**(1/0)	**14**(3/0)	**8**(1/0)
(xxx).W	Absolute Short	**8**(2/0)	**4**(1/0)	**12**(3/0)	**4**(1/0)
(xxx).L	Absolute Long	**12**(3/0)	**8**(2/0)	**16**(4/0)	**8**(2/0)
(d_{16}, PC)	Program Counter Indirect with Displacement	**8**(2/0)	—	**12**(3/0)	—
(d_8, PC, Xn)*	Program Counter Indirect with Index	**10**(2/0)	—	**14**(3/0)	—
#<data>	Immediate	**4**(1/0)	—	**8**(2/0)	—

*The size of the index register (Xn) does not affect execution time.

9.2 MOVE INSTRUCTION EXECUTION TIMES

Tables 9-2, 9-3, 9-4, and 9-5 list the numbers of clock periods for the move instructions. The totals include instruction fetch, operand reads, and operand writes. 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 9-6. Standard Instruction Execution Times

Instruction	Size	op<ea>, An***	op<ea>, Dn	op Dn, <M>
ADD/ADDA	Byte, Word	**8**(1/0)+	**4**(1/0)+	**8**(1/1)+
	Long	**6**(1/0)+	**6**(1/0)+	**12**(1/2)+
AND	Byte, Word	—	**4**(1/0)+	**8**(1/1)+
	Long	—	**6**(1/0)+	**12**(1/2)+
CMP/CMPA	Byte, Word	**6**(1/0)+	**4**(1/0)+	—
	Long	**6**(1/0)+	**6**(1/0)+	—
DIVS	—	—	**122**(1/0)+	—
DIVU	—	—	**108**(1/0)+	—
EOR	Byte, Word	—	**4**(1/0)**	**8**(1/1)+
	Long	—	**6**(1/0)**	**12**(1/2)+
MULS/MULU	—	—	**42**(1/0)+*	—
	—	—	**40**(1/0)*	—
OR	Byte, Word	—	**4**(1/0)+	**8**(1/1)+
	Long	—	**6**(1/0)+	**12**(1/2)+
SUB/SUBA	Byte, Word	**8**(1/0)+	**4**(1/0)+	**8**(1/1)+
	Long	**6**(1/0)+	**6**(1/0)+	**12**(1/2)+

\+ Add effective address calculation time.
* Indicates maximum value.
** Only available address mode is data register direct.
*** Word or long word only.

Table 9-7 Standard Instruction Loop Mode Execution Times

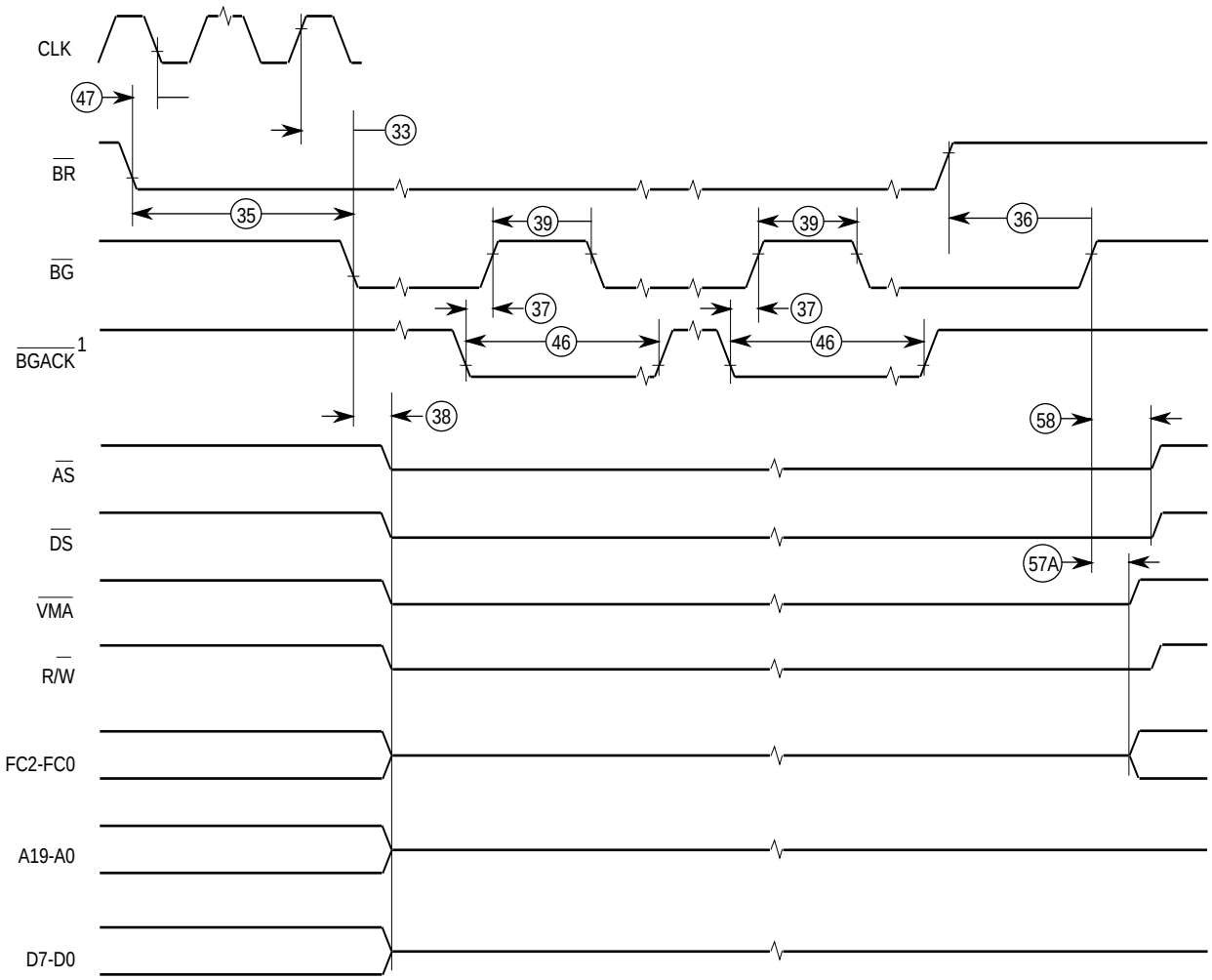
		Loop Continued			Loop Terminated					
		Valid Count cc False			Valid Count cc True			Expired Count		
Instruction	Size	op<ea>, An*	op<ea>, Dn	op Dn, <ea>	op<ea>, An*	op<ea>, Dn	op Dn, <ea>	op<ea>, An*	op<ea>, Dn	op Dn, <ea>
ADD	Byte, Word	**18**(1/0)	**16**(1/0)	**16**(1/1)	**24**(3/0)	**22**(3/0)	**22**(3/1)	**22**(3/0)	**20**(3/0)	**20**(3/1)
	Long	**22**(2/0)	**22**(2/0)	**24**(2/2)	**28**(4/0)	**28**(4/0)	**30**(4/2)	**26**(4/0)	**26**(4/0)	**28**(4/2)
AND	Byte, Word	—	**16**(1/0)	**16**(1/1)	—	**22**(3/0)	**22**(3/1)	—	**20**(3/0)	**20**(3/1)
	Long	—	**22**(2/0)	**24**(2/2)	—	**28**(4/0)	**30**(4/2)	—	**26**(4/0)	**28**(4/2)
CMP	Byte, Word	**12**(1/0)	**12**(1/0)	—	**18**(3/0)	**18**(3/0)	—	**16**(3/0)	**16**(4/0)	—
	Long	**18**(2/0)	**18**(2/0)	—	**24**(4/0)	**24**(4/0)	—	**20**(4/0)	**20**(4/0)	—
EOR	Byte, Word	—	—	**16**(1/0)	—	—	**22**(3/1)	—	—	**20**(3/1)
	Long	—	—	**24**(2/2)	—	—	**30**(4/2)	—	—	**28**(4/2)
OR	Byte, Word	—	**16**(1/0)	**16**(1/0)	—	**22**(3/0)	**22**(3/1)	—	**20**(3/0)	**20**(3/1)
	Long	—	**22**(2/0)	**24**(2/2)	—	**28**(4/0)	**30**(4/2)	—	**26**(4/0)	**28**(4/2)
SUB	Byte, Word	**18**(1/0)	**16**(1/0)	**16**(1/1)	**24**(3/0)	**22**(3/0)	**22**(3/1)	**22**(3/0)	**20**(3/0)	**20**(3/1)
	Long	**22**(2/0)	**20**(2/0)	**24**(2/2)	**28**(4/0)	**26**(4/0)	**30**(4/2)	**26**(4/0)	**24**(4/0)	**28**(4/2)

*Word or long word only.
<ea> may be (An), (An)+, or –(An) only. Add two clock periods to the table value if <ea> is –(An).

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)

SECTION 11
ORDERING INFORMATION AND MECHANICAL DATA

This section provides pin assignments and package dimensions for the devices described in this manual.

11.1 PIN ASSIGNMENTS

Package	68000	68008	68010	68HC000	68HC001	68EC000
64-Pin Dual-In-Line	✔		✔	✔		
68-Terminal Pin Grid Array	✔		✔	✔	✔	
64-Lead Quad Pack						✔
68-Lead Quad Flat Pack	✔		✔	✔	✔	✔
52-Lead Quad		✔				
48-Pin Dual-In-Line		✔				