MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

Advance Information

MC68010/MC68012

16-/32-BIT VIRTUAL MEMORY MICROPROCESSORS

MAY, 1985

ADI942R2

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS
(Continued)

LIST OF TABLES

LIST OF TABLES
(Continued)

SECTION 1
INTRODUCTION

The MC68010 is the third member of a family of advanced microprocessors from Motorola. Utilizing VLSI technology, the MC68010 is a fully-implemented 16-bit microprocessor with 32-bit registers, a rich basic instruction set, and versatile addressing modes. The MC68012 is an expanded address range version of the MC68010 with the additional address pins A24-A29 and A31. A30 is not included due to packaging restrictions. An additional control pin, $\overline{\text{RMC}}$, is provided and can be used as a bus lock to insure system integrity during a read-modify-write operation. Memory management schemes can also use the $\overline{\text{RMC}}$ pin as an advance indication of read-modify-write cycles, as this pin has the same timing as the function code pins. Also, two more GND pins are provided for a better ground plane. With the exception of the additions noted above, all signal functions and timings on the MC68012 are identical to those of the MC68010; therefore, "MC68010" or "A1-A23" in this book can be replaced with "MC68012" or "A1-A29, A31", respectively, in reference to the MC68012.

The MC68010 is fully object code compatible with the earlier members of the M68000 Family and has the added features of virtual memory support and enhanced instruction execution timing.

The MC68010 possesses an asynchronous bus structure with a 24-bit address bus and a 16-bit data bus.

The resources available to the MC68010 user consist of the following:

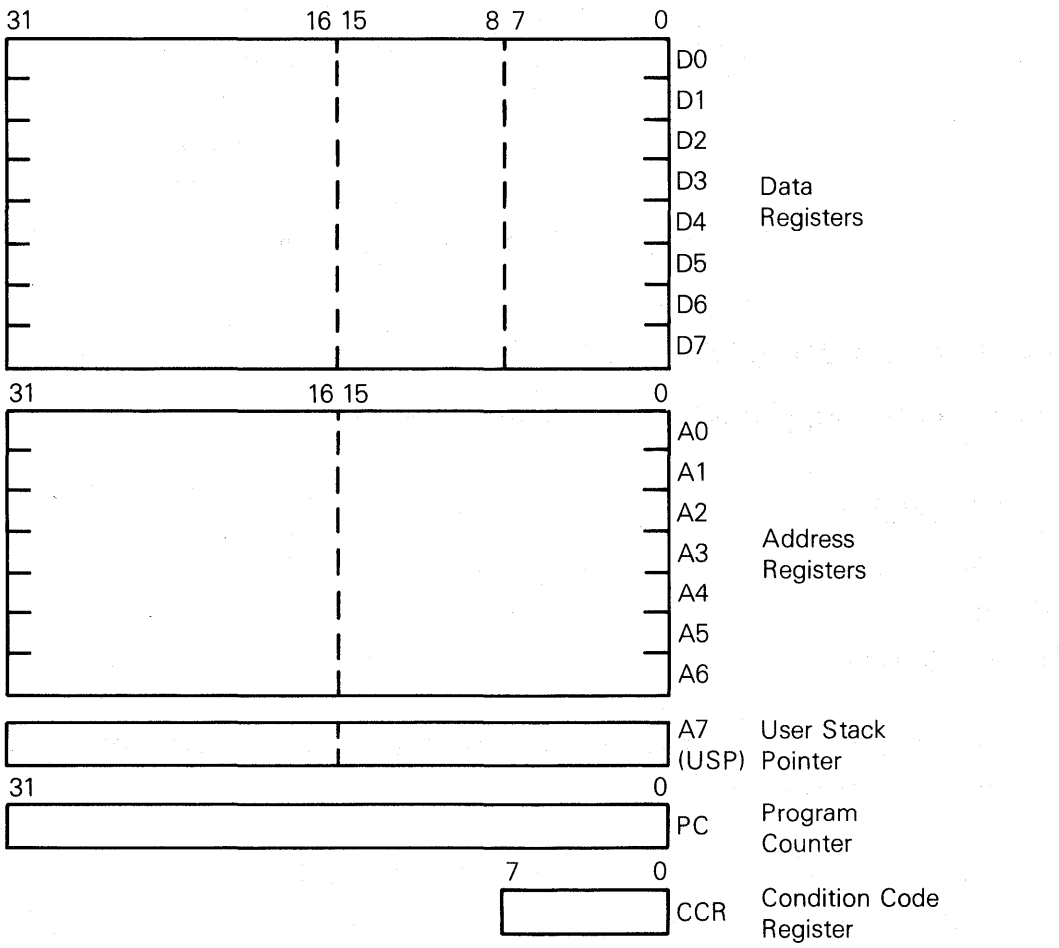
- 17 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- Virtual Memory/Machine Support
- 57 Powerful Instruction Types
- High Performance Looping Instructions
- Operations on Five Main Data Types
- Memory Mapped I/O

The resources available to the MC68010 user are also available to the MC68012 user with the addition of the following:

- 14 Addressing Modes
- Total Direct Address Range of 2 Gigabytes
- $\overline{\text{RMC}}$ Output Pin to Identify a Read-Modify-Write Cycle

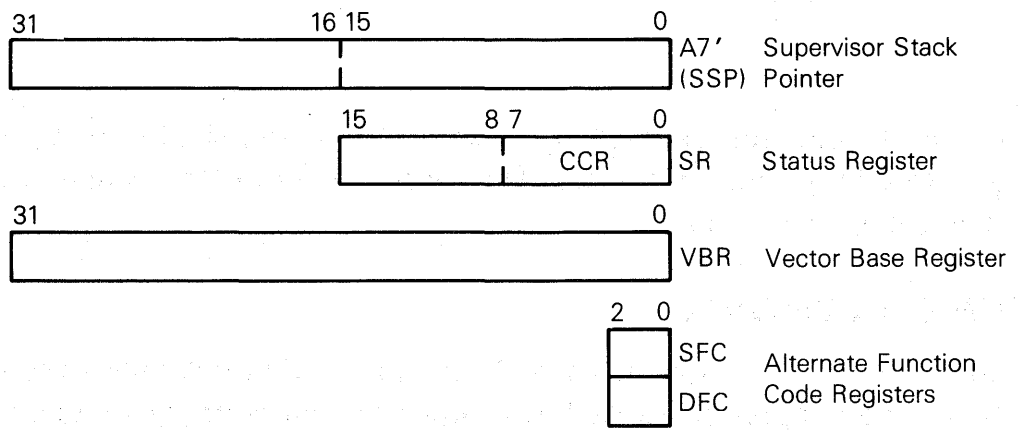
As shown in the programming model (Figures 1-1 and 1-2), the MC68010 offers 17 32-bit general purpose registers, a 32-bit program counter, a 16-bit status register, a 32-bit vector base register, and two 3-bit alternate function code registers. The first eight registers (D0-D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) operations. The second set of seven

registers (A0-A6) and the stack pointers (SSP, USP) may be used as software stack pointers and base address registers. In addition, the address registers may be used for word and long word operations. All of the 17 registers may be used as index registers.



1-286

Figure 1-1. User Programming Model

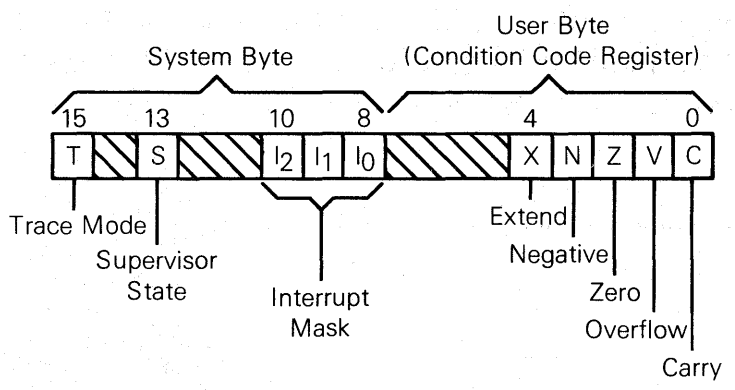


1-287

Figure 1-2. Supervisor Programming Model Supplement

The status register (Figure 1-3) contains the interrupt mask (eight levels available) as well as the condition codes; extend (X), negative (N), zero (Z), overflow (V), and carry (C). Additional status bits indicate that the processor is in the trace (T) mode and in the supervisor (S) or user state.

The vector base register is used to determine the location of the exception vector table in memory to support multiple vector tables. The alternate function code registers allow the supervisor to access user data space or emulate CPU space cycles.



1-288

Figure 1-3. Status Register

1.1 DATA TYPES AND ADDRESSING MODES

Five basic data types are supported. These data types are:

- Bits
- BCD Digits (4 bits)
- Bytes (8 bits)
- Words (16 bits)
- Long Words (32 bits)

In addition, operations on other data types such as memory addresses, status word data, etc., are provided in the instruction set.

The 14 address modes, shown in Table 1-1, include six basic types:

- Register Direct
- Register Indirect
- Absolute
- Program Counter Relative
- Immediate
- Implied

Included in the register indirect addressing modes is the capability to do postincrementing, predecrementing, offsetting, and indexing. The program counter relative mode can also be modified via indexing and offsetting.

1.2 INSTRUCTION SET OVERVIEW

The MC68010 instruction set is shown in Table 1-2. Some additional instructions are variations, or subsets, of these and they appear in Table 1-3. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. By combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned multiply and divide, "quick" arithmetic operations, BCD arithmetic, and expanded operations (through traps). Also, 33 instructions may be used in the loop mode with certain addressing modes and the DBcc instruction to provide 230 high performance string, block manipulation, and extended arithmetic operations.

Table 1-1. Addressing Modes

Mode	Generation
Register Direct Addressing	
Data Register Direct	EA = Dn
Address Register Direct	EA = An
Absolute Data Addressing	
Absolute Short	EA = (Next Word)
Absolute Long	EA = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	EA = (PC) + d_{16}
Relative with Index and Offset	EA = (PC) + (Xn) + d_8
Register Indirect Addressing	
Register Indirect	EA = (An)
Postincrement Register Indirect	EA = (An) An ← An + N
Predecrement Register Indirect	An ← An – N, EA = (An)
Register Indirect with Offset	EA = (An) + d_{16}
Indexed Register Indirect with Offset	EA = (An) + (Xn) + d_8
Immediate Data Addressing	
Immediate	DATA = Next Word(s)
Quick Immediate	Inherent Data
Implied Addressing	
Implied Register	EA = SR, USP, SSP, PC, VBR, SFC, DFC

NOTES:
EA = Effective Address
An = Address Register
Dn = Data Register
Xn = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
( ) = Contents of
d_8 = 8-Bit Offset (Displacement)
d_{16} = 16-Bit Offset (Displacement)
N = 1 for byte, 2 for word, and 4 for long word. If An is the stack pointer and the operand size is byte, N = 2 to keep the stack pointer on a word boundary.
← = Replaces

1-289

Table 1-2. Instruction Set Summary

Mnemonic	Description
ABCD*	Add Decimal with Extend
ADD*	Add
AND*	Logical And
ASL*	Arithmetic Shift Left
ASR*	Arithmetic Shift Right
B_{CC}	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR*	Clear Operand
CMP*	Compare
DB_{CC}	Decrement and Branch Conditionally
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR*	Exclusive Or
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link Stack
LSL*	Logical Shift Left
LSR*	Logical Shift Right

*Loopable Instructions

Mnemonic	Description
MOVE*	Move Source to Destination
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD*	Negate Decimal with Extend
NEG*	Negate
NOP	No Operation
NOT*	One's Complement
OR*	Logical Or
PEA	Push Effective Address
RESET	Reset External Devices
ROL*	Rotate Left without Extend
ROR*	Rotate Right without Extend
ROXL*	Rotate Left with Extend
ROXR*	Rotate Right with Extend
RTD	Return and Deallocate
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD*	Subtract Decimal with Extend
S_{CC}	Set Conditional
STOP	Stop
SUB*	Subtract
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST*	Test
UNLK	Unlink

1-290

Table 1-3. Variations of Instruction Types

Instruction Type	Variation	Description
ADD	**ADD***	Add
	ADDA*	Add Address
	ADDQ	Add Quick
	ADDI	Add Immediate
	ADDX*	Add with Extend
AND	**AND***	Logical And
	ANDI	And Immediate
	ANDI to CCR	And Immediate to Condition Codes
	ANDI to SR	And Immediate to Status Register
CMP	**CMP***	Compare
	CMPA*	Compare Address
	CMPM*	Compare Memory
	CMPI	Compare Immediate
EOR	**EOR***	Exclusive Or
	EORI	Exclusive Or Immediate
	EORI to CCR	Exclusive Or Immediate to Condition Codes
	EORI to SR	Exclusive Or Immediate to Status Register

*Loopable Instructions

Instruction Type	Variation	Description
MOVE	**MOVE***	Move Source to Destination
	MOVEA*	Move Address
	MOVEC	Move Control Register
	MOVEM	Move Multiple Registers
	MOVEP	Move Peripheral Data
	MOVEQ	Move Quick
	MOVES	Move Alternate Address Space
	MOVE from SR	Move from Status Register
	MOVE to SR	Move to Status Register
	MOVE from CCR	Move from Condition Codes
	MOVE to CCR	Move to Condition Codes
	MOVE USP	Move User Stack Pointer
NEG	**NEG***	Negate
	NEGX*	Negate with Extend
OR	**OR***	Logical Or
	ORI	Or Immediate
	ORI to CCR	Or Immediate to Condition Codes
	ORI to SR	Or Immediate to Status Register
SUB	**SUB***	Subtract
	SUBA*	Subtract Address
	SUBI	Subtract Immediate
	SUBQ	Subtract Quick
	SUBX*	Subtract with Extend

1.3 VIRTUAL MEMORY/MACHINE CONCEPTS

In most systems using the MC68010 or MC68012 as the central processor, only a fraction of the 16 megabyte or 2 gigabyte address space will actually contain physical memory. However, by using virtual memory techniques the system can be made to appear to the user to have 16 megabytes or 2 gigabytes of physical memory available to him/her. These techniques have been used for several years in large mainframe computers and more recently in minicomputers and now, with the MC68010, can be fully supported in microprocessor-based systems.

In a virtual memory system, a user program can be written as though it has a large amount of memory available to it when only a small amount of memory is physically present in the system. In a similar fashion, a system can be designed in such a manner as to allow user programs to access other types of devices that are not physically present in the system such as tape drives, disk drives, printers, or CRTs. With proper software emulation, a physical system can be made to appear to a user program as any other computer system and the program may be given full access to all of the resources of that emulated system. Such an emulated system is called a virtual machine.

1.3.1 Virtual Memory

The basic mechanism for supporting virtual memory in computers is to provide only a limited amount of high-speed physical memory that can be accessed directly by the processor while maintaining an image of a much larger "virtual" memory on secondary storage devices such as large capacity disk drives. When the processor attempts to access a location in the virtual memory map that is not currently residing in physical memory (referred to as a page fault), the access to that location is temporarily suspended while the necessary data is fetched from the secondary storage and

placed in physical memory; the suspended access is then completed. The MC68010 provides hardware support for virtual memory with the capability of suspending an instruction's execution when a bus error is signaled and then completing the instruction after the physical memory has been updated as necessary.

The MC68010 uses instruction continuation rather than instruction restart to support virtual memory. With instruction restart, the processor must remember the exact state of the system before each instruction is started in order to restore that state if a page fault occurs during its execution. Then, after the page fault has been repaired, the entire instruction that caused the fault is re-executed. With instruction continuation, when a page fault occurs the processor stores its internal state and then, after the page fault is repaired, restores that internal state and continues execution of the instruction. In order for the MC68010 to utilize instruction continuation, it stores its internal state on the supervisor stack when a bus cycle is terminated with a bus error signal. It then loads the program counter from vector table entry number two (offset $008) and resumes program execution at that new address. When the bus error exception handler routine has completed execution, an RTE instruction is executed which reloads the MC68010 with the internal state stored on the stack, re-runs the faulted bus cycle, and continues the suspended instruction. Instruction continuation has the additional advantage of allowing hardware support for virtual I/O devices. Since virtual registers may be simulated in the memory map, an access to such a register will cause a fault and the function of the register can be emulated by software.

1.3.2 Virtual Machine

One typical use for a virtual machine system is in the development of software such as an operating system for another machine with hardware also under development and not available for programming use. In such a system, the governing operating system (OS) emulates the hardware of the new system and allows the new OS to be executed and debugged as though it were running on the new hardware. Since the new OS is controlled by the governing OS, the new one must execute at a lower privilege level than the governing OS so that any attempts by the new OS to use virtual resources that are not physically present, and should be emulated, will be trapped by the governing OS and handled in software. In the MC68010, a virtual machine may be fully supported by running the new OS in the user mode and the governing OS in the supervisor mode so that any attempts to access supervisor resources or execute privileged instructions by the new OS will cause a trap to the governing OS.

In order to fully support a virtual machine, the MC68010 must protect the supervisor resources from access by user programs. The one supervisor resource that is not fully protected in the MC68000 is the system byte of the status register. In the MC68000, the MOVE from SR instruction allows user programs to test the S bit (in addition to the T bit and interrupt mask) and thus determine that they are running in the user mode. For full virtual machine support, a new OS must not be aware of the fact that it is running in the user mode and thus should not be allowed to access the S bit. For this reason, the MOVE from SR instruction on the MC68010 is a privileged instruction and the MOVE from CCR instruction has been added to allow user programs unhindered access to the condition codes. By making the MOVE from SR instruction privileged, when the new OS attempts to access the S bit, a trap to the governing OS will occur and the SR image passed to the new OS by the governing OS will have the S bit set.

SECTION 2
DATA ORGANIZATION AND ADDRESSING CAPABILITIES

This section contains a description of the registers and the data organization of the MC68010.

2.1 OPERAND SIZE

Operand sizes are defined as follows: a byte equals 8 bits, a word equals 16 bits, and a long word equals 32 bits. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Implicit instructions support some subset of all three sizes.

2.2 DATA ORGANIZATION IN REGISTERS

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers and the stack pointers support address operands of 32 bits. The four control registers support data of 1, 3, 8, 16, or 32 bits depending on the register specified.

2.2.1 Data Registers

Each data register is 32 bits wide. Byte operands occupy the low order 8 bits, word operands the low order 16 bits, and long word operands the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31.

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

2.2.2 Address Registers

Each address register and stack pointer is 32 bits wide and holds a full 32-bit address. Address registers do not support the 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, any other operands are sign extended to 32 bits before the operation is performed.

2.2.3 Control Registers

The status register (SR) is 16 bits wide with the lower byte being accessed as the condition code register (CCR). Not all 16 bits of the SR are defined and will be read as zeroes and ignored when written. Operations to the CCR are word operations; however, the upper byte will be read as all zeroes and ignored when written.

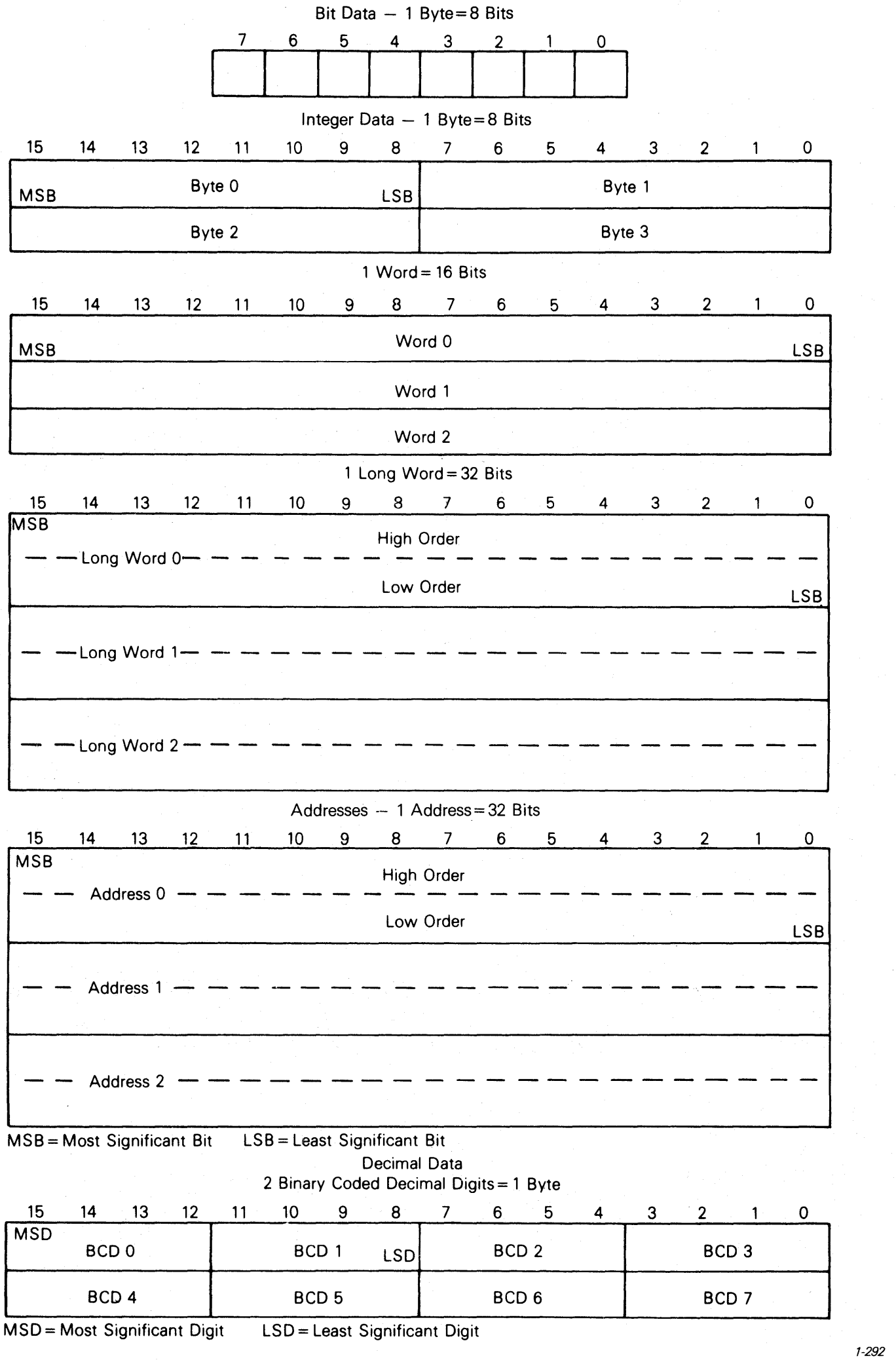


Figure 2-1. Memory Data Organization

The vector base register (VBR) is 32 bits wide and holds a full 32-bit address. All operations involving the VBR are long word operations regardless of whether it is the source or destination operand.

The alternate function code registers (SFC and DFC) are three bits wide and contain the function code values placed on FC0-FC2 during the operand read or write of a MOVES instruction. All transfers to or from the alternate function code registers are 32 bits although the upper 29 bits will be read as zeroes and ignored when written.

2.3 DATA ORGANIZATION IN MEMORY

The data types supported by the MC68010 are: bit data, integer data of 8, 16, or 32 bits, 32-bit addresses and binary coded decimal data. Each of these data types is put in memory, as shown in Figure 2-1. The numbers indicate the order in which the data would be accessed from the processor.

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

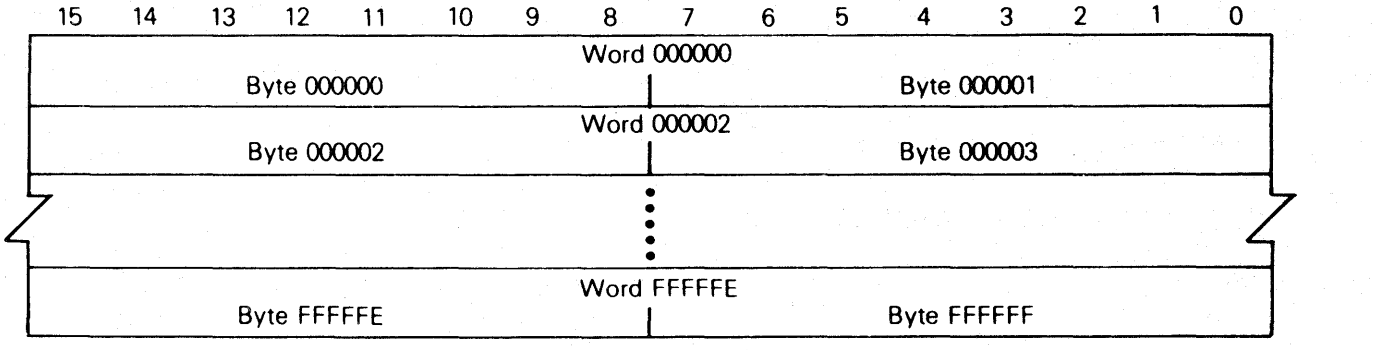


Figure 2-2. Word Organization in Memory

2.4 ADDRESSING

Instructions for the MC68010 contain two kinds of information: the type of function to be performed and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

Register Specification — the number of the register is given in the register field of their instruction.

Effective Address — use of the different effective addressing modes.

Implicit Reference — the definition of certain instructions implies the use of specific registers.

2.5 INSTRUCTION FORMAT

Instructions are from one to five words in length as shown in Figure 2-3. The length of the instruction and the operation to be performed is specified by the first word of the instruction which is called the operation word. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective address mode specified in the operation word.

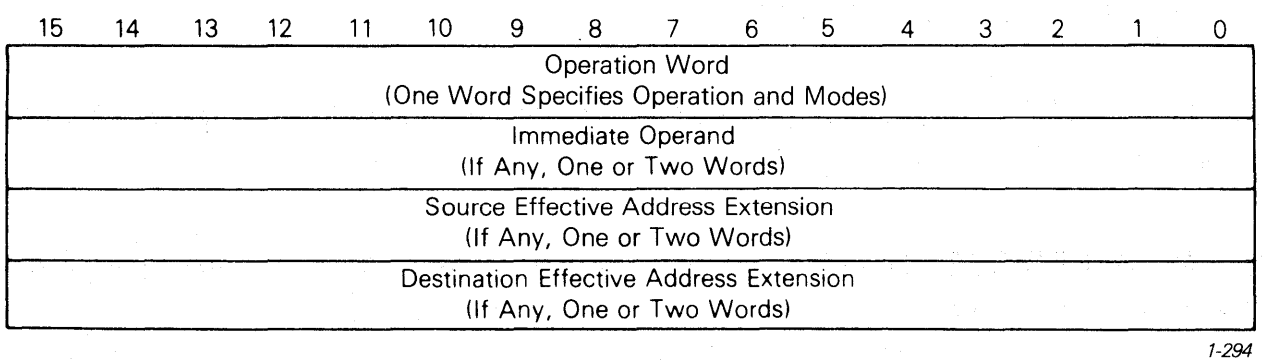


Figure 2-3. Instruction Operation Word General Format

2.6 PROGRAM/DATA REFERENCES

The MC68010 separates memory references into two classes: program references and data references. Program references, as the name implies, are references to that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Generally, operand reads are from the data space. All operand writes are to the data space.

2.7 REGISTER SPECIFICATION

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

2.8 EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 2-4 shows the general format of the single-effective-address instruction operation word. The effective address is composed of two 3-bit fields: the mode field and the register field. The value in the mode field selects the different address modes. The register field contains the number of a register.

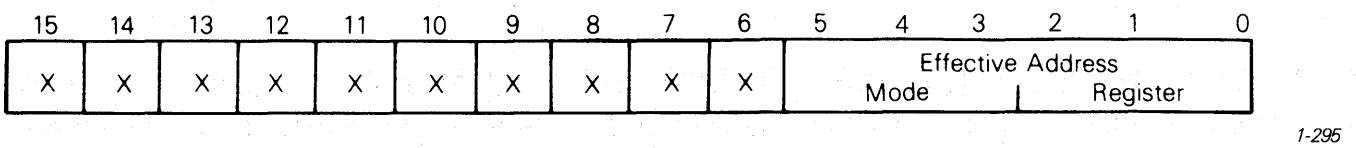


Figure 2-4. Single-Effective-Address Instruction Operation Word

The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in the following word or words and is considered part of the instruction, as shown in Figure 2-3. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

2.8.1 Register Direct Modes

These effective addressing modes specify that the operand is in one of sixteen general purpose registers or one of four control registers.

2.8.1.1 DATA REGISTER DIRECT. The operand is in the data register specified by the effective address register field.

2.8.1.2 ADDRESS REGISTER DIRECT. The operand is in the address register specified by the effective address register field.

2.8.2 Memory Address Modes

These effective addressing modes specify that the operand is in memory and provide the specific address of the operand.

2.8.2.1 ADDRESS REGISTER INDIRECT. The address of the operand is in the address register specified by the register field. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.2.2 ADDRESS REGISTER INDIRECT WITH POSTINCREMENT. The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four depending upon whether the size of the operand is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

2.8.2.3 ADDRESS REGISTER INDIRECT WITH PREDECREMENT. The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four depending upon whether the operand size is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

2.8.2.4 ADDRESS REGISTER INDIRECT WITH DISPLACEMENT. This addressing mode requires one word of extension. The address of the operand is the sum of the address in the address register and the sign-extended 16-bit displacement integer in the extension word. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.2.5 ADDRESS REGISTER INDIRECT WITH INDEX. This addressing mode requires one word of extension. The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents

of the index register. The index may be specified as the sign extended low-order word or the long word in the index register. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.3 Special Address Modes

The special address modes use the effective address register field to specify the special addressing mode instead of a register number.

2.8.3.1 ABSOLUTE SHORT ADDRESS. This addressing mode requires one word of extension. The address of the operand is in the extension word. The 16-bit address is sign extended before it is used. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.3.2 ABSOLUTE LONG ADDRESS. This addressing mode requires two words of extension. The address of the operand is developed by the concatenation of the extension words. The high order part of the address is the first extension word; the low order part of the address is the second extension word. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.3.3 PROGRAM COUNTER WITH DISPLACEMENT. This addressing mode requires one word of extension. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference is classified as a program reference.

2.8.3.4 PROGRAM COUNTER WITH INDEX. This addressing mode requires one word of extension. The address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The index may be specified as the sign extended low-order word or the long word in the index register. The value in the program counter is the address of the extension word. The reference is classified as a program reference.

2.8.3.5 IMMEDIATE DATA. This addressing mode requires either one or two words of extension depending on the size of the operation.

Byte Operation — operand is in the low order byte of extension word

Word Operation — operand is in the extension word

Long Word Operation — operand is in the two extension words, high order 16 bits are in the first extension word, low order 16 bits are in the second extension word.

2.8.3.6 IMPLICIT REFERENCE. Some instructions make implicit reference to the program counter (PC), the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), the status register (SR), the condition code register (CCR), the vector base register (VBR), or the alternate function code registers (SFC or DFC).

A selected set of instructions may reference the status register by means of the effective address field. These are:

ANDI to CCR	EORI to SR	MOVE to CCR
ANDI to SR	ORI to CCR	MOVE to SR
EORI to CCR	ORI to SR	MOVE from SR

2.9 EFFECTIVE ADDRESS ENCODING SUMMARY

Table 2-1 is a summary of the effective addressing modes discussed in the previous paragraphs.

Table 2-1. Effective Address Encoding Summary

Addressing Mode	Mode	Register
Data Register Direct	000	Register Number
Address Register Direct	001	Register Number
Address Register Indirect	010	Register Number
Address Register Indirect with Postincrement	011	Register Number
Address Register Indirect with Predecrement	100	Register Number
Address Register Indirect with Displacement	101	Register Number
Address Register Indirect with Index	110	Register Number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate	111	100

1-296

2.10 SYSTEM STACK

The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes. Address register seven (A7) is the system stack pointer (SP). The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S bit in the status register. If the S bit indicates supervisor state, the SSP is the active system stack pointer and the USP cannot be referenced as an address register. If the S bit indicates user state, the USP is the active system stack pointer, and the SSP cannot be referenced. Each system stack fills from high memory to low memory.

SECTION 3
INSTRUCTION SET SUMMARY

This section contains an overview of the form and structure of the MC68010 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

Data Movement	Shift and Rotate	Program Control
Integer Arithmetic	Bit Manipulation	System Control
Logical	Binary Coded Decimal	

The complete range of instruction capabilities combined with the flexible addressing modes described previously provide a very flexible base for program development.

3.1 DATA MOVEMENT OPERATIONS

The basic method of data acquisition (transfer and storage) is provided by the move (MOVE) instruction. The move instruction and the effective addressing modes allow both address and data manipulation. Data movement instructions allow byte, word, and long word operands to be transferred from memory to memory, memory to register, register to memory, and register to register. Address movement instructions allow word and long word operand transfers and ensure that only legal address manipulations are executed. In addition to the general move instruction there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), unlink stack (UNLK), move quick (MOVEQ), move control register (MOVEC), and move alternate address space (MOVES). Table 3-1 is a summary of the data movement operations.

Table 3-1. Data Movement Operations

Instruction	Operand Size	Operation
EXG	32	Rx ↔ Ry
LEA	32	EA → An
LINK	–	(An) → – (SP) (SP) → An (SP) + displacement → SP
MOVE	8, 16, 32	(EA)s → EAd
MOVEC	32	(Rn) → Cr (Cr) → Rn
MOVEM	16, 32	(EA) → An, Dn (An, Dn) → EA
MOVES	8, 16, 32	(EA) → Rn (Rn) → EA
MOVEP	16, 32	d(An) → Dn Dn → d(An)
MOVEQ	8	#xxx → Dn
PEA	32	EA → – (SP)
SWAP	32	Dn[31:16] ↔ Dn[15:0]
UNLK	–	(An) → Sp (SP) + → An

NOTES:
s = source
d = destination
[ ] = bit numbers
– ( ) = indirect with predecrement
( ) + = indirect with postdecrement
= immediate data

1-297

3.2 INTEGER ARITHMETIC OPERATIONS

The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP), clear (CLR), and negate (NEG). The add and subtract instructions are available for both address and data operations, with data operations accepting all operand sizes. Address operations are limited to legal address size operands (16 or 32 bits). Data, address, and memory compare operations are also available. The clear and negate instructions may be used on all sizes of data operands.

The multiply and divide operations are available for signed and unsigned operands using word multiply to produce a long word product, and a long word dividend with word divisor to produce a word quotient with a word remainder.

Multiprecision and mixed size arithmetic can be accomplished using a set of extended instructions. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

A test operand (TST) instruction that will set the condition codes as a result of a compare of the operand with zero is also available. Test and set (TAS) is a synchronization instruction useful in multiprocessor systems. Table 3-2 is a summary of the integer arithmetic operations.

Table 3-2. Integer Arithmetic Operations

Instruction	Operand Size	Operation
ADD	8, 16, 32 16, 32	(Dn) + (EA) → Dn (EA) + (Dn) → EA (EA) + #xxx → EA (An) + (EA) → An
ADDX	8, 16, 32 16, 32	(Dx) + (Dy) + X → Dx – (Ax) + – (Ay) + X → (Ax)
CLR	8, 16, 32	0 → EA
CMP	8, 16, 32 16, 32	(Dn) – (EA) (EA) – #xxx (Ax) + – (Ay) + (An) – (EA)
DIVS	32 ÷ 16	(Dn)/(EA) → Dn
DIVU	32 ÷ 16	(Dn)/(EA) → Dn
EXT	8 → 16 16 → 32	$(Dn)_8 \rightarrow Dn_{16}$ $(Dn)_{16} \rightarrow Dn_{32}$
MULS	16 X 16 → 32	(Dn) X (EA) → Dn
MULU	16 X 16 → 32	(Dn) X (EA) → Dn
NEG	8, 16, 32	0 – (EA) → EA
NEGX	8, 16, 32	0 – (EA) – X → EA
SUB	8, 16, 32 16, 32	(Dn) – (EA) → Dn (EA) – (Dn) → EA (EA) – #xxx → EA (An) – (EA) → An
SUBX	8, 16, 32	(Dx) – (Dy) – X → Dx – (Ax) – – (Ay) – X → (Ax)
TAS	8	[EA] – 0, 1 → EA[7]
TST	8, 16, 32	(EA) – 0

NOTES:
[ ] = bit number
= immediate data
– = indirect with predecrement
+ = indirect with postdecrement

1-298

3.3 LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are available for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 3-3 is a summary of the logical operations.

Table 3-3. Logical Operations

Instruction	Operand Size	Operation
AND	8, 16, 32	(Dn) Λ (EA) → Dn (EA) Λ (Dn) → EA (EA) Λ #xxx → EA
OR	8, 16, 32	(Dn) v (EA) → Dn (EA) v (Dn) → EA (EA) v #xxx → EA
EOR	8, 16, 32	(EA) ⊕ (Dy) → EA (EA) ⊕ #xxx → EA
NOT	8, 16, 32	~(EA) → EA

NOTES:
~ = invert
# = immediate data
Λ = logical AND
V = logical OR
⊕ = logical exclusive OR

1-299

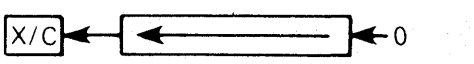
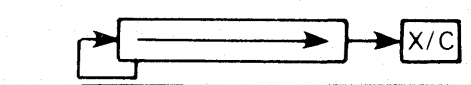
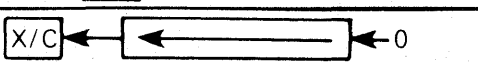
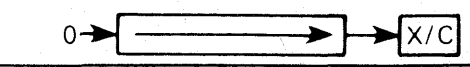
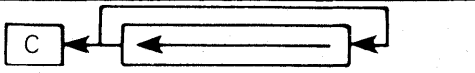
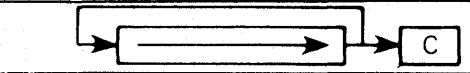
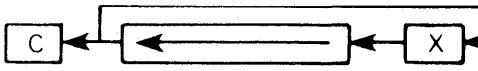
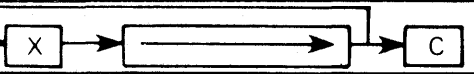
3.4 SHIFT AND ROTATE OPERATIONS

Shift operations in both directions are provided by the arithmetic shift instructions ASR and ASL and logical shift instructions LSR and LSL. The rotate instructions (with and without extend) available are ROXR, ROXL, ROR, and ROL. All shift and rotate operations can be performed in either registers or memory. Register shifts and rotates support all operand sizes and allow a shift count specified in a data register.

Memory shifts and rotates are for word operands only and allow only single-bit shifts or rotates.

Table 3-4 is a summary of the shift and rotate operations.

Table 3-4. Shift and Rotate Operations

Instruction	Operand Size	Operation
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	

1-300

3.5 BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). Table 3-5 is a summary of the bit manipulation operations. (Z is bit 2 of the status register.)

3.6 BINARY CODED DECIMAL OPERATIONS

Multiprecision arithmetic operations on binary coded decimal numbers are accomplished using the following instructions: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). Table 3-6 is a summary of the binary coded decimal-operations.

Table 3-5. Bit Manipulation Operations

Instruction	Operand Size	Operation
BTST	8, 32	~bit of (EA) → Z
BSET	8, 32	~bit of (EA) → Z 1 → bit of EA
BCLR	8, 32	~bit of (EA) → Z 0 → bit of EA
BCHG	8, 32	~bit of (EA) → Z ~bit of (EA) → bit of EA

NOTE: ~ = invert

1-301

Table 3-6. Binary Coded Decimal Operations

Instruction	Operand Size	Operation
ABCD	8	$(Dx)_{10} + (Dy)_{10} + X \rightarrow Dx$ $-(Ax)_{10} + -(Ay)_{10} + x \rightarrow (Ax)$
SBCD	8	$(Dx)_{10} - (Dy)_{10} - X \rightarrow Dx$ $-(Ax)_{10} - -(Ay)_{10} - X \rightarrow (Ax)$
NBCD	8	$0 - (EA)_{10} - X \rightarrow (EA)$

NOTES:
- = indirect with predecrement
+ = indirect with postdecrement

1-302

3.7 PROGRAM CONTROL OPERATIONS

Program control operations are accomplished using a series of conditional and unconditional branch instructions and return instructions. These instructions are summarized in Table 3-7.

The conditional instructions provide setting and branching for the following conditions:

CC — carry clear
CS — carry set
EQ — equal
F — never true
GE — greater or equal
GT — greater than
HI — high
LE — less or equal
LS — low or same
LT — less than
MI — minus
NE — not equal
PL — plus
T — always true
VC — no overflow
VS — overflow

Table 3-7. Program Control Operations

Instruction	Operation
Conditional	
B_{CC}	Branch Conditionally (14 Conditions) 8- and 16-Bit Displacement
DB_{CC}	Test Condition, Decrement, and Branch 16-Bit Displacement
S_{CC}	Set Byte Conditionally (16 Conditions)
Unconditional	
BRA	Branch Always 8- and 16-Bit Displacement
BSR	Branch to Subroutine 8- and 16-Bit Displacement
JMP	Jump
JSR	Jump to Subroutine
Returns	
RTD	Return from Subroutine and and Deallocate Stack
RTR	Return and Restore Condition Codes
RTS	Return from Subroutine

1-303

3.8 SYSTEM CONTROL OPERATIONS

System control operations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the condition code register. These instructions are summarized in Table 3-8.

Table 3-8. System Control Operations

Instruction	Operation
Privileged	
ANDI to SR	Logical AND to Status Register
EORI to SR	Logical EOR to Status Register
MOVE EA to SR	Load New Status Register
MOVE SR to EA	Store Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVES	Move Alternate Address Space
ORI to SR	Logical OR to Status Register
RESET	Reset External Devices
RTE	Return from Exception
STOP	Stop Program Execution
Trap Generating	
CHK	Check Data Register Against Upper Bounds
TRAP	Trap
TRAPV	Trap on Overflow
Condition Code Register	
ANDI to CCR	Logical AND to Condition Codes
EORI to CCR	Logical EOR to Condition Codes
MOVE EA to CCR	Load New Condition Codes
MOVE CCR to EA	Store Condition Codes
ORI to CCR	Logical OR to Condition Codes

1-304

SECTION 4
SIGNAL AND BUS OPERATION DESCRIPTION

This section contains a brief description of the input and output signals. A discussion of bus operation during the various machine cycles and operations is also given.

NOTE

The terms **assertion** and **negation** will be used extensively. This is done to avoid confusion when dealing with a mixture of "active-low" and "active-high" signals. The term assert or assertion is used to indicate that a signal is active or true, independent of whether that level is represented by a high or low voltage. The term negate or negation is used to indicate that a signal is inactive or false.

4.1 SIGNAL DESCRIPTION

The input and output signals can be functionally organized into the groups shown in Figure 4-1. The following paragraphs provide a brief description of the signals and a reference (if applicable) to other paragraphs that contain more detail about the function being performed.

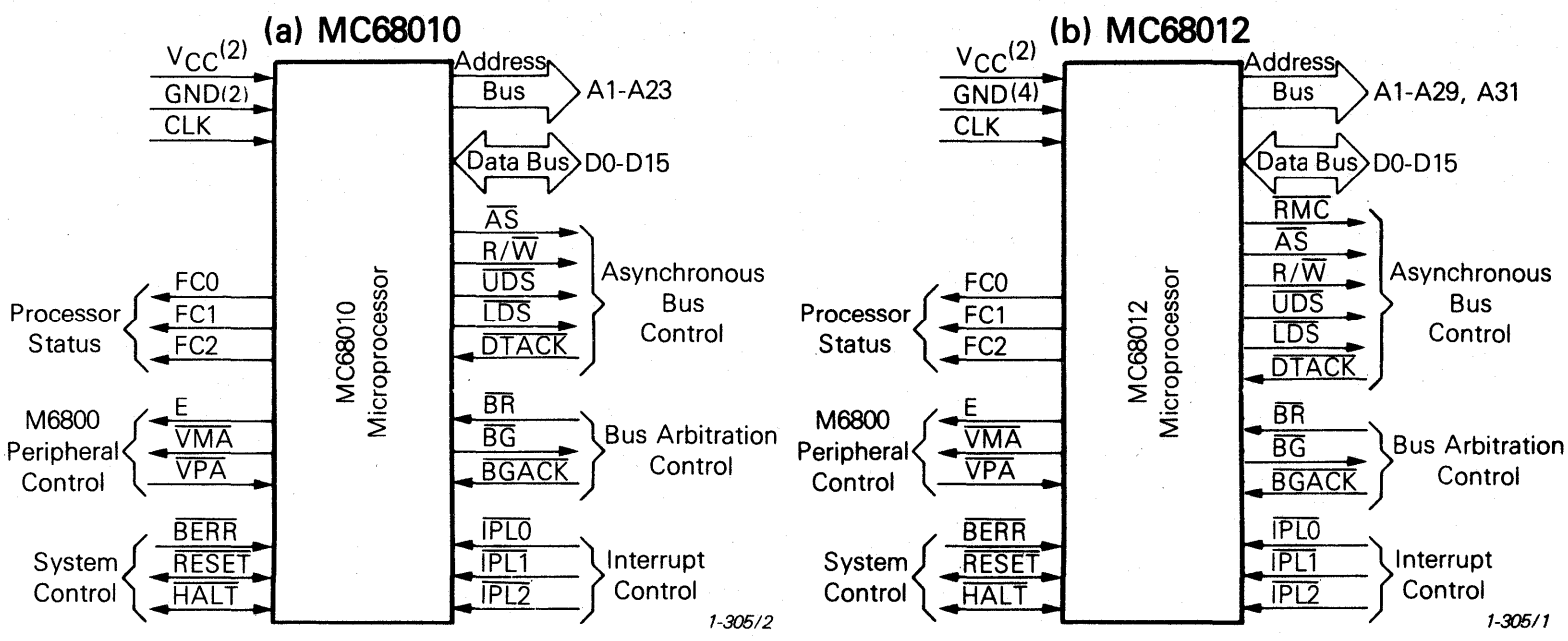


Figure 4-1. Input and Output Signals

4.1.1 Address Bus

The following paragraphs describe the address bus for the MC68010 and MC68012 respectively.

4.1.1.1. MC68010 ADDRESS BUS (A1 THROUGH A23). This 23-bit, unidirectional, three-state bus is capable of addressing 8 megawords of data. It provides the address for bus operation during all cycles except CPU space cycles.

4.1.1.2 MC68012 ADDRESS BUS (A1 THROUGH A29 and A31). This 30-bit, undirectional, three-state bus is capable of linearly addressing 500 megawords of data with A31 differentiating between the two non-contiguous sections; therefore, the total direct addressing range is one gigaword. This bus provides the address for bus operations during all cycles except CPU space cycles.

4.1.2 Data Bus (D0 through D15)

This 16-bit, bidirectional, three-state bus is the general purpose data path. It can transmit and accept data in either word or byte length.

4.1.3 Asynchronous Bus Control

Asynchronous data transfers are handled using the following control signals: read-modify cycle (MC68012 only), address strobe, read/write, upper and lower data strobes, and data transfer acknowledge. These signals are explained in the following paragraphs.

4.1.3.1 READ—MODIFY CYCLE ($\overline{\text{RMC}}$—MC68012 ONLY). This three-state output signal provides an indication that the current bus operation is an indivisible read-modify-write cycle. This signal is asserted for the duration of the read-modify-write sequence. Typically, $\overline{\text{RMC}}$ is used as a bus lock to insure integrity of instructions which use the read-modify-write operation.

4.1.3.2 ADDRESS STROBE ($\overline{\text{AS}}$). This signal indicates that there is a valid address on the address bus.

4.1.3.3 READ/WRITE (R/$\overline{\text{W}}$). This signal defines the data bus transfer as a read or write cycle. The R/$\overline{\text{W}}$ signal also works in conjunction with the data strobes as explained in the following paragraph.

4.1.3.4 UPPER AND LOWER DATA STROBE ($\overline{\text{UDS}}$, $\overline{\text{LDS}}$). These signals control the flow of data on the data bus, as shown in Table 4-1. When the R/$\overline{\text{W}}$ line is high, the processor will read from the data bus as indicated. When the R/$\overline{\text{W}}$ line is low, the processor will write to the data bus as shown.

Table 4-1. Data Strobe Control of Data Bus

$\overline{\text{UDS}}$	$\overline{\text{LDS}}$	R/$\overline{\text{W}}$	D8-D15	D0-D7
High	High	—	No Valid Data	No Valid Data
Low	Low	High	Valid Data Bits 8-15	Valid Data Bits 0-7
High	Low	High	No Valid Data	Valid Data Bits 0-7
Low	High	High	Valid Data Bits 8-15	No Valid Data
Low	Low	Low	Valid Data Bits 8-15	Valid Data Bits 0-7
High	Low	Low	Valid Data Bits 0-7*	Valid Data Bits 0-7
Low	High	Low	Valid Data Bits 8-15	Valid Data Bits 8-15*

* These conditions are a result of current implementation and may not appear on future devices.

1-306

4.1.3.5 DATA TRANSFER ACKNOWLEDGE ($\overline{\text{DTACK}}$). This input indicates that the data transfer is completed. When the processor recognizes $\overline{\text{DTACK}}$ during a read cycle, data is latched one clock cycle later and the bus cycle terminated. When $\overline{\text{DTACK}}$ is recognized during a write cycle, the bus cycle is terminated. Refer to **4.4 ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION.**

4.1.4 Bus Arbitration Control

The three signals, bus request, bus grant, and bus grant acknowledge, form a bus arbitration circuit to determine which device will be the bus master device.

4.1.4.1 BUS REQUEST ($\overline{BR}$). This input is wire ORed with all other devices that could be bus masters. This input indicates to the processor that some other device desires to become the bus master.

4.1.4.2 BUS GRANT ($\overline{BG}$). This output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.

4.1.4.3 BUS GRANT ACKNOWLEDGE ($\overline{BGACK}$). This input indicates that some other device has become the bus master. This signal should not be asserted until the following four conditions are met:

1. a bus grant has been received,
2. address strobe is inactive which indicates that the microprocessor is not using the bus,
3. data transfer acknowledge is inactive which indicates that neither memory nor peripherals are using the bus, and
4. bus grant acknowledge is inactive which indicates that no other device is still claiming bus mastership.

4.1.5 Interrupt Control ($\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$)

These input pins indicate the encoded priority level of the device requesting an interrupt. Level seven is the highest priority while level zero indicates that no interrupts are requested. Level seven cannot be masked. The least significant bit is $\overline{IPL0}$ and the most significant bit is $\overline{IPL2}$. These lines must remain stable until the processor signals interrupt acknowledge (FC0-FC2 are all high, A16-A19 are all high) to insure that the interrupt is recognized.

4.1.6 System Control

The system control inputs are used to either reset or halt the processor and to indicate to the processor that bus errors have occurred. The three system control inputs are explained in the following paragraphs.

4.1.6.1 BUS ERROR ($\overline{BERR}$). This input informs the processor that there is a problem with the cycle currently being executed. Problems may be a result of:

1. nonresponding devices,
2. interrupt vector number acquisition failure,
3. illegal access request as determined by a memory management unit, or
4. other application dependent errors.

The bus error signal interacts with the halt signal to determine if the current bus cycle should be re-executed or if exception processing should be performed.

Refer to **4.2.4 Bus Error and Halt Operation** for additional information about the interaction of the $\overline{BERR}$ and $\overline{HALT}$ signals.

4.1.6.2 RESET ($\overline{\text{RESET}}$). This bidirectional signal line acts to reset (start a system initialization sequence) the processor in response to an external reset signal. An internally generated reset (result of a reset instruction) causes all external devices to be reset and the internal state of the processor is not affected. A total system reset (processor and external devices) is the result of external $\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ signals applied at the same time. Refer to **4.2.5 Reset Operation** for further information.

4.1.6.3 HALT ($\overline{\text{HALT}}$). When this bidirectional line is driven by an external device, it will cause the processor to stop at the completion of the current bus cycle. When the processor has been halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state (refer to Table 4-3). Refer to **4.2.4 Bus Error and Halt Operation** for additional information about the interaction between the $\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ signals.

When the processor has stopped executing instructions, due to a double bus fault condition (refer to **4.2.4.4 DOUBLE BUS FAULTS**), the $\overline{\text{HALT}}$ line is driven by the processor to indicate to external devices that the processor has stopped.

4.1.7 M6800 Peripheral Control

These control signals are used to allow the interfacing of synchronous M6800 peripheral devices with the asynchronous MC68010. These signals are explained in the following paragraphs.

4.1.7.1 ENABLE (E). This signal is the standard enable signal common to all M6800 type peripheral devices. The period for this output is ten MC68010 clock periods (six clocks low, four clocks high). Enable is generated by an internal ring counter which may come up in any state (i.e., at power on, it is impossible to guarantee phase relationship of E to CLK). E is a free-running clock and runs regardless of the state of the bus on the MPU.

4.1.7.2 VALID PERIPHERAL ADDRESS ($\overline{\text{VPA}}$). This input indicates that the device addressed is an M6800 Family device and that data transfer should be synchronized with the enable (E) signal. This input also indicates that the processor should use automatic vectoring for an interrupt. Refer to **SECTION 6 INTERFACE WITH M6800 PERIPHERALS.**

4.1.7.3 VALID MEMORY ADDRESS ($\overline{\text{VMA}}$). This output is used to indicate to M6800 peripheral devices that there is a valid address on the address bus and the processor is synchronized to enable (E). This signal only responds to a valid peripheral address ($\overline{\text{VPA}}$) input which indicates that the peripheral is an M6800 Family device.

4.1.8 Processor Status (FC0, FC1, FC2)

These function code outputs indicate the state (user or supervisor) and the address space currently being accessed, as shown in Table 4-2. The information indicated by the function code outputs is valid whenever address strobe ($\overline{\text{AS}}$) is active.

4.1.9 Clock (CLK)

The clock input is a TTL-compatible signal that is internally buffered for development of the internal clocks needed by the processor. The clock input should not be gated off at any time and the clock signal must conform to minimum and maximum pulse width times.

Table 4-2. Function Code Assignments

Function Code Output			Address Space
FC2	FC1	FC0	
0	0	0	Undefined, Reserved*
0	0	1	User Data Space
0	1	0	User Program Space
0	1	1	Undefined, Reserved*
1	0	0	Undefined, Reserved*
1	0	1	Supervisor Data Space
1	1	0	Supervisor Program Space
1	1	1	CPU Space

*Address space 3 is reserved for user definition, while 0 and 4 are reserved for future use by Motorola.

1-307

4.1.10 Signal Summary

Table 4-3 is a summary of all the signals discussed in the previous paragraphs.

Table 4-3. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Hi-Z	
				On $\overline{HALT}$	On $\overline{BGACK}$
Address Bus	A1-A23 (A24-A29, A31)**	Output	High	Yes	Yes
Data Bus	D0-D15	Input/Output	High	Yes	Yes
Read-Modify Cycle**	$\overline{RMC}$	Output	Low	No	Yes
Address Strobe	$\overline{AS}$	Output	Low	No	Yes
Read/Write	R/$\overline{W}$	Output	Read-High Write-Low	No	Yes
Upper and Lower Data Stobes	$\overline{UDS}$, $\overline{LDS}$	Output	Low	No	Yes
Data Transfer Acknowledge	$\overline{DTACK}$	Input	Low	–	–
Bus Request	$\overline{BR}$	Input	Low	–	–
Bus Grant	$\overline{BG}$	Output	Low	No	No
Bus Grant Acknowledge	$\overline{BGACK}$	Input	Low	–	–
Interrupt Priority Level	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	Input	Low	–	–
Bus Error	$\overline{BERR}$	Input	Low	–	–
Reset	$\overline{RESET}$	Input/Output	Low	No*	No*
Halt	$\overline{HALT}$	Input/Output	Low	No*	No*
Enable	E	Output	High	No	No
Valid Memory Address	$\overline{VMA}$	Output	Low	No	Yes
Valid Peripheral Address	$\overline{VPA}$	Input	Low	–	–
Function Code Output	FC0, FC1, FC2	Output	High	No	Yes
Clock	CLK	Input	High	–	–
Power Input	V_{CC}	Input	–	–	–
Ground	GND	Input	–	–	–

*Open Drain
**MC68012 Only

1-308

4.2 BUS OPERATION

The following paragraphs explain control signal and bus operation during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation.

4.2.1 Data Transfer Operations

Transfer of data between devices involves the following signals:

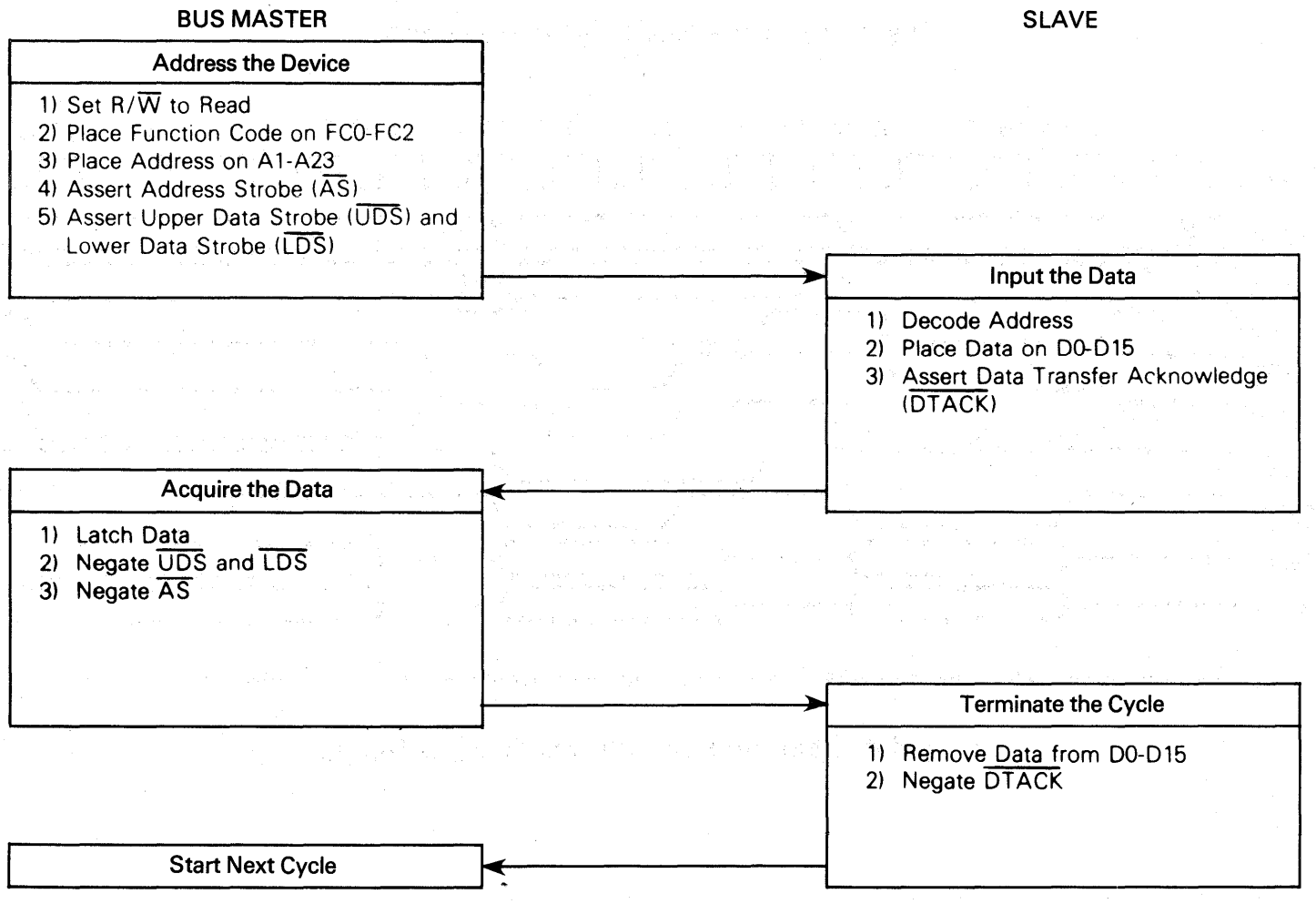
1. address bus A1 through A23,
2. data bus D0 through D15, and
3. control signals.

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cycles, the bus master assumes responsibility for deskewing all signals it issues at both the start and end of a cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave device.

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the MC68010 for interlocked multiprocessor communications. On the MC68012, the $\overline{\text{RMC}}$ pin is asserted to provide a bus lock capability to insure integrity of the read-modify-write cycle.

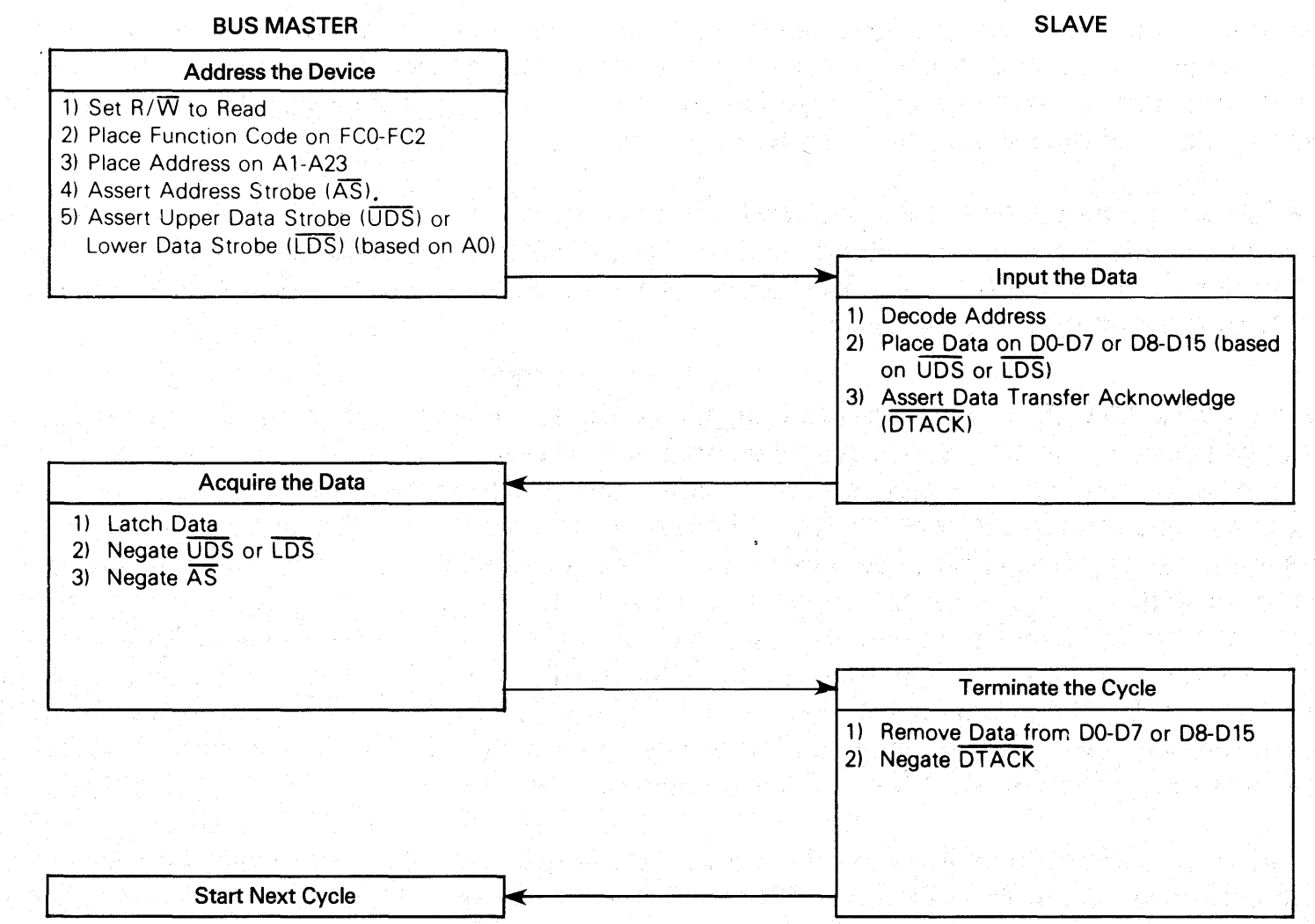
4.2.1.1 READ CYCLE. During a read cycle, the processor receives data from the memory or a peripheral device. The processor reads bytes of data in all cases. If the instruction specifies a word (or long word) operation, the processor reads both upper and lower bytes simultaneously by asserting both upper and lower data strobes. When the instruction specifies byte operation, the processor uses an internal A0 bit to determine which byte to read and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. When the data is received, the processor correctly positions it internally. If $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, or $\overline{\text{VPA}}$ is not asserted for the required setup time before the falling edge of S4, a wait cycle will be inserted in the bus cycle and $\overline{\text{DTACK}}$ will be sampled again on the falling edge of each wait cycle. The MC68010 will continue to insert wait cycles until $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, or $\overline{\text{VPA}}$ is recognized.

A word read cycle flowchart is given in Figure 4-2. A byte read cycle flowchart is given in Figure 4-3. Read cycle timing is given in Figure 4-4. Figure 4-5 details word and byte read cycle operations.



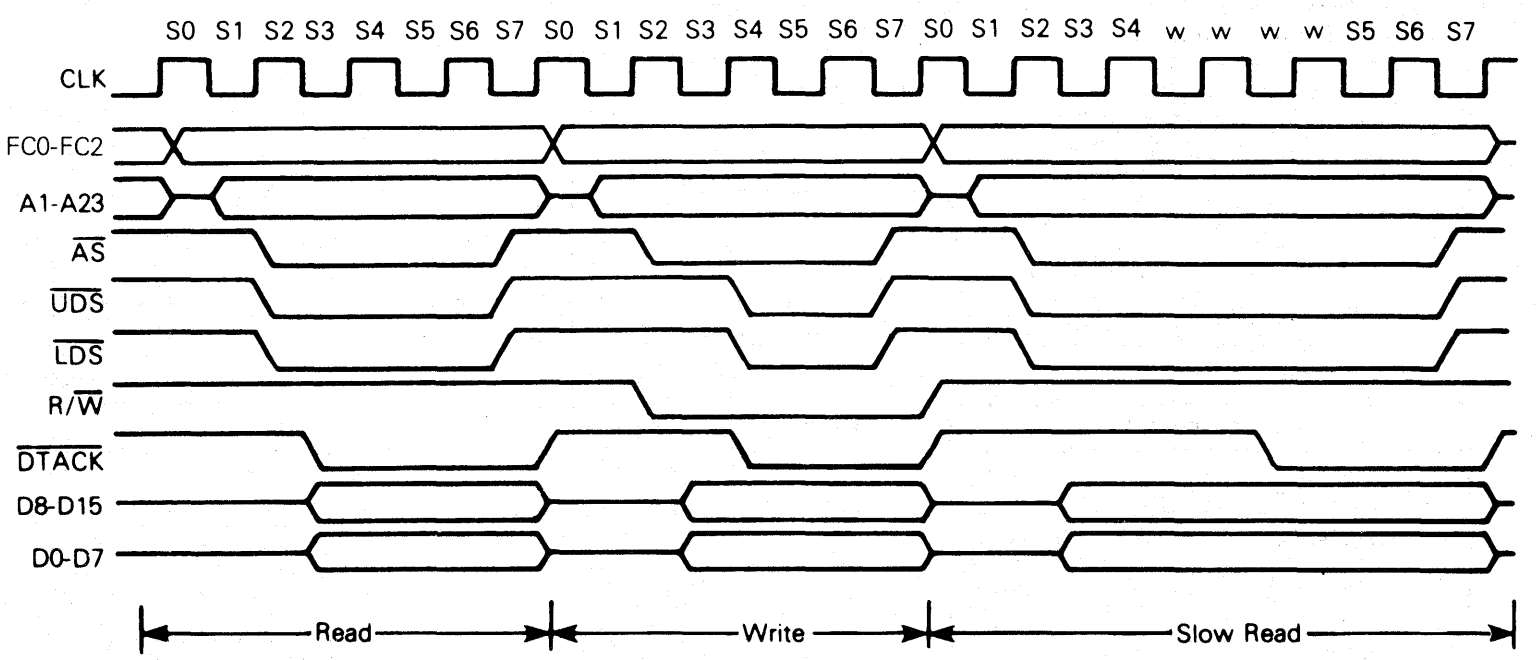
1-309

Figure 4-2. Word Read Cycle Flowchart



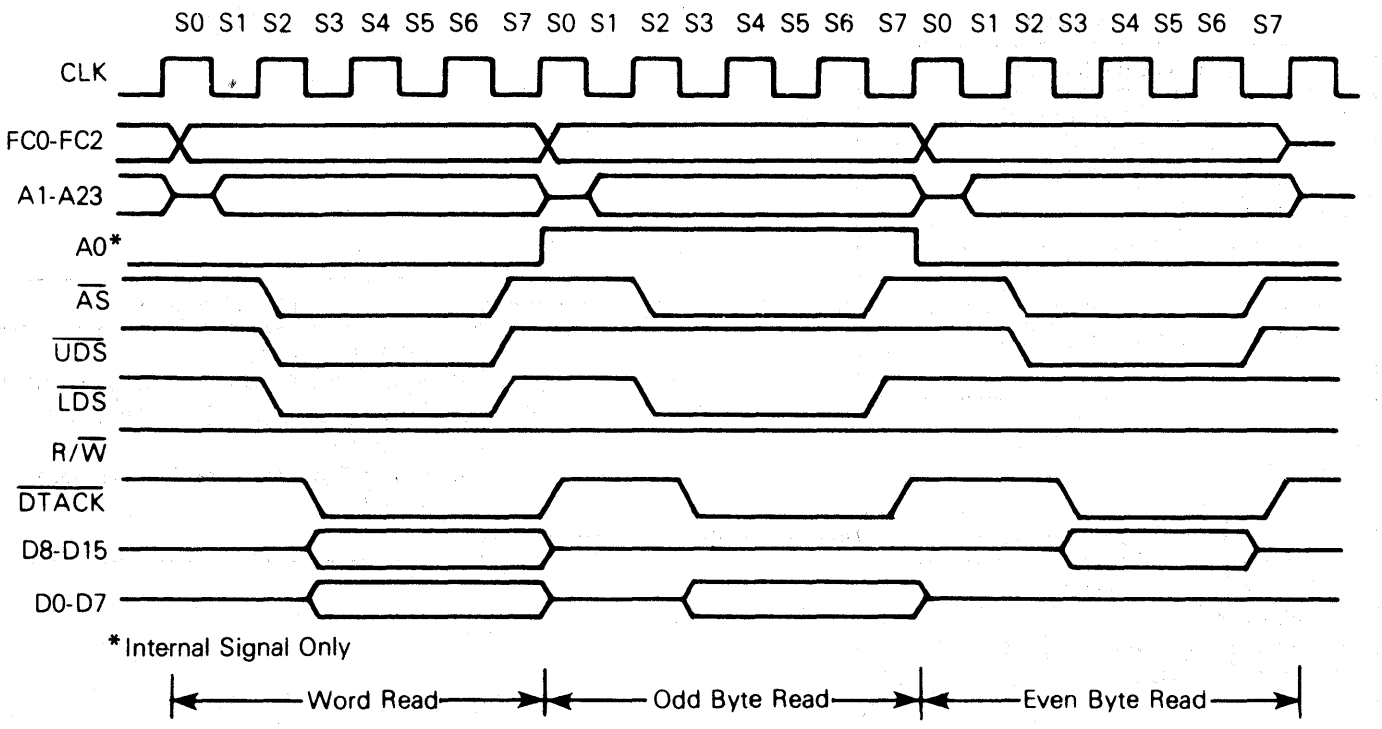
1-310

Figure 4-3. Byte Read Cycle Flowchart



1-311

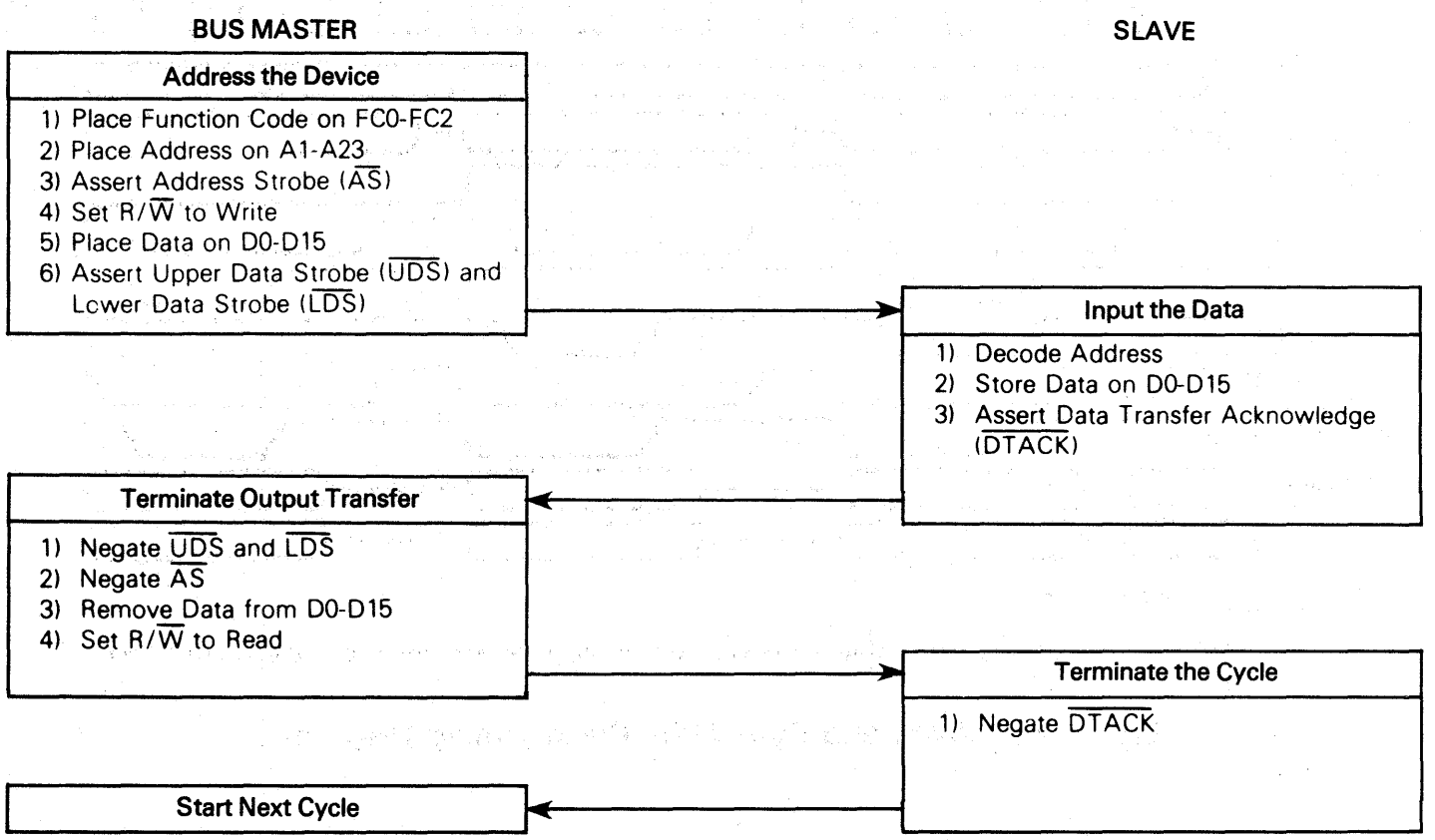
Figure 4-4. Read and Write Cycle Timing Diagram



1-312

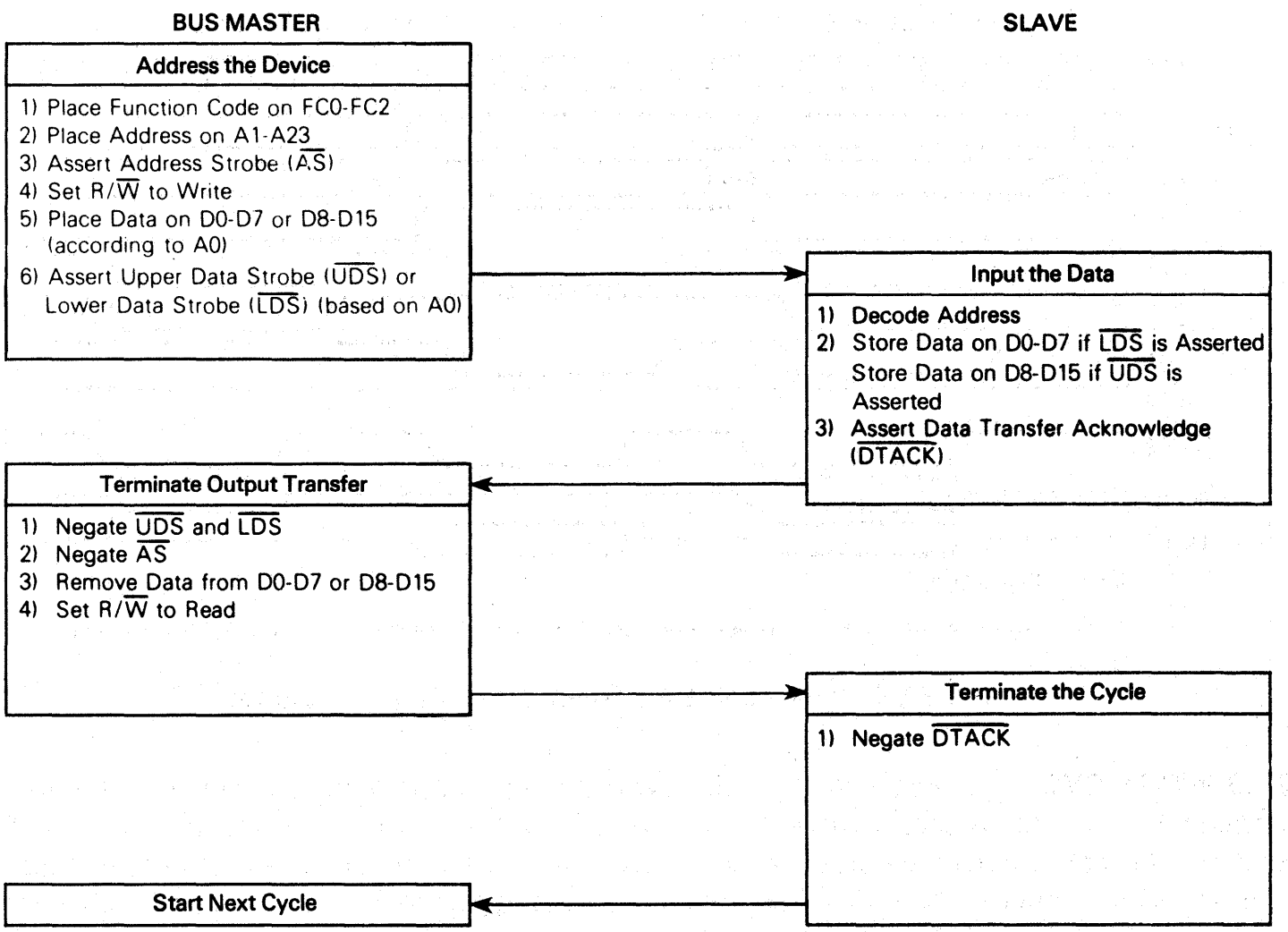
Figure 4-5. Word and Byte Read Cycle Timing Diagram

4.2.1.2 WRITE CYCLE. During a write cycle, the processor sends data to either the memory or a peripheral device. The processor writes bytes of data in all cases. If the instruction specifies a word operation, the processor writes both bytes. When the instruction specifies a byte operation, the processor uses an internal A0 bit to determine which byte to write and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. A word write flowchart is given in Figure 4-6. A byte write cycle flowchart is given in Figure 4-7. Write cycle timing is given in Figure 4-4. Figure 4-8 details word and byte write cycle operation.



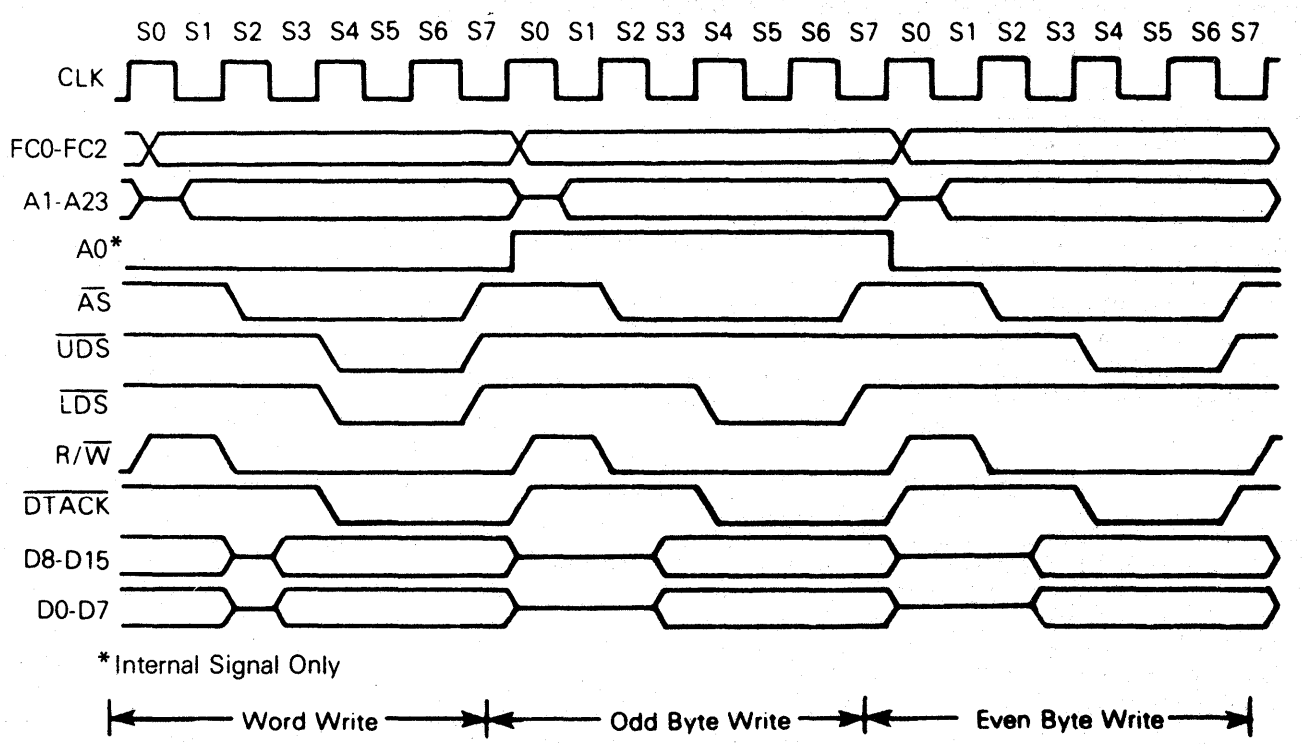
1-313

Figure 4-6. Word Write Cycle Flowchart



1-314

Figure 4-7. Byte Write Cycle Flowchart



1-315

Figure 4-8. Word and Byte Write Cycle Timing Diagram

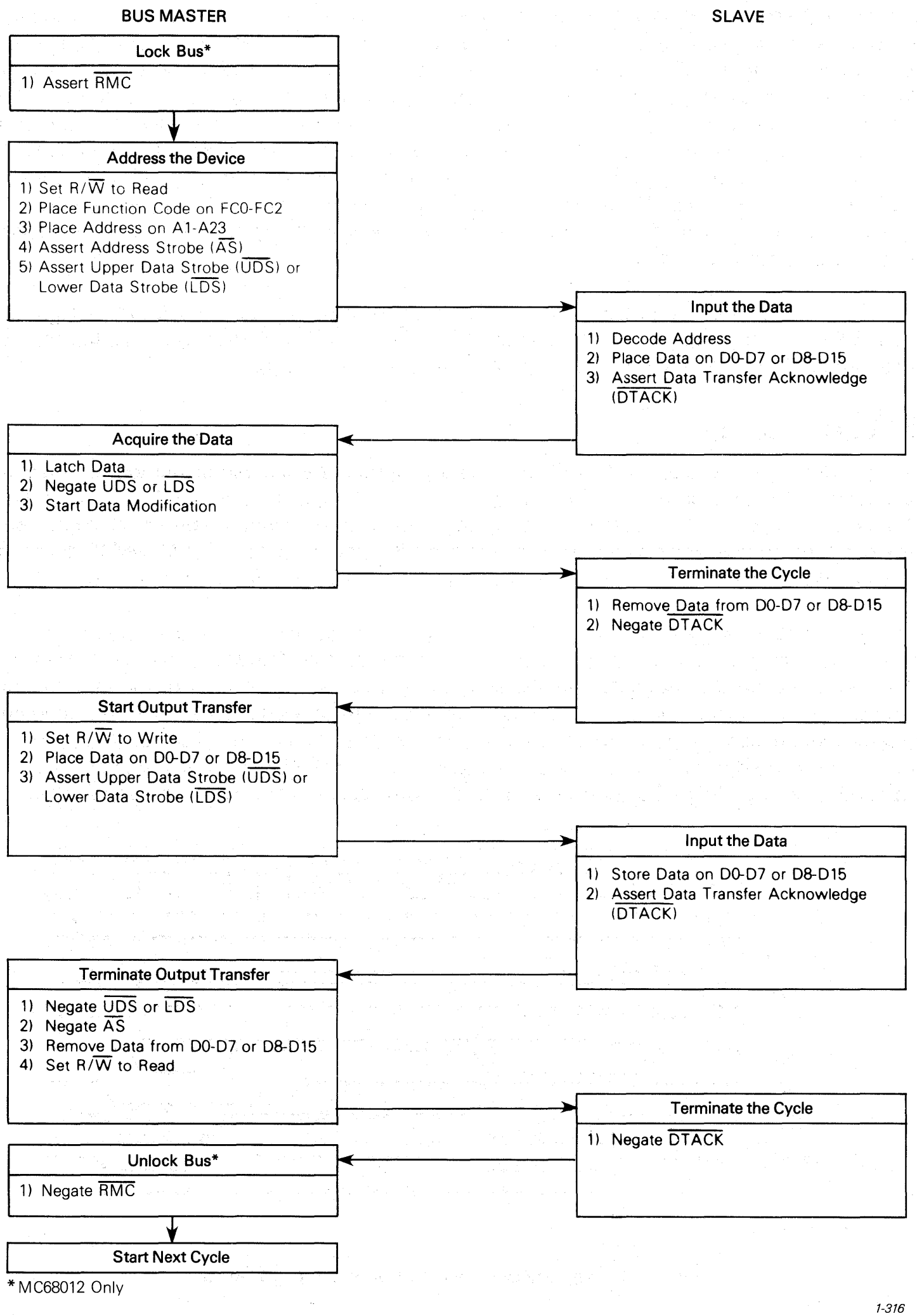


Figure 4-9. Read-Modify-Write Cycle Flowchart

4.2.1.3 READ-MODIFY-WRITE CYCLE. The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the MC68010, this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycle; and, since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. On the MC68012 the $\overline{\text{RMC}}$ pin is asserted, throughout the entire read-modify-write cycle $\overline{\text{RMC}}$ can be used by memory management schemes that require advanced indication of read-modify-write cycles. A read-modify-write flowchart is given in Figure 4-9 and a timing diagram is given in Figure 4-10.

Wait cycles will be inserted between S4 and S5 on the read portion of the bus cycle and between S16 and S17 on the write portion of the cycle if $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, or $\overline{\text{VPA}}$ is not asserted for the required setup time prior to the falling edge of S4 and S16 respectively.

4.2.1.4 CPU Space Cycle

During a CPU space cycle, the MC68010 reads a peripheral device vector number or indicates a breakpoint instruction. If the cycle is to read a vector number it is referred to as an interrupt acknowledge cycle. A CPU space cycle is indicated when the function codes are all high. The address bus then defines what type of CPU space cycle is being executed. The MC68010 defines two types of CPU space cycles, the interrupt acknowledge cycle, and the breakpoint cycle.

The interrupt acknowledge cycle on an M68000 Family compatible processor is defined as a CPU space cycle with the most significant address lines high; on the MC68010 this means that A4-A23 will be high and on the MC68012 A4-A29 and A31 will be high. The level of the interrupt being acknowledged is encoded on address lines A1-A3. An interrupt acknowledge cycle is terminated in the same manner as a normal read cycle. The processor expects a peripheral device to respond to an interrupt acknowledge cycle with a vector number that will be used to transfer control to an interrupt handler routine. See **5.3.2 Interrupts** for further discussion of the interrupt acknowledge cycle.

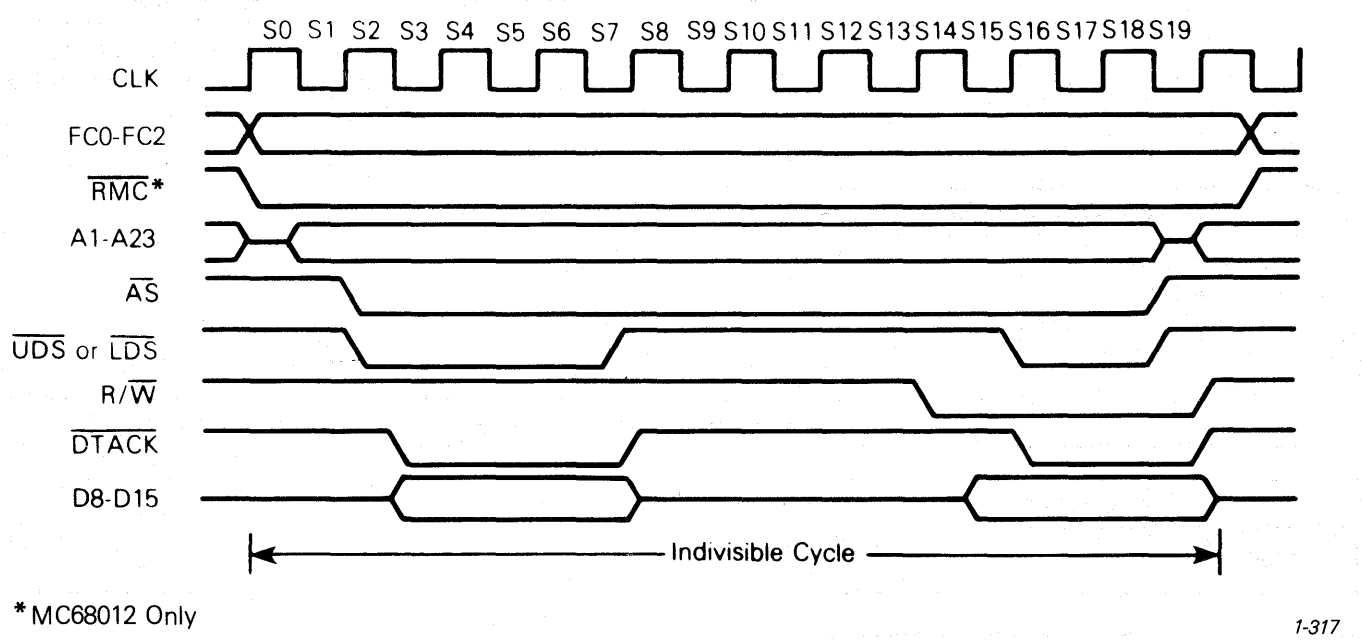
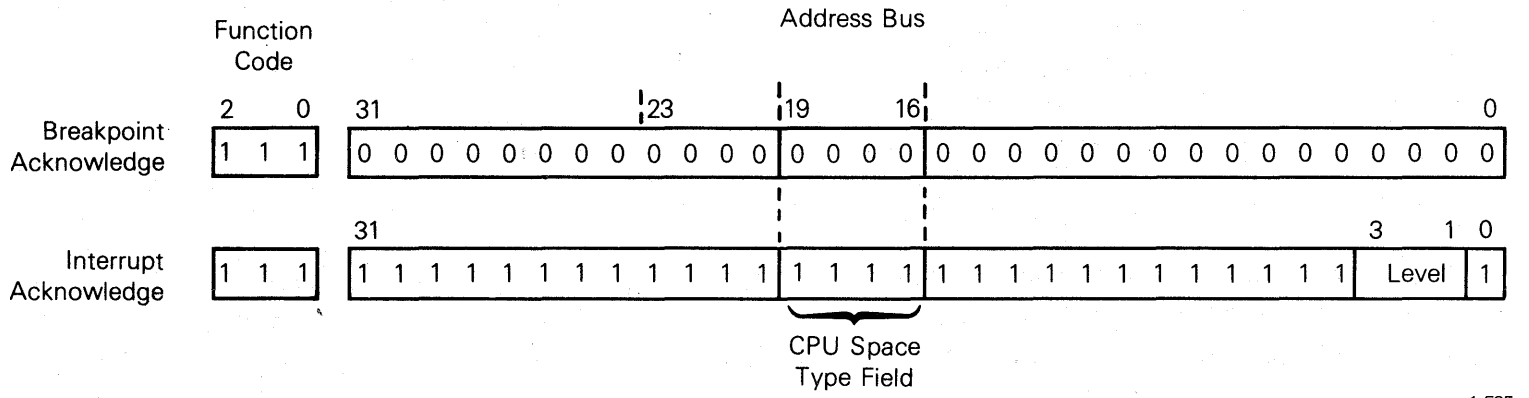


Figure 4-10. Read-Modify-Write Cycle Timing Diagram

The breakpoint read cycle is executed by the MC68010 in response to a breakpoint illegal instruction. A breakpoint cycle on the MC68010 is defined as a CPU space cycle with all of the address lines low. The processor does not accept or send any data during this cycle. The breakpoint cycle may be terminated by $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, or $\overline{\text{VPA}}$. See **5.3.6 Illegal and Unimplemented Instructions** for further discussion of breakpoints.

Since all members of the M68000 Family do not implement A20-A31, these lines do not need to be decoded for CPU space functions. Only A16-A19 are used to distinguish between different CPU space cycle types. The MC68010 only uses the $0 and $F CPU space types as shown in Figure 4-11; however, all unused encodings of bits A16-A19 are reserved by Motorola for future extensions of the CPU space functions.



1-585

Figure 4-11. MC68010 CPU-Space Address Encoding

4.2.2 Bus Arbitration

Bus arbitration is a technique used by master-type devices to request, be granted, and acknowledge bus mastership. In its simplest form, it consists of the following:

1. asserting a bus mastership request,
2. receiving a grant that the bus is available at the end of the current cycle, and
3. acknowledging that mastership has been assumed.

Figure 4-12 is a flowchart showing the detail involved in a request from a single device. Figure 4-13 is a timing diagram for the same operation. This technique allows processing of bus requests during data transfer cycles.

The timing diagram shows that the bus request is negated at the time that an acknowledge is asserted. This type of operation would be true for a system consisting of the processor and one device capable of bus mastership. In systems having a number of devices capable of bus mastership, the bus request line from each device is wire ORed to the processor. In this system, it is easy to see that there could be more than one bus request being made. The timing diagram shows that the bus grant signal is negated a few clock cycles after the transition of the acknowledge ($\overline{\text{BGACK}}$) signal.

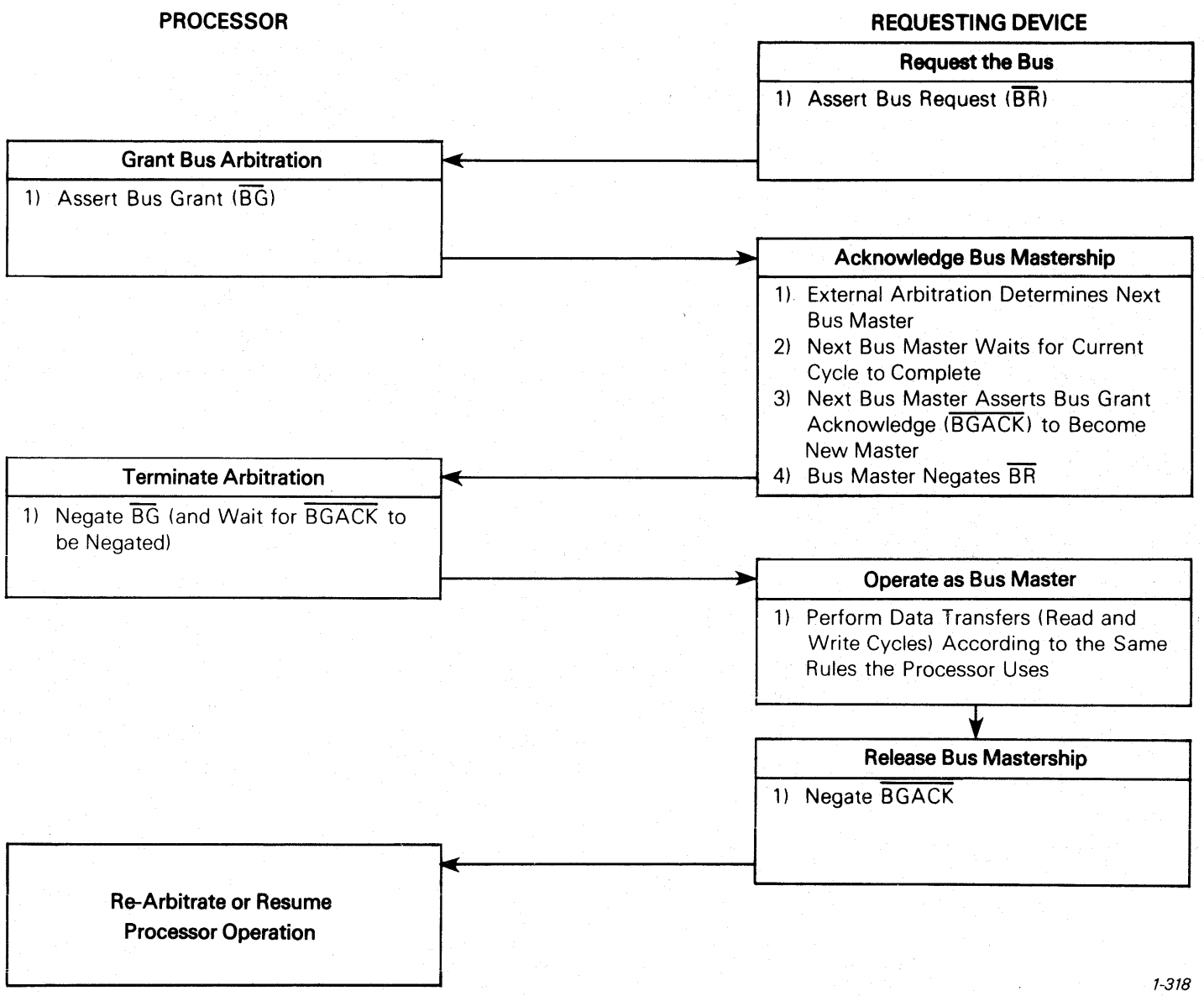


Figure 4-12. Bus Arbitration Cycle Flowchart

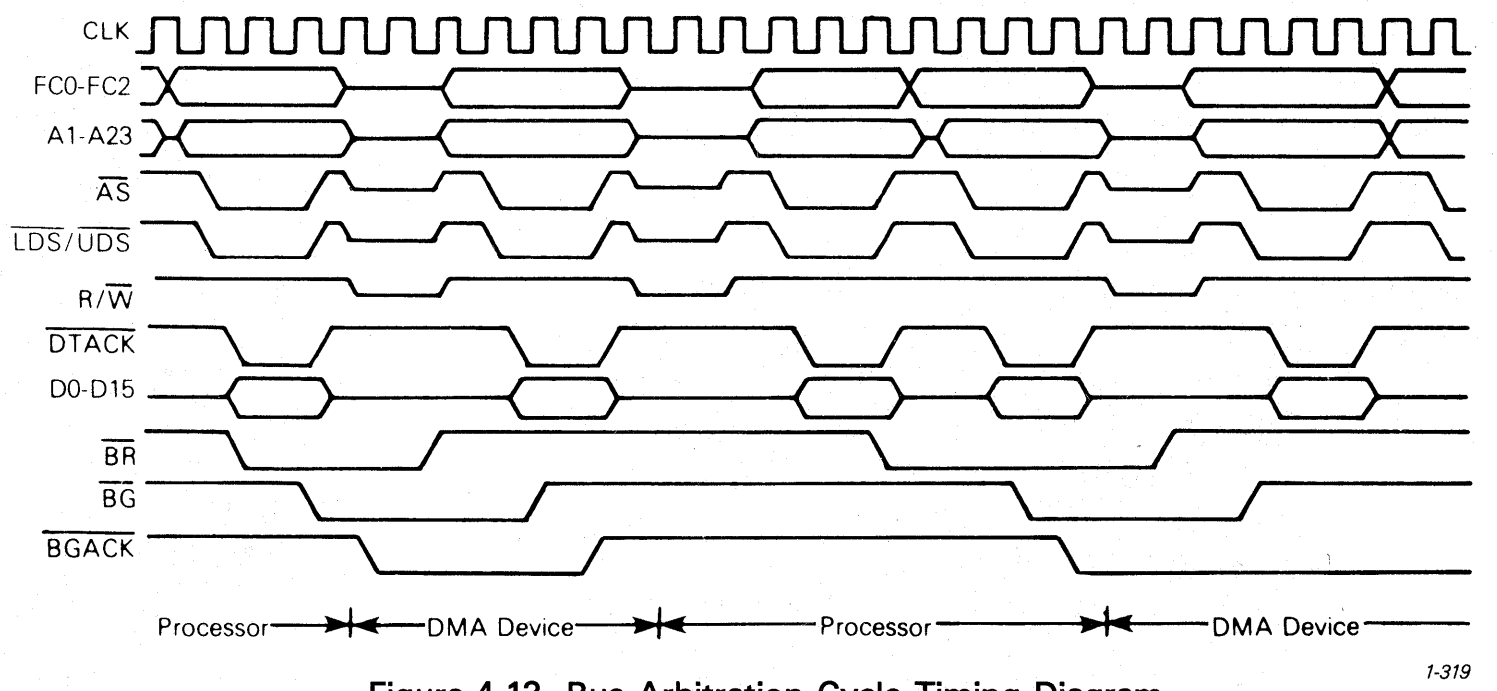


Figure 4-13. Bus Arbitration Cycle Timing Diagram

However, if bus requests are still pending, the processor will assert another bus grant within a few clock cycles after it was negated. This additional assertion of bus grant allows external arbitration circuitry to select the next bus master before the current bus master has completed its requirements. The following paragraphs provide additional information about the three steps in the arbitration process.

4.2.2.1 REQUESTING THE BUS. External devices capable of becoming bus masters request the bus by asserting the bus request ($\overline{BR}$) signal. This is a wire-ORed signal (although it need not be constructed from open-collector devices) that indicates to the processor that some external device requires control of the external bus. The processor is effectively at a lower bus priority level than the external device and will relinquish the bus after it has completed the last bus cycle it has started.

When no acknowledge is received before the bus request signal goes inactive, the processor will continue processing when it detects that the bus request is inactive. This allows ordinary processing to continue if the arbitration circuitry responded to noise inadvertently.

4.2.2.2 RECEIVING THE BUS GRANT. The processor asserts bus grant ($\overline{BG}$) as soon as possible. Normally this is immediately after internal synchronization. The only exception to this occurs when the processor has made an internal decision to execute the next bus cycle but has not progressed far enough into the cycle to have asserted the address strobe ($\overline{AS}$) signal. In this case, bus grant will be delayed until $\overline{AS}$ is asserted to indicate to external devices that a bus cycle is being executed.

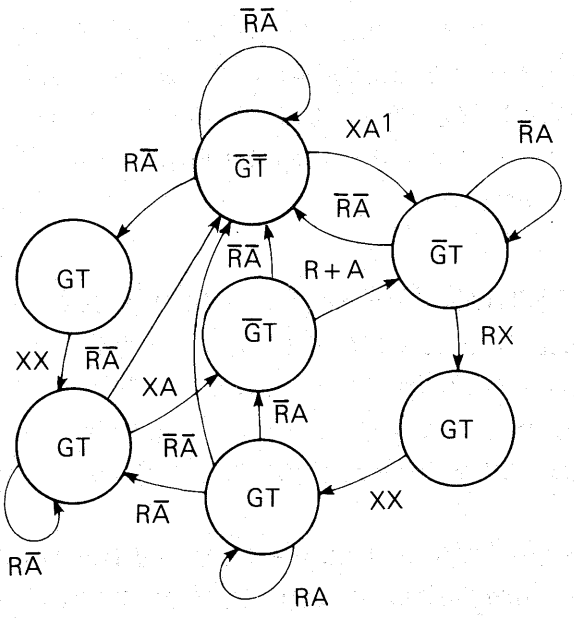
The bus grant signal may be routed through a daisy-chained network or through a specific priority-encoded network. The processor is not affected by the external method of arbitration as long as the protocol is obeyed.

4.2.2.3 ACKNOWLEDGEMENT OF MASTERSHIP. Upon receiving a bus grant, the requesting device waits until address strobe, data transfer acknowledge, and bus grant acknowledge are negated before issuing its own $\overline{BGACK}$. The negation of the $\overline{AS}$ indicates that the previous master has completed its cycle; the negation of $\overline{BGACK}$ indicates that the previous master has released the bus. (While address strobe is asserted, no device is allowed to "break into" a cycle.) The negation of $\overline{DTACK}$ indicates the previous slave has terminated its connection to the previous master. Note that in some applications data transfer acknowledge might not enter into this function. General purpose devices would then be connected such that they were only dependent on address strobe. When bus grant acknowledge is issued, the device is a bus master until it negates bus grant acknowledge. Bus grant acknowledge should not be negated until after the bus cycle(s) is (are) completed. Bus mastership is terminated at the negation of bus grant acknowledge.

The bus request from the granted device should be negated after bus grant acknowledge is asserted. If a bus request is still pending, another bus grant will be asserted within a few clocks of the negation of the bus grant. Refer to **4.2.3 Bus Arbitration Control.** Note that the processor does not perform any external bus cycles before it re-asserts bus grant.

4.2.3 Bus Arbitration Control

The bus arbitration control unit in the MC68010 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 4-14. All asynchronous signals to the MC68010 are synchronized before they are used internally. This synchronization is accomplished in a maximum of



R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three-State Control to Bus Control Logic[2]
X = Don't Care

NOTES:
1. State machine will not change if bus is in S0 or S1. Refer to **4.2.3 Bus Arbitration Control**.
2. The address bus will be placed in the high-impedance state if T is asserted and $\overline{AS}$ is negated.

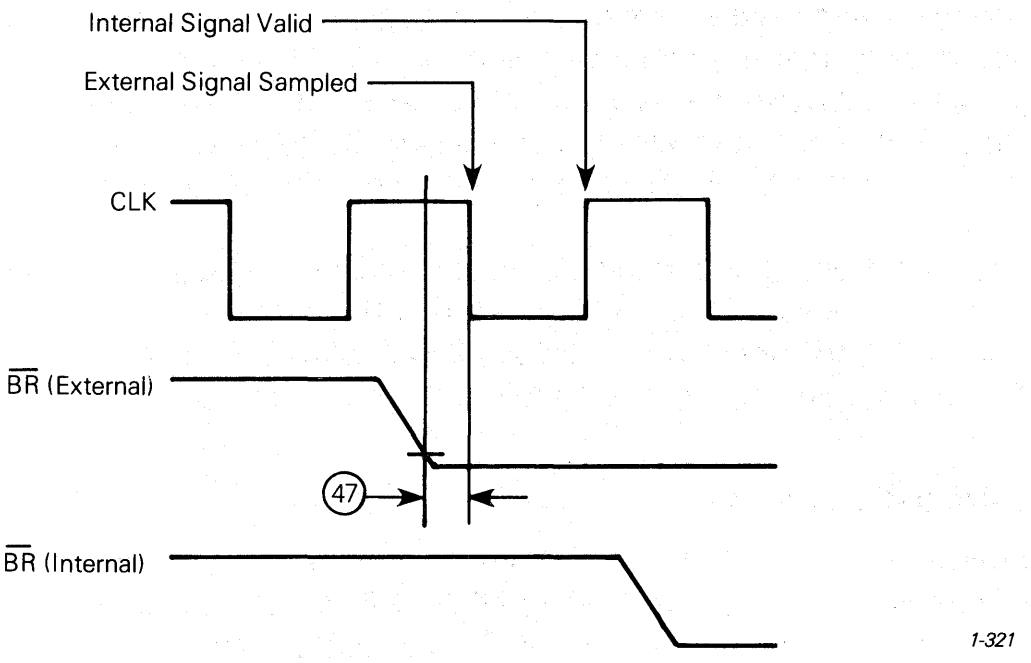
1-320

Figure 4-14. MC68010 Bus Arbitration Unit State Diagram

one cycle of the system clock, assuming that the asynchronous input setup time (#47) has been met (see Figure 4-15). The input signal is sampled on the falling edge of the clock and is valid internally after the next rising edge.

As shown in Figure 4-14, input signals labeled R and A are internally synchronized on the bus request and bus grant acknowledge pins respectively. The bus grant output is labeled G and the internal three-state control signal is labeled T. If T is true, the address, data, and control buses are placed in a high-impedance state when $\overline{AS}$ is negated. All signals are shown in positive logic (active high) regardless of their true active voltage level.

State changes (valid outputs) occur on the next rising edge after the internal signal is valid.



1-321

Figure 4-15. Timing Relationship of External Asynchronous Inputs to Internal Signals

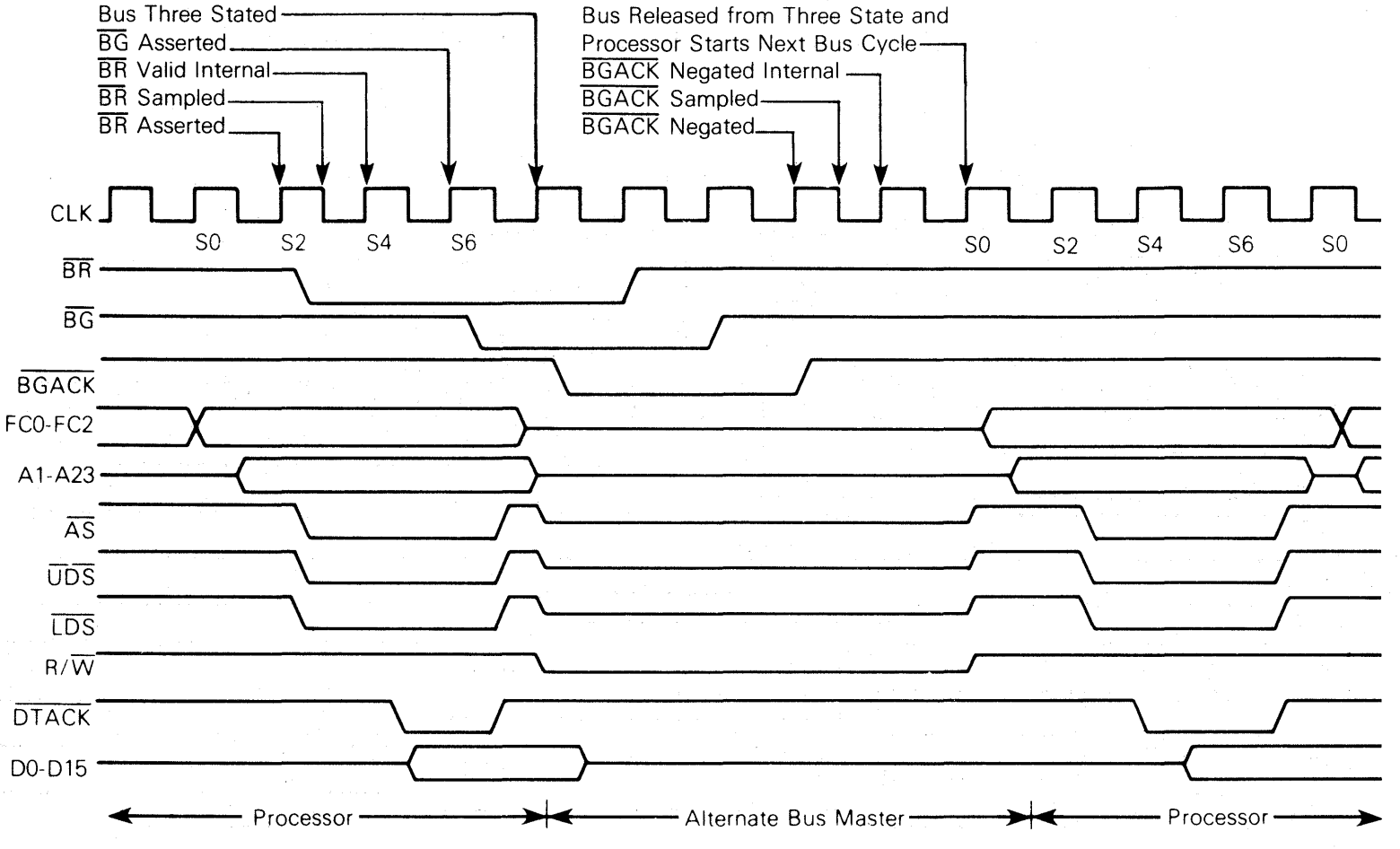
A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 4-16. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 4-17.

If a bus request is made at a time when the MPU has already begun a bus cycle but $\overline{AS}$ has not been asserted (bus state S0), $\overline{BG}$ will not be asserted on the next rising edge. Instead, $\overline{BG}$ will be delayed until the second rising edge following its internal assertion. This sequence is shown in Figure 4-18.

4.2.4 Bus Error and Halt Operation

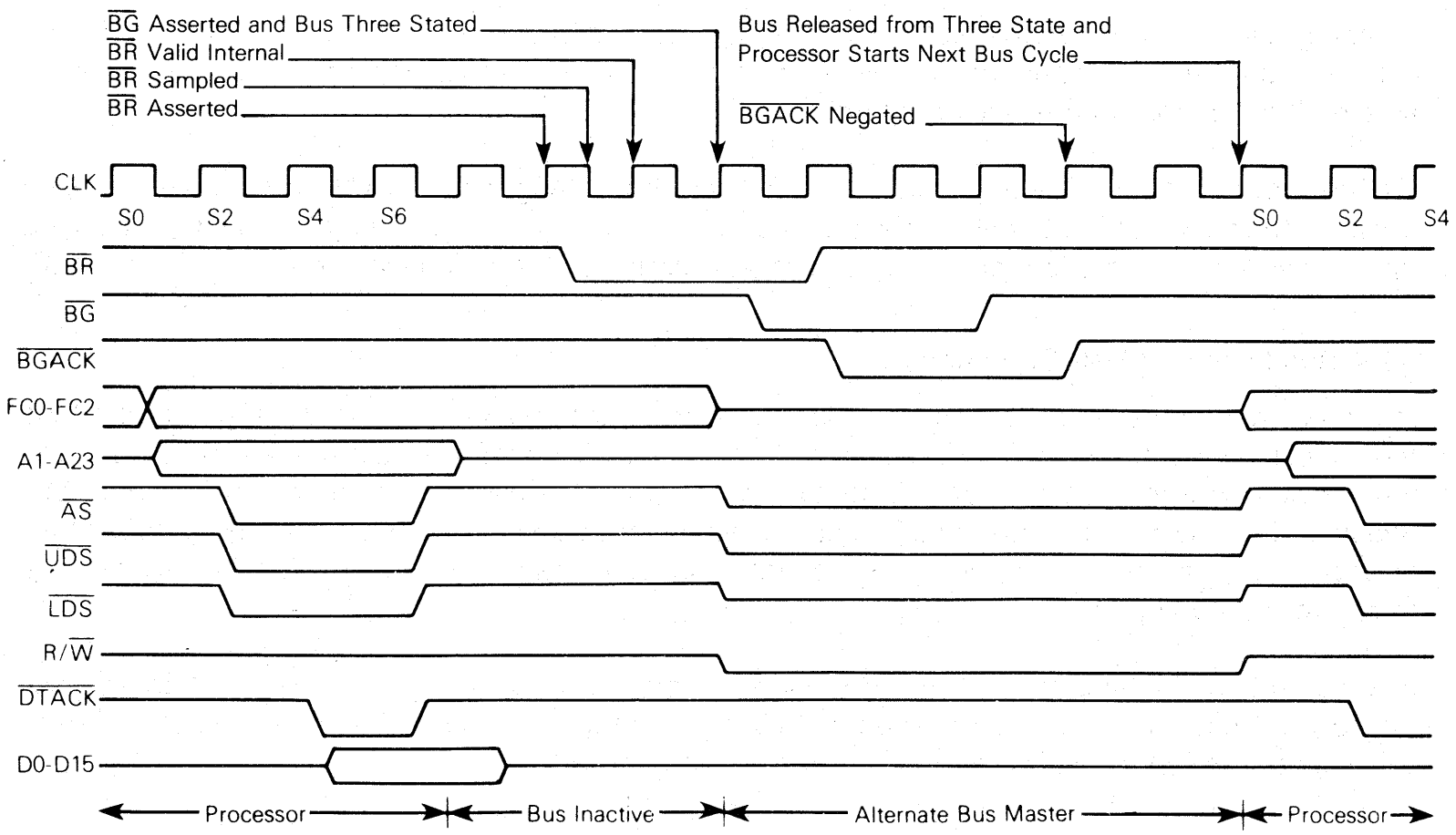
In a bus architecture that requires a handshake from an external device, the possibility exists that the handshake might not occur. Since different systems will require a different maximum response time, a bus error input is provided. External circuitry must be used to determine the duration between address strobe and data transfer acknowledge before issuing a bus error signal. When a bus error or/and halt signal is received, the processor will initiate a bus error exception sequence or try to re-run the bus cycle.

In addition to a bus timeout indicator, the bus error input is used to indicate a page fault in a virtual memory system. When an external memory management unit detects an invalid access, a bus error is signaled to suspend execution of the current instruction.



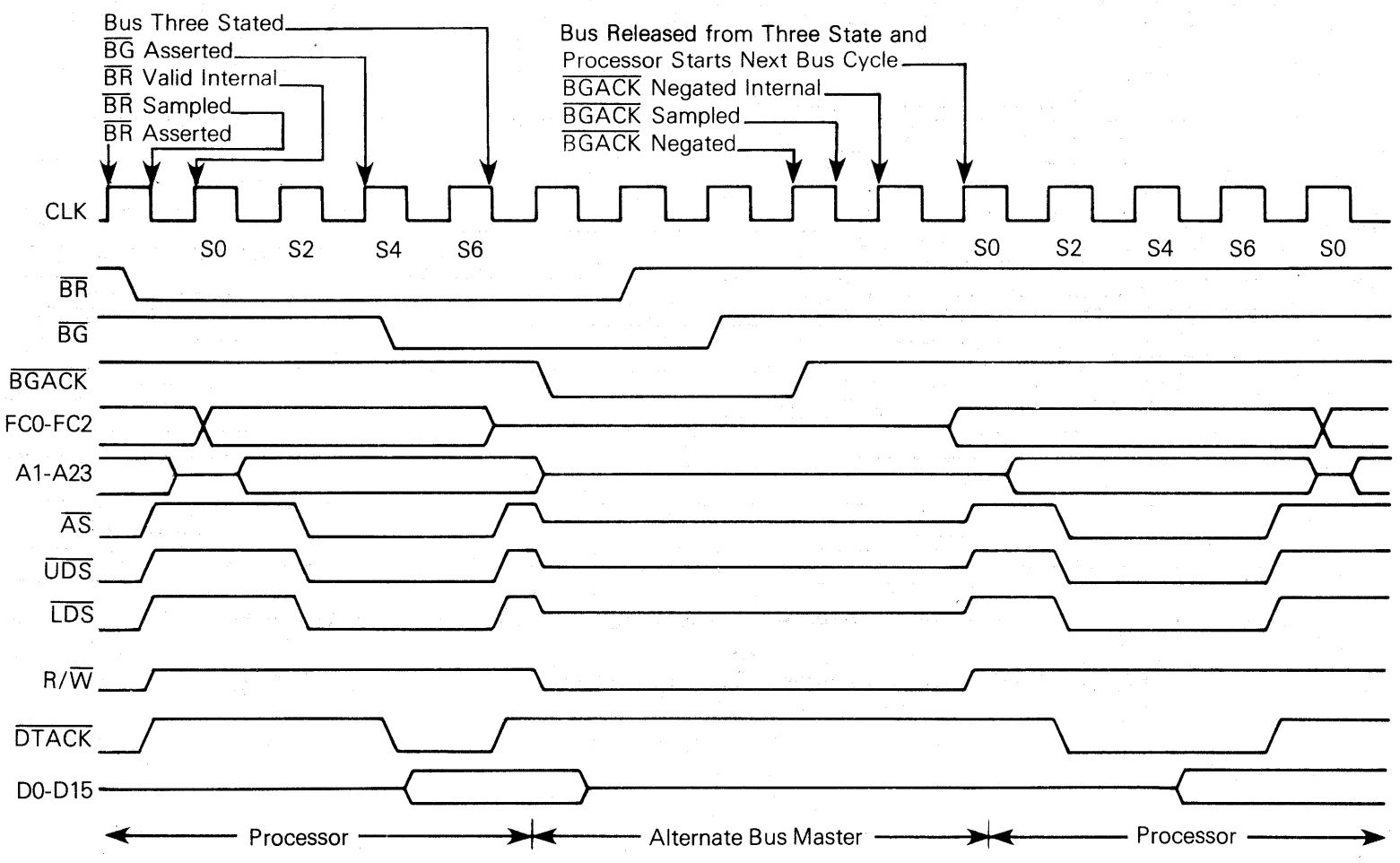
1-322

Figure 4-16. Bus Arbitration Timing Diagram — Processor Active



1-323

Figure 4-17. Bus Arbitration Timing Diagram — Bus Inactive



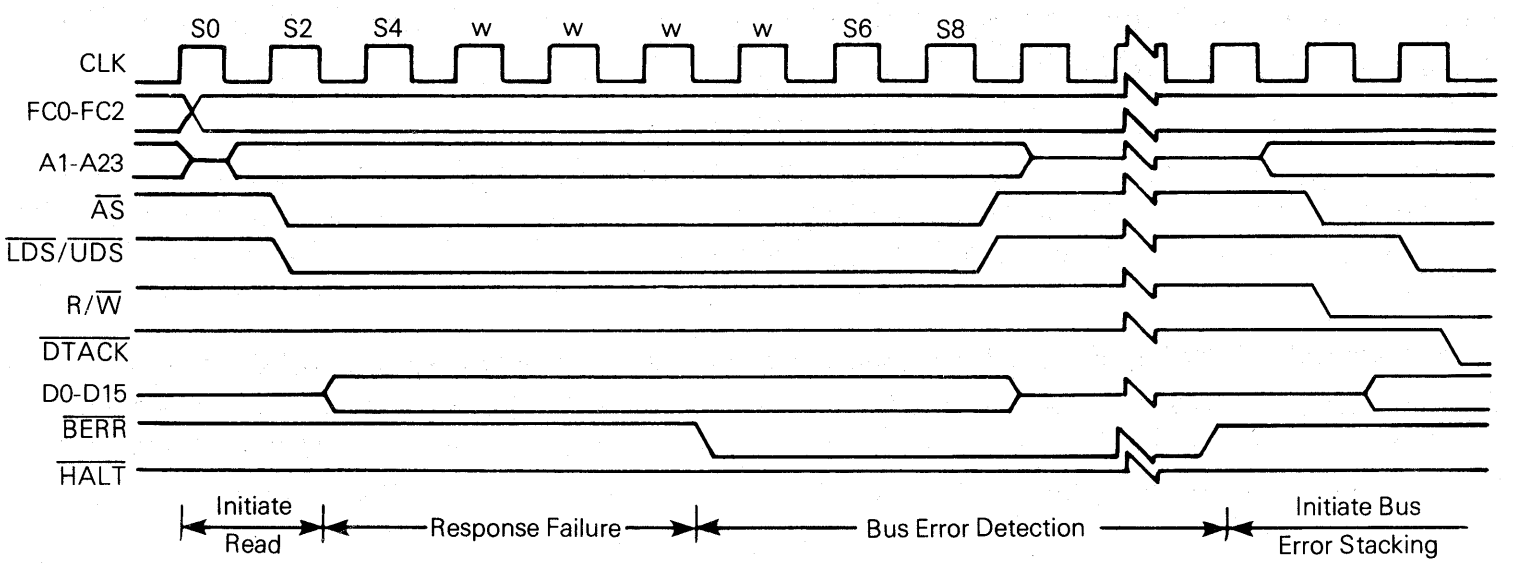
1-324

Figure 4-18. Bus Arbitration Timing Diagram — Special Case

4.2.4.1 BUS ERROR OPERATION. When the bus error signal is used to terminate a bus cycle, the MC68010 will enter exception processing immediately following the bus cycle. The bus error signal is recognized in either of the following cases:

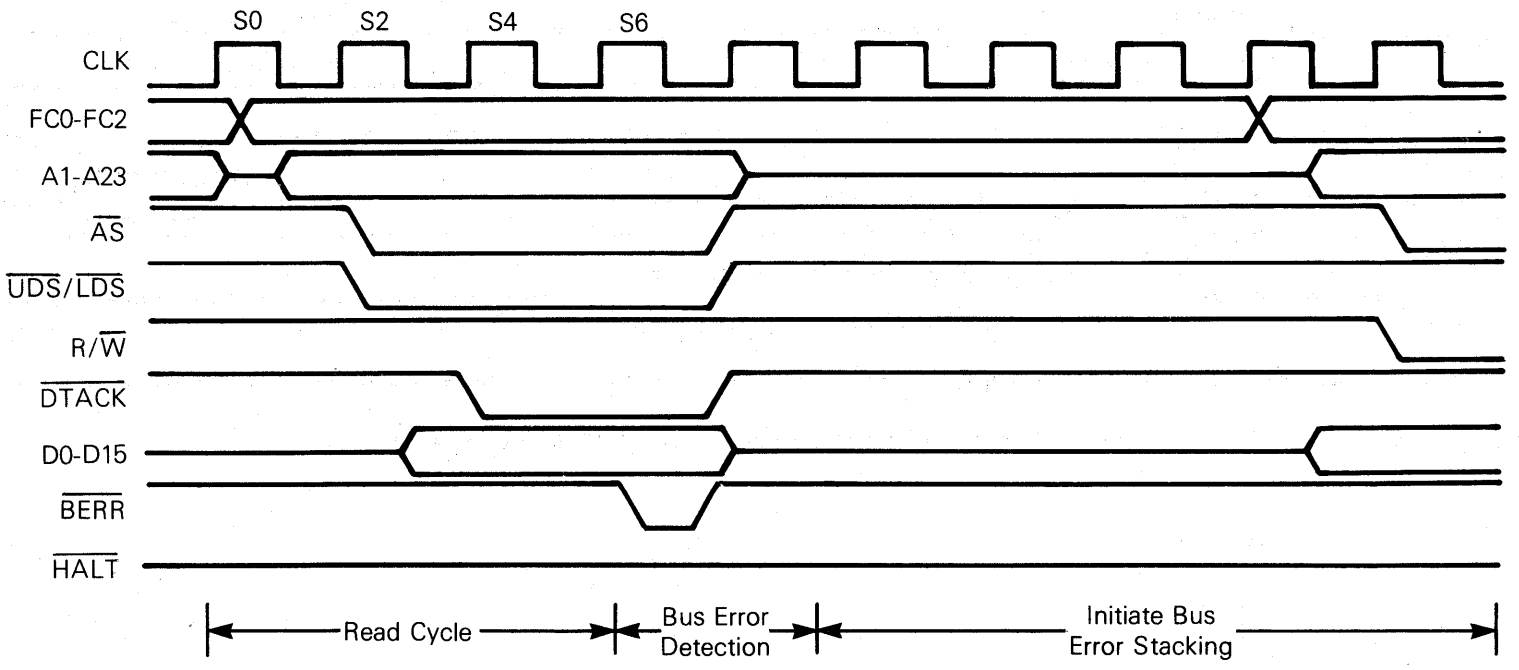
1. $\overline{\text{DTACK}}$ and $\overline{\text{HALT}}$ are negated and $\overline{\text{BERR}}$ is asserted.
2. $\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ are negated and $\overline{\text{DTACK}}$ is asserted. $\overline{\text{BERR}}$ is then asserted within one clock cycle.

When the bus error condition is recognized, the current bus cycle will be terminated in S9 for a read cycle, a write cycle, or the read portion of a read-modify-write cycle and in S21 of the write portion of a read-modify-write cycle. As long as $\overline{\text{BERR}}$ remains asserted, the data and address buses will be in the high-impedance state. Figures 4-19 and 4-20 show the timing diagrams for both types of bus error signals.



1-325

Figure 4-19. Bus Error Timing Diagram



1-326

Figure 4-20. Delayed Bus Error Timing Diagram

After the aborted bus cycle is terminated and $\overline{BERR}$ is negated, the MC68010 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, may be up to five words past the instruction being executed)
3. Frame format and vector offset
4. Internal register information, 22 words

Note that the first four words of information are identical to the information stacked by any other exception such as an interrupt or TRAP instruction. The additional information is used by the MC68010 to continue the execution of the suspended instruction when it is reloaded by an RTE instruction. See **5.3.9 Bus Error** for further details.

After the MC68010 has placed the above information on the stack, the bus error exception vector is read from vector table entry number two (offset $08) and placed in the program counter. The processor then resumes instruction execution.

NOTE

If a read-modify-write instruction is terminated with a bus error and later continued with an RTE instruction, the processor will re-run the entire cycle whether the bus error occurred on the read or the write portion of the cycle.

4.2.4.2 RE-RUN OPERATION. When, during a bus cycle, the processor receives a bus error signal and the halt pin is being driven by an external device, the processor enters the re-run sequence. A delayed re-run signal may be used similarly to the delayed bus error signal described above. Figures 4-21 and 4-22 are timing diagrams for both methods of re-running the bus cycle.

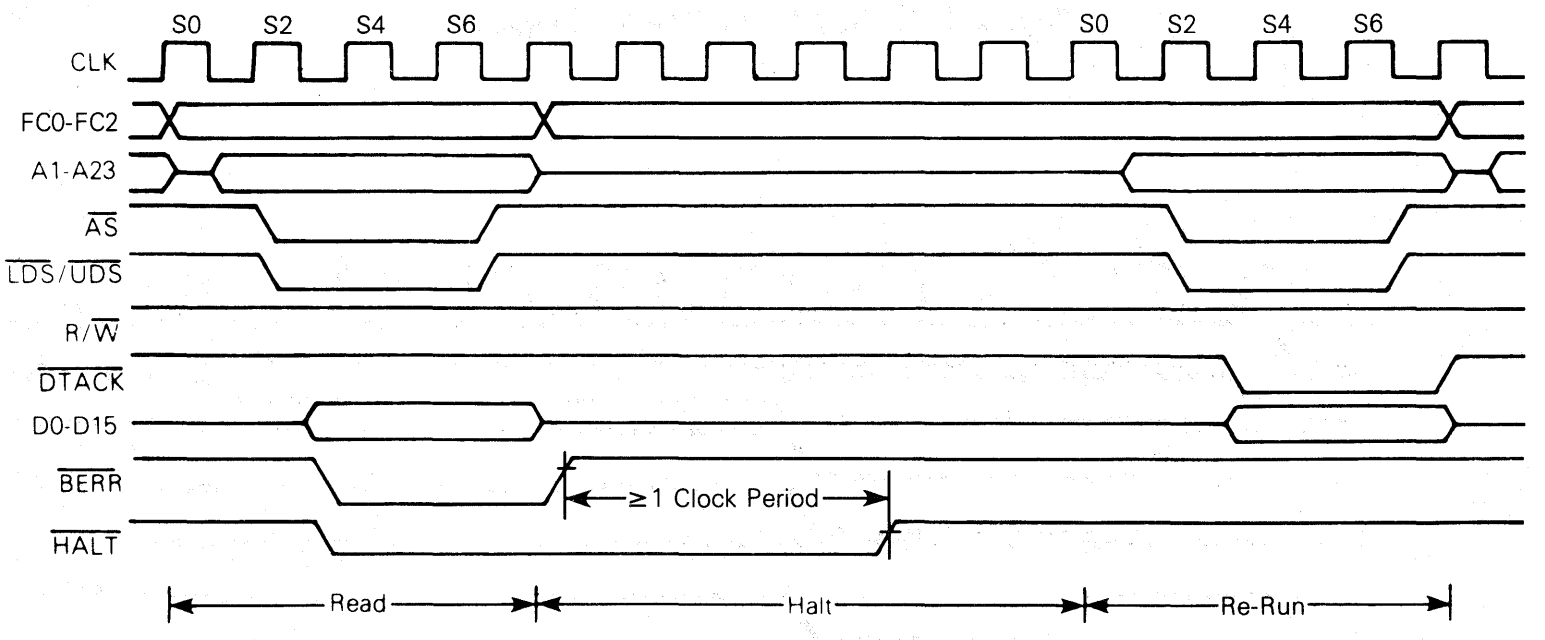


Figure 4-21. Re-Run Bus Cycle Timing Diagram

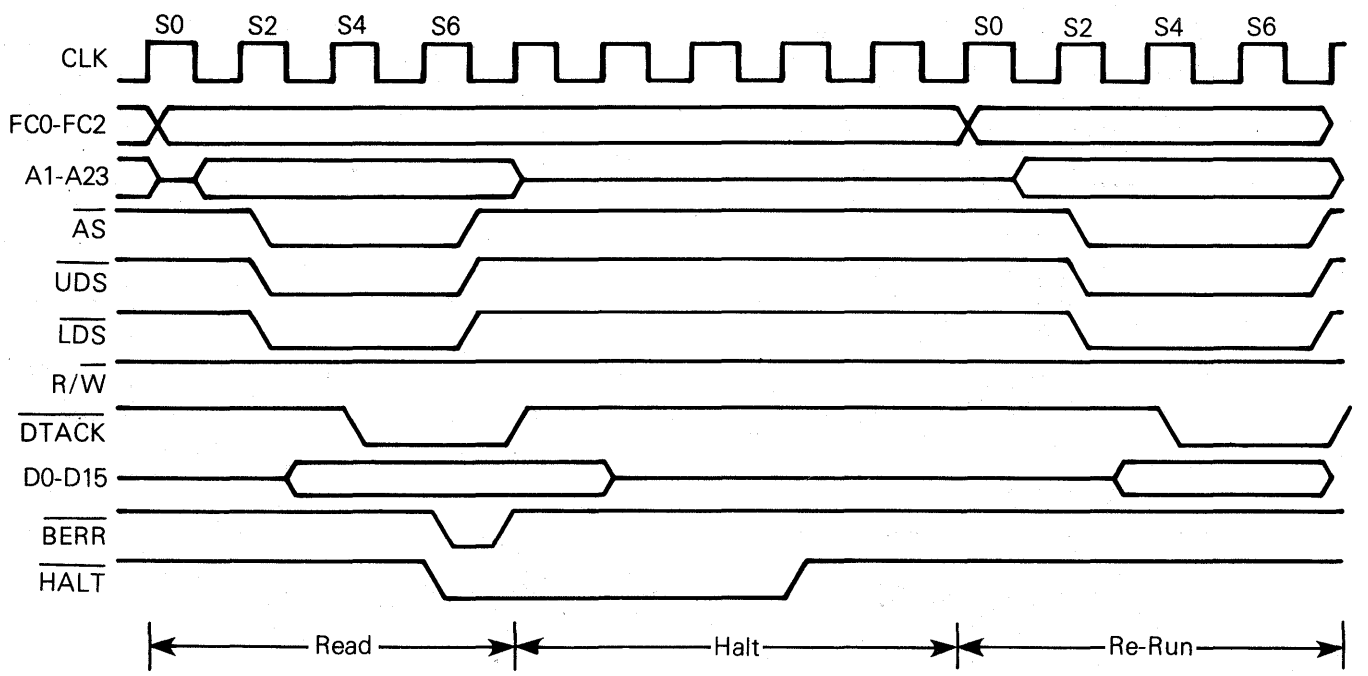


Figure 4-22. Delayed Re-Run Bus Cycle Timing Diagram

The processor terminates the bus cycle, then puts the address and data lines in the high-impedence state. The processor remains "halted", and will not run another bus cycle until the halt signal is removed by external logic. Then the processor will re-run the previous cycle using the same function codes, the same data (for a write operation), and the same address. The bus error signal should be removed at least one clock cycle before the halt signal is removed.

NOTE

The processor will not re-run a read-modify-write cycle. This restriction is made to guarantee that the entire cycle runs correctly and that the write operation of a test-and-set operation is performed without ever releasing $\overline{AS}$. If $\overline{BERR}$ and $\overline{HALT}$ are asserted during a read-modify-write bus cycle, a bus error operation results.

4.2.4.3 HALT OPERATION. The halt input signal to the MC68010 performs a halt/run/single-step function in a similar fashion to the M6800 halt function. The halt and run modes are somewhat self explanatory in that when the halt signal is constantly active the processor "halts" (does nothing) and when the halt signal is constantly inactive the processor "runs" (does something).

This single-step mode is derived from correctly timed transitions on the halt signal input. It forces the processor to execute a single bus cycle by entering the run mode until the processor starts a bus cycle then changing to the halt mode. Thus, the single-step mode allows the user to proceed through (and therefore debug) processor operations one bus cycle at a time.

Figure 4-23 details the timing required for correct single-step operations. Some care must be exercised to avoid harmful interactions between the bus error signal and the halt pin when using the single-cycle mode as a debugging tool. This is also true of interactions between the halt and reset lines since these can reset the machine.

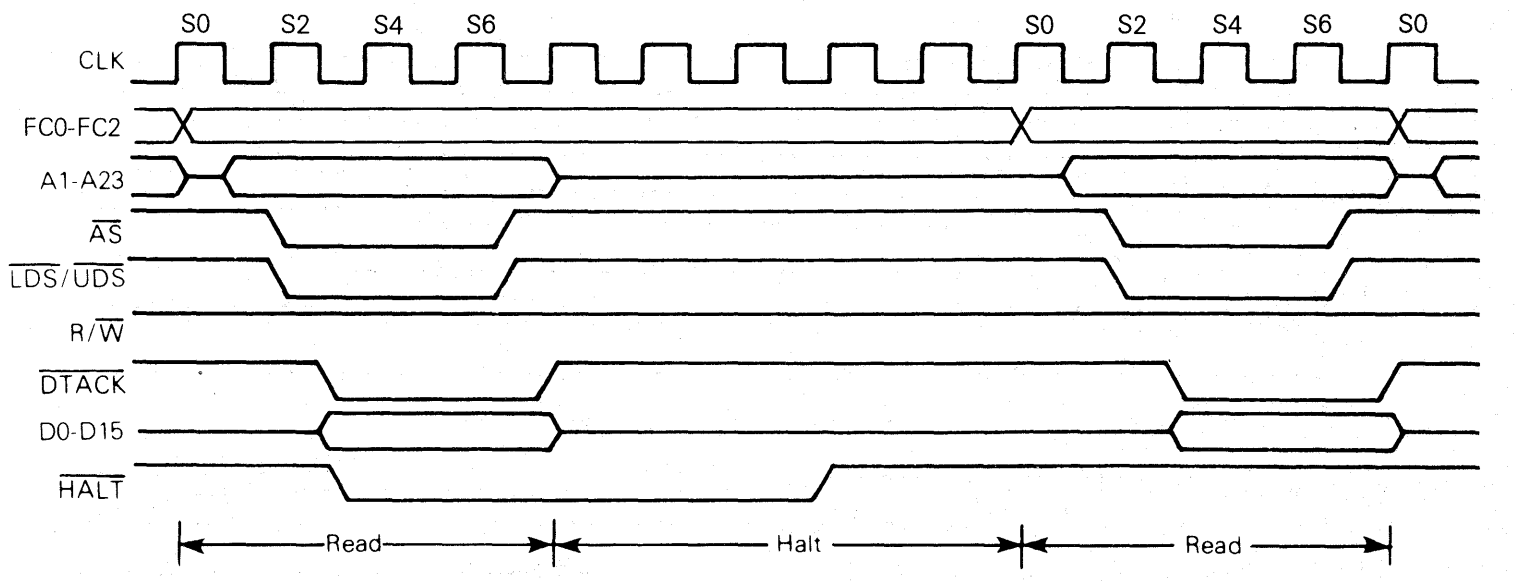


Figure 4-23. Halt Processor Timing Diagram

When the processor completes a bus cycle after recognizing that the halt signal is active, most three-state signals are put in the high-impedence state, these include:

1. address lines, and
2. data lines.

This is required for correct performance of the re-run bus cycle operation.

While the processor is honoring the halt request, bus arbitration performs as usual. That is, halting has no effect on bus arbitration. It is the bus arbitration function that removes the control signals from the bus.

The halt function and the hardware trace capability allow the hardware debugger to trace single bus cycles or single instructions at a time. These processor capabilities, along with a software debugging package, give total debugging flexibility.

4.2.4.4 DOUBLE BUS FAULTS. When a bus error exception occurs, the processor will attempt to stack several words containing information about the state of the machine. If a bus error exception occurs during the stacking operation, there have been two bus errors in a row. This is commonly referred to as a double bus fault. When a double bus fault occurs, the processor will halt and drive the $\overline{\text{HALT}}$ line low. Once a bus error exception has occurred, any bus error exception occurring before the execution of the next instruction constitutes a double bus fault.

Note that a bus cycle which is re-run does not constitute a bus error exception and does not contribute to a double bus fault. Note also that this means that as long as the external hardware requests it, the processor will continue to re-run the same bus cycle.

The bus error pin also has an effect on processor operation after the processor receives an external reset input. The processor reads the vector table after a reset to determine the address to start program execution. If a bus error occurs while reading the vector table (or at any time before the first instruction is executed), the processor reacts as if a double bus fault has occurred and it halts. Only an external reset will start a halted processor.

4.2.5 Reset Operation

The reset signal is a bidirectional signal that allows either the processor or an external device to reset the system. Figure 4-24 is a timing diagram for the reset operation. Both the halt and reset lines must be asserted to ensure total reset of the processor in all cases.

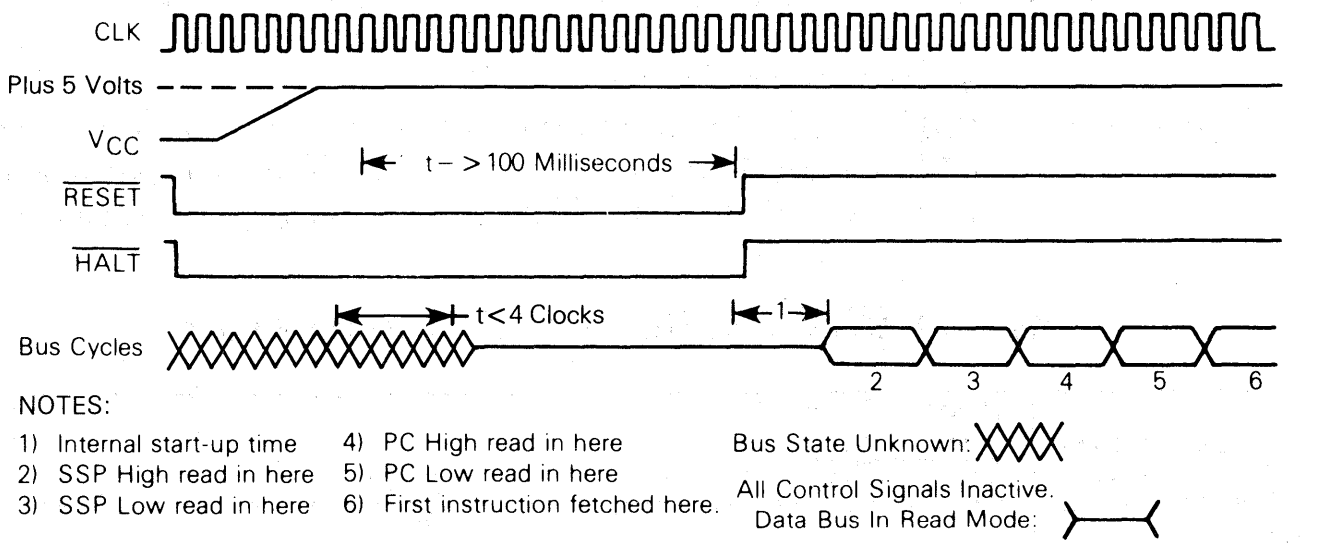


Figure 4-24. Reset Operation Timing Diagram

When the reset and halt lines are driven by an external device, it is recognized as an entire system reset, including the processor. The processor responds by reading the reset vector table entry (vector number zero, address $000000) and loads it into the supervisor stack pointer (SSP). Vector table entry number one at address $000004 is read next and loaded into the program counter. The processor initializes the status register to an interrupt level of seven and the vector base register to $00000000. No other registers are affected by the reset sequence.

When a reset instruction is executed, the processor drives the reset pin for 124 clock periods. In this case, the processor is trying to reset the rest of the system. Therefore, there is no effect on the internal state of the processor. All of the processor's internal registers and the status register are unaffected by the execution of a reset instruction. All external devices connected to the reset line should be reset at the completion of the reset instruction.

Asserting the $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ lines for ten clock cycles will cause a processor reset, except when V_{CC} is initially applied to the processor. In this case, an external reset must be applied for at least 100 milliseconds.

4.3 THE RELATIONSHIP OF $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, AND $\overline{\text{HALT}}$

In order to properly control termination of a bus cycle for a re-run or a bus error condition, $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ should be asserted and negated on the rising edge of the MC68010 clock. This will assure that when two signals are asserted simultaneously, the required setup time (#47) for both of them will be met during the same bus state. This, or some equivalent precaution, should be designed external to the MC68010.

The preferred bus cycle terminations may be summarized as follows (case numbers refer to Table 4-4):

Normal Termination: $\overline{DTACK}$ is asserted, $\overline{BERR}$ and $\overline{HALT}$ remain negated (case 1).

Halt Termination: $\overline{HALT}$ is asserted at same time, or before $\overline{DTACK}$ and $\overline{BERR}$ remains negated (case 2).

Bus Error Termination: $\overline{BERR}$ is asserted in lieu of, at the same time, or before $\overline{DTACK}$ (case 3) or after $\overline{DTACK}$ (case 4) and $\overline{HALT}$ remains negated; $\overline{BERR}$ is negated at the same time or after $\overline{DTACK}$.

Re-Run Termination: $\overline{HALT}$ and $\overline{BERR}$ are asserted in lieu of, at the same time, or before $\overline{DTACK}$ (case 5) or after $\overline{DTACK}$ (case 6); $\overline{BERR}$ is negated at the same time or after $\overline{DTACK}$, $\overline{HALT}$ must be held at least one cycle after $\overline{BERR}$.

Table 4-4. $\overline{DTACK}$, $\overline{BERR}$, and $\overline{HALT}$ Assertion Results

Case No.	Control Signal	Asserted on Rising Edge of State		Result
		N	N+2	
1	$\overline{DTACK}$	A	S	Normal cycle terminate and continue.
	$\overline{BERR}$	NA	NA	
	$\overline{HALT}$	NA	X	
2	$\overline{DTACK}$	A	S	Normal cycle terminate and halt. Continue when $\overline{HALT}$ removed.
	$\overline{BERR}$	NA	NA	
	$\overline{HALT}$	A/S	S	
3	$\overline{DTACK}$	X	X	Terminate and take bus error trap.
	$\overline{BERR}$	A	S	
	$\overline{HALT}$	NA	NA	
4	$\overline{DTACK}$	A	X	Terminate and take bus error trap.
	$\overline{BERR}$	NA	A	
	$\overline{HALT}$	NA	NA	
5	$\overline{DTACK}$	X	X	Terminate and re-run when $\overline{HALT}$ removed.
	$\overline{BERR}$	A	S	
	$\overline{HALT}$	A/S	S	
6	$\overline{DTACK}$	A	X	Terminate and re-run when $\overline{HALT}$ removed.
	$\overline{BERR}$	NA	A	
	$\overline{HALT}$	NA	A	

LEGEND:
- N — the number of the current even bus state (e.g., S4, S6, etc.)
- A — signal is asserted in this bus state
- NA — signal is not asserted in this state
- X — don't care
- S — signal was asserted in previous state and remains asserted in this state

1-331

Table 4-4 details the resulting bus cycle termination under various combinations of control signal sequences. The negation of these same control signals under several conditions is shown in Table 4-5 ($\overline{DTACK}$ is assumed to be negated normally in all cases; for best results, both $\overline{DTACK}$ and $\overline{BERR}$ should be negated when address strobe is negated).

EXAMPLE A:

A system uses a watch-dog timer to terminate accesses to unpopulated address space. The timer asserts $\overline{BERR}$ after time out (case 3).

EXAMPLE B:

A system uses error detection on RAM contents. Designer may:

a) Delay $\overline{\text{DTACK}}$ until data verified, and return $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ simultaneously to re-run error cycle (case 5), or if valid, return $\overline{\text{DTACK}}$ (case 1).

b) Delay $\overline{\text{DTACK}}$ until data verified, and return $\overline{\text{BERR}}$ at same time as $\overline{\text{DTACK}}$ if data in error (case 3).

c) Return $\overline{\text{DTACK}}$ prior to data verification, as described in the next section. If data is invalid, $\overline{\text{BERR}}$ is asserted on next clock cycle (case 4).

d) Return $\overline{\text{DTACK}}$ prior to data verification, if data is invalid assert $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ on next clock cycle (case 6). The memory controller may then correct the RAM prior to or during the re-run.

Table 4-5. $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ Negation Results

Conditions of Termination in Table 4-4	Control Signal	Negated on Rising Edge of State			Results — Next Cycle
		N		N+2	
Bus Error	$\overline{\text{BERR}}$	●	or	●	Takes bus error trap.
	$\overline{\text{HALT}}$	●	or	●	
Re-run	$\overline{\text{BERR}}$	●	or	●	Illegal sequence; usually traps to vector number 0.
	$\overline{\text{HALT}}$	●			
Re-run	$\overline{\text{BERR}}$	●			Re-runs the bus cycle.
	$\overline{\text{HALT}}$			●	
Normal	$\overline{\text{BERR}}$	●			May lengthen next cycle.
	$\overline{\text{HALT}}$	●	or	●	

● = Signal is negated in this bus state.

4.4 ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION

4.4.1 Asynchronous Operation

To achieve clock frequency independence at a system level, the MC68010 can be used in an asynchronous manner. This entails using only the bus handshake lines ($\overline{\text{AS}}$, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$, $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, $\overline{\text{HALT}}$ and $\overline{\text{VPA}}$) to control the data transfer. Using this method, $\overline{\text{AS}}$ signals the start of a bus cycle and the data strobes are used as a condition for valid data on a write cycle. The slave device (memory or peripheral) then responds by placing the requested data on the data bus for a read cycle or latching data on a write cycle and asserting the data transfer acknowledge signal ($\overline{\text{DTACK}}$) to terminate the bus cycle. If no slave responds or the access is invalid, external control logic asserts the $\overline{\text{BERR}}$, or $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$, signal to abort or rerun the bus cycle.

The $\overline{\text{DTACK}}$ signal is allowed to be asserted before the data from a slave device is valid on a read cycle. The length of time that $\overline{\text{DTACK}}$ may precede data is given as parameter #31 and it must be met in any asynchronous system to insure that valid data is latched into the processor. Notice that there is no maximum time specified from the assertion of $\overline{\text{AS}}$ to the assertion of $\overline{\text{DTACK}}$. This is because the MPU will insert wait cycles of one clock period each until $\overline{\text{DTACK}}$ is recognized.

The $\overline{\text{BERR}}$ signal is allowed to be asserted after the $\overline{\text{DTACK}}$ signal is asserted. $\overline{\text{BERR}}$ must be asserted within the time given as parameter #48 after $\overline{\text{DTACK}}$ is asserted in any asynchronous system to insure proper operation. If this maximum delay time is violated, the processor may exhibit erratic behavior.

4.4.2 Synchronous Operation

To allow for those systems which use the system clock as a signal to generate $\overline{\text{DTACK}}$ and other asynchronous inputs, the asynchronous input setup time is given as parameter #47. If this setup is met on an input, such as $\overline{\text{DTACK}}$, the processor is guaranteed to recognize that signal on the next falling edge of the system clock. However, the converse is not true—if the input signal does not meet the setup time it is not guaranteed not to be recognized. In addition, if $\overline{\text{DTACK}}$ is recognized on a falling edge, valid data will be latched into the processor (on a read cycle) on the next falling edge provided that the data meets the setup time given as parameter #27. Given this, parameter #31 may be ignored. Note that if $\overline{\text{DTACK}}$ is asserted, with the required setup time, before the falling edge of S4, no wait states will be incurred and the bus cycle will run at its maximum speed of four clock periods.

In order to assure proper operation in a synchronous system when $\overline{\text{BERR}}$ is asserted after $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$ must meet the setup time parameter #27A prior to the falling edge of the clock one clock cycle after $\overline{\text{DTACK}}$ was recognized. This setup time is critical to proper operation, and the MC68010 may exhibit erratic behavior if it is violated.

NOTE

During an active bus cycle, $\overline{\text{VPA}}$ and $\overline{\text{BERR}}$ are sampled on every falling edge of the clock starting with S0. $\overline{\text{DTACK}}$ is sampled on every falling edge of the clock starting with S4 and data is latched on the falling edge of S6 during a read. The bus cycle will then be terminated in S7 except when $\overline{\text{BERR}}$ is asserted in the absence of $\overline{\text{DTACK}}$, in which case it will terminate one clock cycle later in S9.

SECTION 5
PROCESSING STATES

This section describes the actions of the MC68010 which are outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the processor interrupt priority mask. Finally, the sequence of memory references and actions taken by the processor on exception conditions are detailed.

The MC68010 is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. Two special cases of the normal state are the stopped state, which the processor enters when a STOP instruction is executed, and the loop mode, which the processor may enter when a DBcc instruction is executed. In the stopped state, no further memory references are made and in the loop mode only operand references are made.

The exception processing state is associated with interrupts, trap instructions, tracing and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

5.1 PRIVILEGE STATES

The processor operates in one of two states of privilege: the "supervisor" state or the "user" state. The privilege state determines which operations are legal, are used to choose between the supervisor stack pointer and the user stack pointer in instruction references, and may by used by an external memory management device to control and translate accesses.

The privilege state is a mechanism for providing security in a computer system. Programs should access only their own code and data areas, and ought to be restricted from accessing information which they do not need and must not modify.

The privilege mechanism provides security by allowing most programs to execute in user state. In this state, the accesses are controlled, and the effects on other parts of the system are limited. The operating system executes in the supervisor state, has access to all resources, and performs the overhead tasks for the user programs.

5.1.1 Supervisor State

The supervisor state is the higher state of privilege. For instruction execution, the supervisor state is determined by the S bit of the status register; if the S bit is asserted (high), the processor is in the supervisor state. All instructions can be executed in the supervisor state. The bus cycles generated by instructions executed in the supervisor state are classified as supervisor references. While the processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the supervisor stack pointer.

All exception processing is done in the supervisor state, regardless of the previous setting of the S bit. The bus cycles generated during exception processing are classified as supervisor references. All stacking operations during exception processing use the supervisor stack pointer.

5.1.2 User State

The user state is the lower state of privilege. For instruction execution, the user state is determined by the S bit of the status register; if the S bit is negated (low), the processor is executing instructions in the user state.

Most instructions execute the same in user state as in the supervisor state. However, some instructions which have important system effects are made privileged. User programs are not permitted to execute the STOP instruction, or the RESET instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole status register are privileged. To aid in debugging programs which are to be used as operating systems, the move from status register (MOVE from SR), move to/from user stack pointer (MOVE USP), move to/from control register (MOVEC), and move alternate address space (MOVES) instructions are also privileged.

The bus cycles generated by an instruction executed in the user state are classified as user state references. This allows an external memory management device to translate the address and to control access to protected portions of the address space. While the processor is in the user privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly, access the user stack pointer.

5.1.3 Privilege State Changes

Once the processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the previous setting of the S bit of the status register is saved and the S bit is asserted, putting the processor in the supervisor state. Therefore, when instruction execution resumes at the address specified to process the exception, the processor is in the supervisor privilege state.

5.1.4 Address Space Classification

When the processor makes a reference, it classifies the kind of reference being made, using the encoding on the three function code output lines. This allows external translation of addresses, control of access, and differentiation of special processor state, such as interrupt acknowledge. Table 5-1 lists the classification of address spaces.

Table 5-1. Bus Cycle Classification

Function Code Output			Address Space
FC2	FC1	FC0	
0	0	0	Unassigned, Reserved*
0	0	1	User Data
0	1	0	User Program
0	1	1	Unassigned, Reserved*
1	0	0	Unassigned, Reserved*
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	CPU Space

*Address space 3 is reserved for user definition, while 0 and 4 are reserved for future use by Motorola.

1-333

5.2 EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made and the status register is set for exception processing. In the second step the exception vector is determined and the third step is the saving of the current processor context. In the fourth step a new context is obtained and the processor resumes instruction processing.

5.2.1 Exception Vectors

Exception vectors are memory locations from which the processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length (Figure 5-1), except for the reset vector, which is four words. All exception vectors lie in the supervisor data space, except for the reset vector which is in the supervisor program space. A vector number is an 8-bit number which, when multiplied by four, gives the offset of an exception vector. Vector numbers are generated internally or externally, depending on the cause of the exception. In the case of interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 5-2) to the processor on data bus lines D0 through D7. The processor translates the vector number into a full 32-bit offset which is added to the contents of the vector base register to generate the address used to fetch the vector, as shown in Figure 5-3. The memory layout for exception vectors is given in Table 5-2.

As shown in Table 5-2, the memory layout is 512 words long (1024 bytes). It starts at offset 0 and proceeds through offset 1023. This provides 255 unique vectors; some of these are reserved for TRAPS and other system functions. Of the 255, there are 192 reserved for user interrupt vectors. However, there is no protection on the first 63 entries, so externally generated interrupt vector numbers may reference any of the exception vectors at the discretion of the system designer.

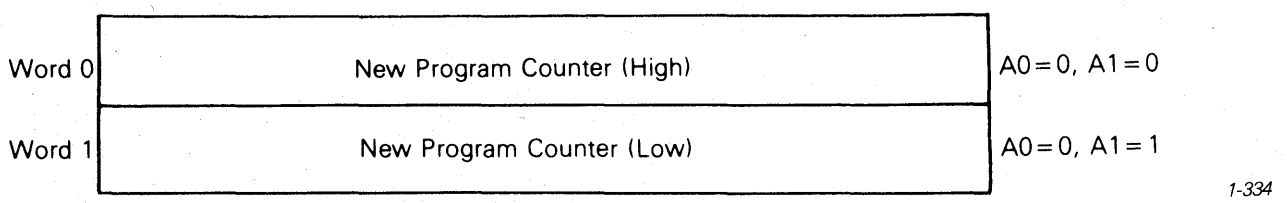
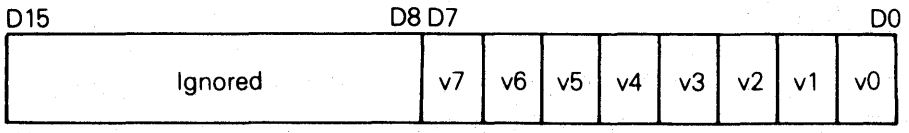


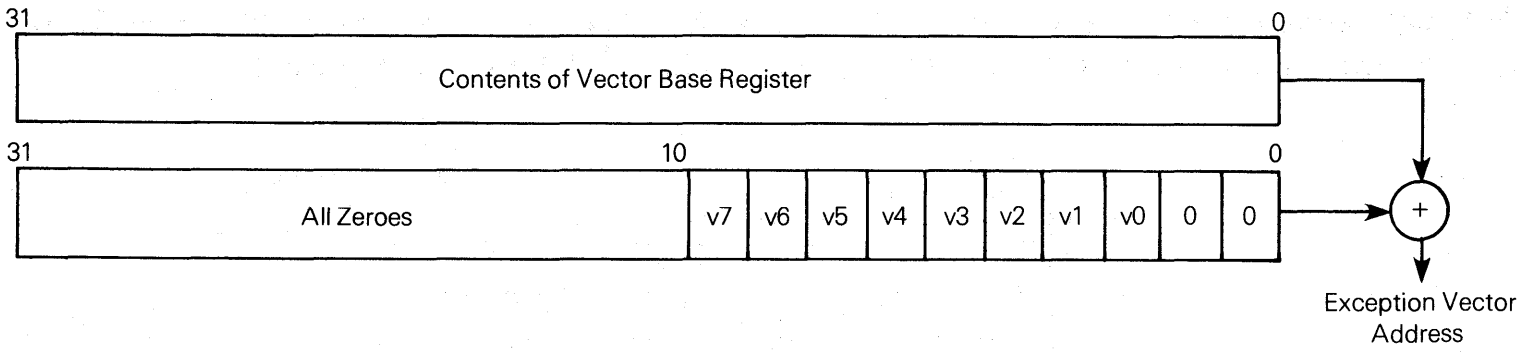
Figure 5-1. Format of Vector Table Entries



Where:
v7 is the MSB of the Vector Number
v0 is the LSB of the Vector Number

1-335

Figure 5-2. Vector Number Format



1-336

Figure 5-3. Exception Vector Address Calculation

Table 5-2. Vector Table

Vector Number(s)	Offset			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
–	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14	56	038	SD	Format Error
15	60	03C	SD	Uninitialized Interrupt Vector
16-23*	64	040	SD	(Unassigned, reserved)
	92	05C		–
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	188	0BC		(#0-15)
48-63*	192	0C0	SD	(Unassigned, reserved)
	252	0FC		–
64-255	256	100	SD	User Interrupt Vectors
	1020	3FC	SD	–

*Vector numbers 12, 13, 16 through 23, and 48 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.

1-337

5.2.2 Exception Stack Frame

Exception processing saves the most volatile portion of the current processor context on the top of the supervisor stack. This context is organized in a format called the exception stack frame. This information always includes the status register and program counter of the processor when the exception occurred. In order to support generic handlers, the processor also places the vector offset in the exception stack frame. The format code field allows the RTE (return from exception) instruction to identify what information is on the stack so that it may be properly restored. The general form of the exception stack frame is illustrated in Figure 5-4; Table 5-3 lists the MC68010 stack frame codes. Although some formats are peculiar to a particular M68000 Family processor, the format 0000 is always legal, and indicates that just the first four words of the frame are present.

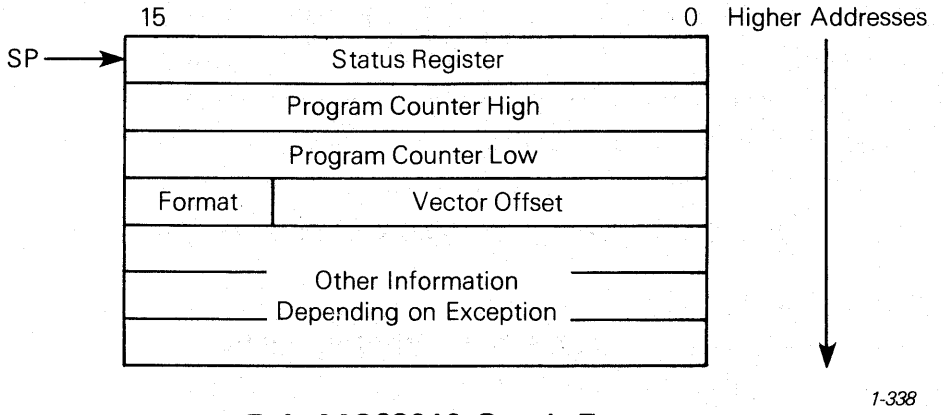


Figure 5-4. MC68010 Stack Format

Table 5-3. MC68010 Format Codes

Format Code	Stacked Information
0000	MC68010 Short Format (4 Words)
1000	MC68010 Long Format (29 Words)
All Others	Unassigned, Reserved

1-339

5.2.3 Kinds of Exceptions

Exceptions can be generated by either internal or external causes. The externally generated exceptions are the interrupts, bus error, and reset requests. The interrupts are requests from peripheral devices for processor action while the bus error and reset inputs are used for access control and processor restart. The internally generated exceptions come from instructions, or from address errors or tracing. The trap (TRAP), trap on overflow (TRAPV), check data register against upper bounds (CHK), and divide (DIV) instructions all can generate exceptions as part of their instruction execution. In addition, illegal instructions, word or long word fetches from odd addresses, and privilege violations cause exceptions. Tracing behaves like a very high-priority internally-generated interrupt after each instruction execution.

5.2.4 Exception Processing Sequence

Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the S bit is asserted, putting the processor into the

supervisor privilege state. Also, the T bit is negated which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the offset of the exception vector and is added to the vector base register.

The third step is to save the current processor status, except for the reset exception. The exception stack frame is created at the top of the supervisor stack. The current program counter value, the saved copy of the status register, and the format/offset word are written into the stack frame. The program counter value stacked usually points to the next unexecuted instruction; however, for bus error and address error, the value stacked for the program counter is unpredictable, and may be incremented by up to five words from the address of the instruction which caused the error. Group 1 and 2 exceptions (see **5.2.5 Multiple Exceptions**) use a short format exception stack frame (format = 0000). Additional information defining the current context is stacked for the bus error and address error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector table. The processor then resumes instruction execution. The instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution is started.

5.2.5 Multiple Exceptions

These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped according to their occurrence and priority. The group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted and the exception processing to commence within two clock cycles. The group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instructions. These exceptions allow the current instruction to execute to completion, but pre-empt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Group 0 exceptions have highest priority, while group 2 exceptions have lowest priority. Within group 0, reset has highest priority, followed by address error and then bus error. Within group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, there is no priority relation within group 2.

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instruction processing is suspended. In another example, if an interrupt request occurs during the execution of an instruction while the T bit is asserted,

the trace exception has priority, and is processed first. Before instruction processing resumes, however, the interrupt exception is also processed, and instruction processing commences finally in the interrupt handler routine. A summary of exception grouping and priority is given in Table 5-4.

Table 5-4. Exception Grouping and Priority

Group	Exception	Processing
0	Reset Address Error Bus Error	Exception processing begins within two clock cycles
1	Trace Interrupt Illegal Privilege	Exception processing begins before the next instruction
2	TRAP, TRAPV, CHK, Zero Divide Format Error	Exception processing is started by normal instruction execution

1-586

5.3 EXCEPTION PROCESSING IN DETAIL

Exceptions have a number of sources and each exception has processing which is peculiar to it. The following paragraphs detail the sources of exceptions, how each arises, and how each is processed.

5.3.1 Reset

The reset input provides the highest exception level. The processing of the reset signal is designed for system initiation, and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state, the trace state is forced off, and the processor interrupt priority mask is set to level seven. The vector base register is set to $00000000 and the vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is fetched as the initial supervisor stack pointer, and the address in the last two words of the reset exception vector is fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The power-up/restart code should be pointed to by the initial program counter.

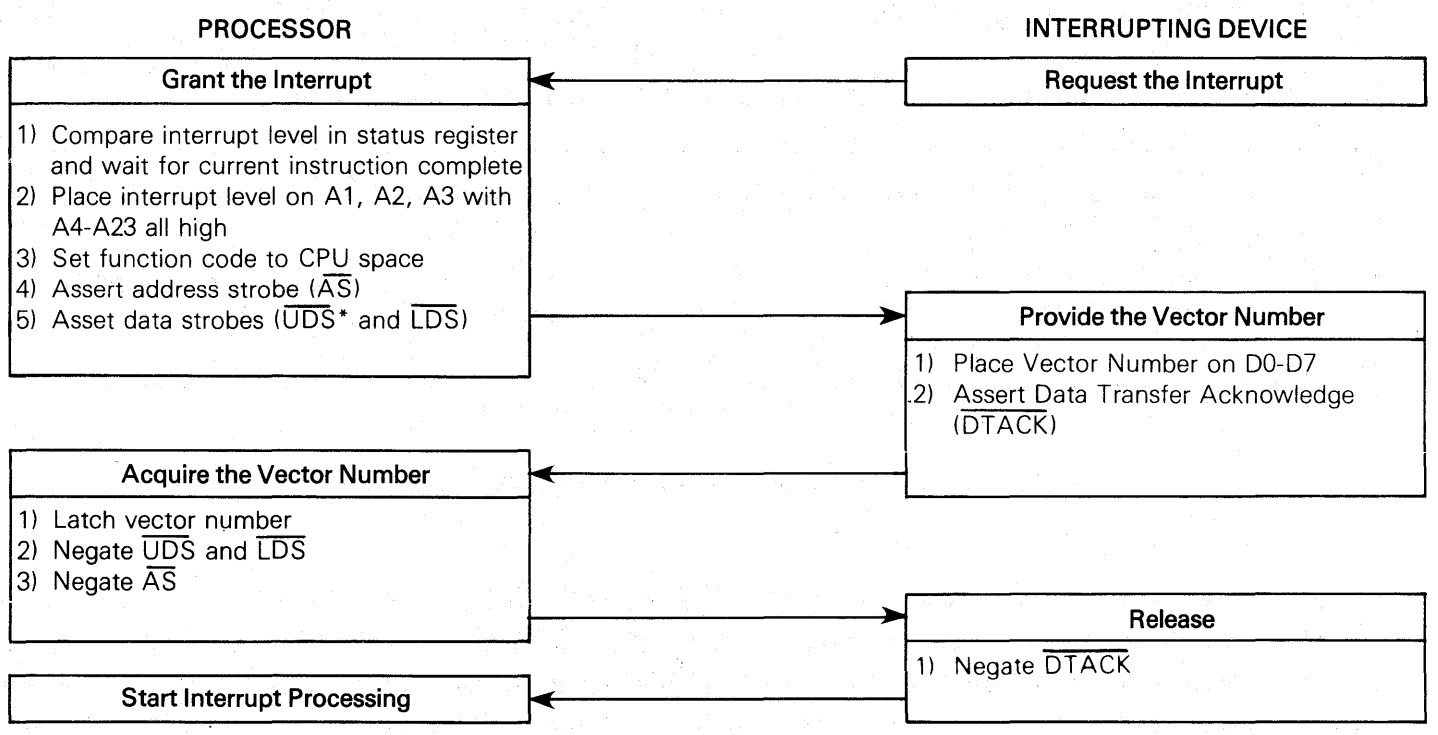
The reset instruction does not cause loading of the reset vector, but does assert the reset line to reset external devices. This allows the software to reset the system to a known state and then continue processing with the next instruction.

5.3.2 Interrupts

Seven levels of interrupt priorities are provided. Devices may be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. Interrupt priority levels are numbered from one to seven, level seven being the highest priority. The status register contains a 3-bit mask which indicates the current processor priority, and interrupts are inhibited for all priority levels less than or equal to the current processor priority.

An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines; a zero indicates no interrupt request. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts may cause exception processing to start at the end of an instruction depending on the current processor priority level. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction and the interrupt exception processing is postponed. (The recognition of level seven is slightly different, as explained in the following paragraph.)

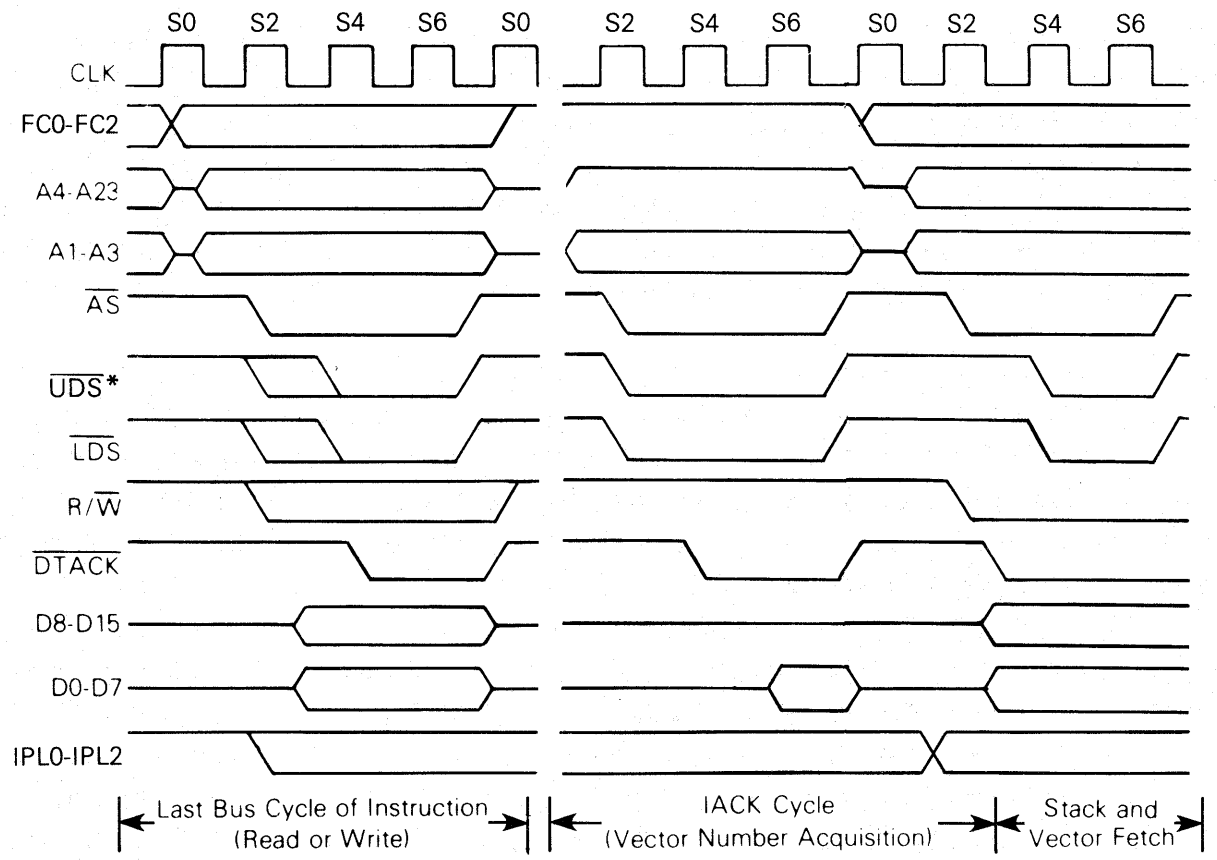
If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. A copy of the status register is saved, the privilege state is set to supervisor state, tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged. The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the format/offset word, program counter, and status register on the supervisor stack. The offset value in the format/offset word is the externally supplied or internally generated vector number multiplied by four. The format will be all zeroes. The saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine. A flowchart for the interrupt acknowledge sequence is given in Figure 5-5, a timing diagram is given in Figure 5-6, and the interrupt processing sequence is shown in Figure 5-7.



* Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D8 through D15 at this time.

1-340

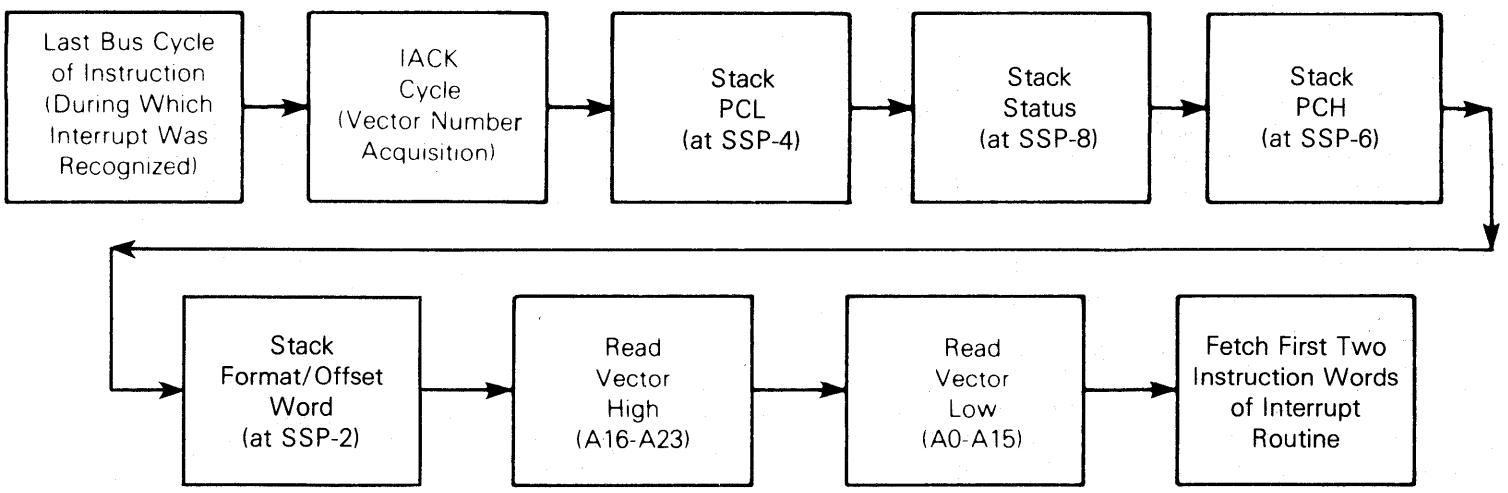
Figure 5-5. Vector Acquisition Flowchart



* Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D8 through D15 at this time.

1-341

Figure 5-6. Interrupt Acknowledge Cycle Timing Diagram



NOTE: SSP refers to the value of the supervisor stack pointer before the interrupt occurs.

1-342

Figure 5-7. Interrupt Processing Sequence

Priority level seven is a special case. Level seven interrupts cannot be inhibited by the interrupt priority mask, thus providing a "non-maskable interrupt" capability. An interrupt is generated each time the interrupt request level changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

5.3.3 Uninitialized Interrupt

An interrupting device asserts $\overline{\text{VPA}}$, $\overline{\text{BERR}}$, or provides an interrupt vector number and asserts $\overline{\text{DTACK}}$ during an interrupt acknowledge cycle by the MC68010. If the vector register has not been initialized, the responding M68000 Family peripheral will provide vector number 15, the uninitialized interrupt vector. This provides a uniform way to recover from a programming error.

5.3.4 Spurious Interrupt

If during the interrupt acknowledge cycle no device responds by asserting $\overline{\text{DTACK}}$ or $\overline{\text{VPA}}$, $\overline{\text{BERR}}$ should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by forming a short format exception stack and fetching the spurious interrupt vector instead of the bus error vector. The processor then proceeds with the usual exception processing.

5.3.5 Instruction Traps

Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception and is useful for implementing supervisor calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a runtime error, which may be an arithemetic overflow or a subscript out of bounds.

The signed divide (DIVS) and unsigned divide (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

5.3.6 Illegal and Unimplemented Instructions

Illegal instruction is the term used to refer to any of the word bit patterns which are not the bit patterns of the first word of a legal MC68010 instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs. Motorola reserves the right to define instructions whose opcodes may be any of the illegal instructions. Three bit patterns will always force an illegal instruction trap on all M68000 Family compatible microprocessors. They are: $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for Motorola system products. The third pattern, $4AFC, is reserved for customer use.

In addition to the previously defined illegal instruction opcodes, the MC68010 defines eight breakpoint illegal instructions with the bit patterns $4848-$484F. These instructions cause the processor to enter illegal instruction exception processing as usual, but a breakpoint bus cycle is executed before the stacking operations are performed as shown in Figure 5-8. The processor does not accept or send any data during this cycle. Whether the breakpoint cycle is terminated with a $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, or $\overline{\text{VPA}}$ signal, the processor will continue with the illegal instruction processing. The purpose of this cycle is to provide a software breakpoint that will signal external hardware when it is executed.

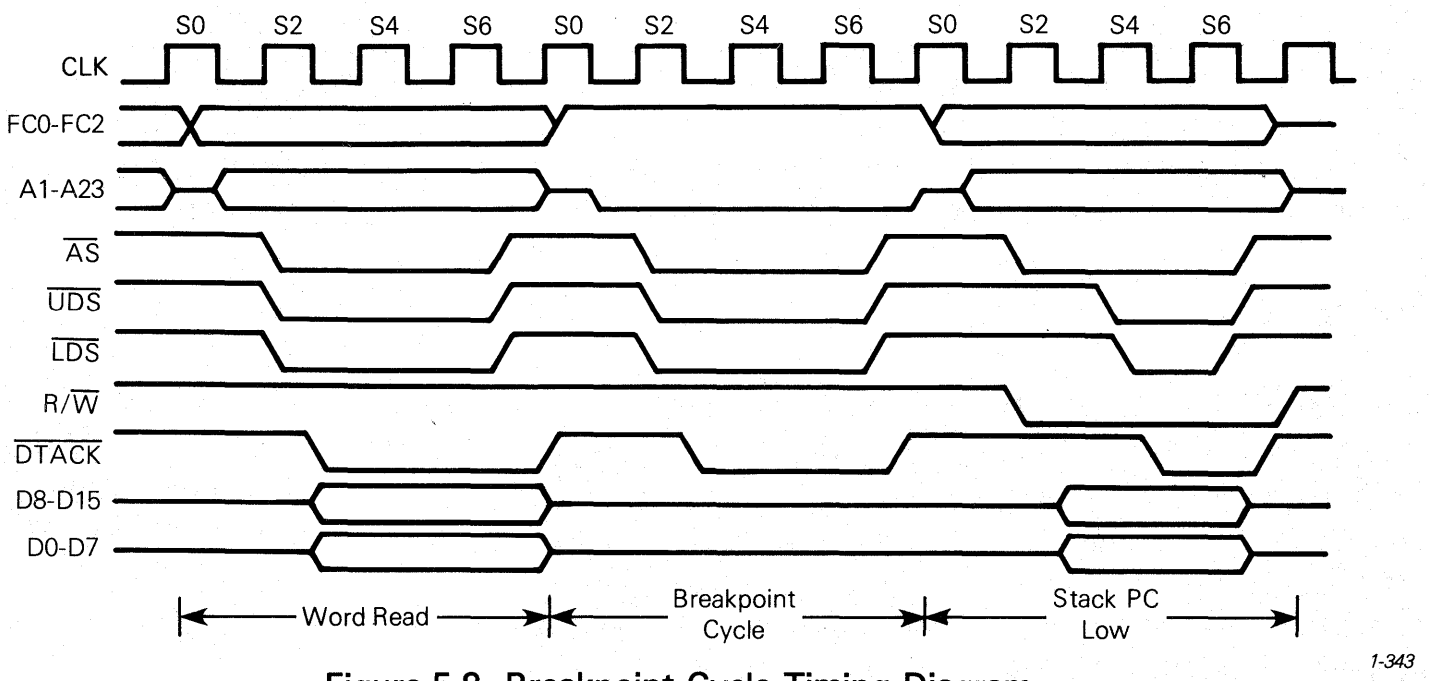


Figure 5-8. Breakpoint Cycle Timing Diagram

Word patterns with bits 12-15 equaling 1010 or 1111 are distinguished as unimplemented instructions and separate exception vectors are given to these patterns to permit efficient emulation. This facility allows the operating system to detect program errors, or to emulate unimplemented instructions, such as the MC68881 Floating-Point Coprocessor instructions, in software.

5.3.7 Privilege Violations

In order to provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user state will cause an exception. The privileged instructions are:

AND Immediate to SR
EOR Immediate to SR
MOVE to SR
MOVE from SR
MOVEC
MOVES
MOVE USP
OR Immediate to SR
RESET
RTE
STOP

5.3.8 Tracing

To aid in program development, the MC68010 includes a facility to allow instruction-by-instruction tracing. In the trace state, after each instruction is executed an exception is forced, allowing a debugging program to monitor the execution of the program under test

The trace facility uses the T bit in the supervisor portion of the status register. If the T bit is negated (off), tracing is disabled, and instruction execution proceeds from instruction to instruction as normal. If the T bit is asserted (on) at the beginning of the execution of an instruction, a trace exception will be generated as the execution of that instruction is completed. If the instruction is not executed, either because an interrupt is taken, or the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus error, or address error exception. If the instruction is indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If, during

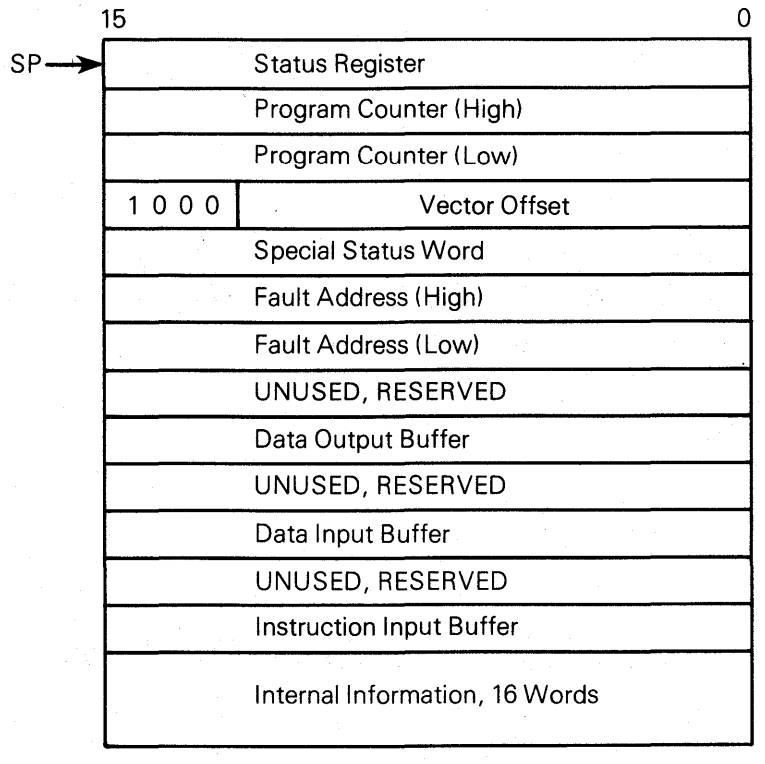
the execution of the instruction an exception is forced by that instruction, the forced exception is processed before the trace exception.

As an extreme illustration of the above rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.

5.3.9 Bus Error

Bus error exceptions occur when external logic terminates a bus cycle with a bus error signal. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing. However, if a bus error occurs during exception processing for a bus error, address error, or reset, the processor detects a double bus fault and halts. When exception processing is completed, instruction execution continues at the address contained in exception vector table entry two, at offset $008.

Exception processing for a bus error follows a slightly different sequence than the sequence for group 1 and 2 exceptions. In addition to the four steps executed during exception processing for all other exceptions, 22 words of additional information are placed on the stack. This additional information describes the internal state of the processor at the time of the bus error and is reloaded by the RTE instruction to continue the instruction that caused the error. Figure 5-9 shows the order of the stacked information.



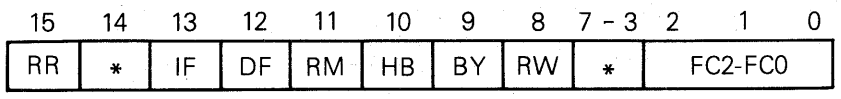
NOTE: The stack pointer is decremented by 29 words, although only 26 words of information are actually written to memory. The three additional words are reserved for future use by Motorola.

1-344

Figure 5-9. Exception Stack Order (Bus and Address Error)

The value of the saved program counter does not necessarily point to the instruction that was executing when the bus error occurred, but may be advanced by up to five words. This is due to the prefetch mechanism on the MC68010 that always fetches a new instruction word as each previously fetched instruction word is used (see **7.1.2 Instruction Prefetch**). However, enough information is placed on the stack for the bus error exception handler routine to determine why the bus fault occurred. This additional information includes the address that was being accessed, the function codes for the access, whether it was a read or a write, and what internal register was included in the transfer. The fault address can be used by an operating system to determine what virtual memory location is needed so that the requested data can be brought into physical memory. The RTE instruction is then used to reload the processor's internal state at the time of the fault, the faulted bus cycle will then be re-run and the suspended instruction completed. If the faulted bus cycle was a read-modify-write, the entire cycle will be re-run whether the fault occurred during the read or the write operation.

An alternate method of handling a bus error is to complete the faulted access in software. In order to use this method, use of the special status word, the instruction input buffer, the data input buffer, and the data output buffer image is required. The format of the special status word is shown in Figure 5-10. If the bus cycle was a write, the data output buffer image should be written to the fault address location using the function code contained in the special status word. If the cycle was a read, the data at the fault address location should be written to the images of the data input buffer, instruction input buffer, or both according to the DF and 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. This is because the entire read-modify-write cycle is assumed to have been completed by software. Once the cycle has been completed by software, the RR bit in the special status word is set to indicate to the processor that it should not re-run the cycle when the RTE instruction is executed. If the re-run flag is set when an RTE instruction executes, the MC68010 still reads all of the information from the stack.



RR — Re-run flag; 0 = processor re-run (default), 1 = software re-run.
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.

1-345

Figure 5-10. Special Status Word Format

*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 HB bit is set. This condition will only occur if a MOVEP instruction caused the fault during the transfer of bits 8-15 of a word or long word or bits 24-31 of a long word.

5.3.10 Address Error

Address error exceptions occur when the processor attempts to access a word or long word operand or an instruction at an odd address. The effect is much like an internally generated bus error, so that the bus cycle is aborted, and the processor begins exception processing. After exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector offset refers to the address error exception vector. If an address error occurs during exception processing for a bus error, address error, or reset, the processor detects a double bus fault and halts.

As shown in Figure 5-11, an address error will execute a short bus cycle followed by exception processing. This short bus cycle is similar to a normal read or write cycle, except that the data strobes are not asserted and no external signals are used to terminate the cycle. During an address error bus cycle, AS is asserted to indicate that the MC68010 will drive the address bus (thus allowing for proper operation in a multiple bus master system). Note that data strobes are not asserted allowing for address error detection and memory protection.

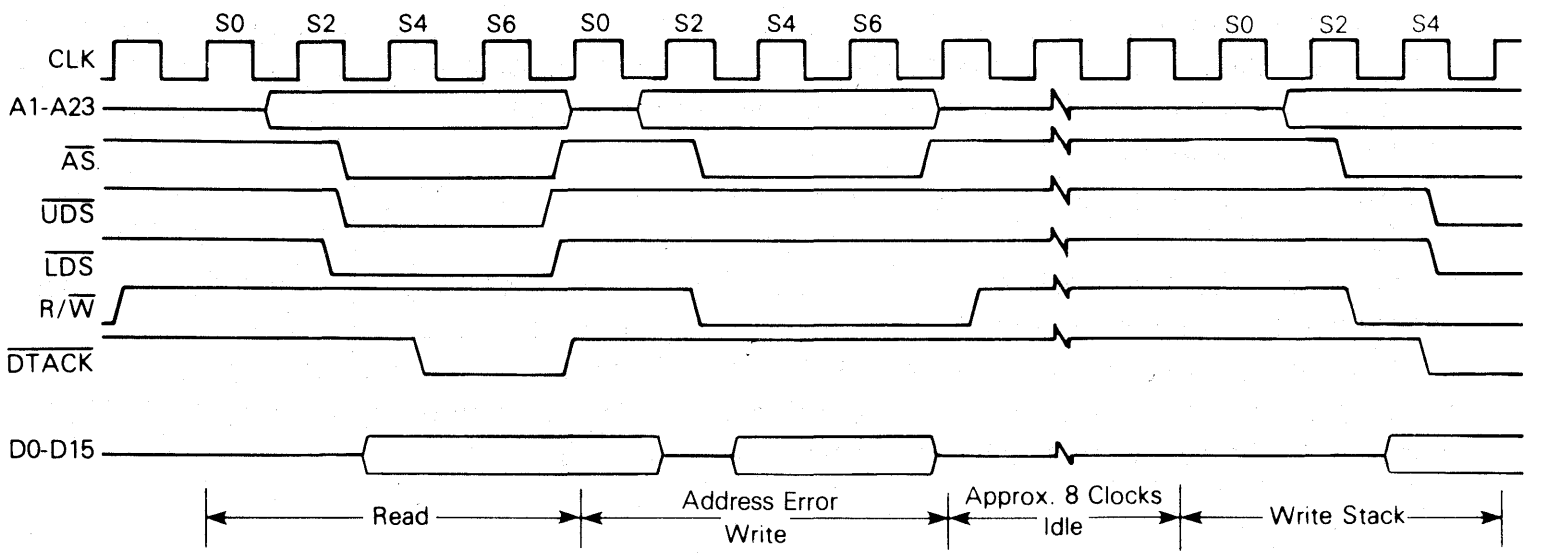


Figure 5-11. Address Error Timing Diagram

Since the address error exception stacks the same information that is stacked by a bus error exception, it is possible to use the RTE instruction to continue execution of the suspended instruction. However, if the software re-run flag is not set, the fault address will be used when the cycle is re-run and another address error exception will occur. 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.

5.4 RETURN FROM EXCEPTION

In addition to returning from any exception handler routine, the RTE instruction is used to resume the execution of a suspended instruction by restoring all of the temporary register and control information stored during a bus error and returning to the normal processing state. For the RTE instruction to execute properly, the stack must contain valid and accessible data. The RTE instruction

checks for data validity in two ways; first, by checking the format/offset word for a valid stack format code, and second, if the format code indicates the long stack format, the long stack data is checked for validity as it is loaded into the processor. In addition, the data is checked for accessibility when the processor starts reading the long data. Because of these checks, the RTE instruction executes as follows:

1. Determine the stack format. This step is the same for any stack format and consists of reading the status register, program counter, and format/offset word. If the format code indicates a short stack format, execution continues at the new program counter address. If the format code is not one of the MC68010 defined stack format codes, exception processing starts for a format error.
2. Determine data validity. For a long stack format, the MC68010 will begin to read the remaining stack data, checking for validity of the data. The only word checked for validity is the first of the 16 internal information words (SP + 26) shown in Figure 5-9. This word contains a processor version number in bits 10 through 13, that must match the version number of the MC68010 that is attempting to read the data. This validity check is used to insure that in dual processor systems, the data will be properly interpreted by the RTE instruction if the two processors are of different versions. If the version number is incorrect for this processor, the RTE instruction will be aborted and exception processing will begin for a format error exception. Since the stack pointer is not updated until the RTE instruction has successfully read all of the stack data, a format error occurring at this point will not stack new data over the previous bus error stack information.
3. Determine data accessibility. If the long stack data is valid, the MC68010 performs a read from the last word (SP + 56) of the long stack to determine data accessibility. If this read is terminated normally, the processor assumes that the remaining words on the stack frame are also accessible. If a bus error is signaled before or during this read, a bus error exception is taken as usual. After this read, the processor must be able to load the remaining data without receiving a bus error; therefore, if a bus error occurs on any of the remaining stack reads, the MC68010 treats this as a double bus fault and enters the halted state.

SECTION 6
INTERFACE WITH M6800 PERIPHERALS

Motorola's extensive line of M6800 peripherals are directly compatible with the MC68010. Some of these devices that are particularly useful are:

MC6821 Peripheral Interface Adapter
MC6840 Programmable Timer Module
MC6843 Floppy Disk Controller
MC6845 CRT Controller
MC6850 Asynchronous Communications Interface Adapter
MC6852 Synchronous Serial Data Adapter
MC6854 Advanced Data Link Controller
MC68488 General Purpose Interface Adapter

To interface the synchronous M6800 peripherals with the asynchronous MC68010, the processor modifies its bus cycle to meet the M6800 cycle requirements whenever an M6800 device address is detected. This is possible since both processors use memory mapped I/O. Figure 6-1 is a flowchart of the interface operation between the processor and M6800 devices.

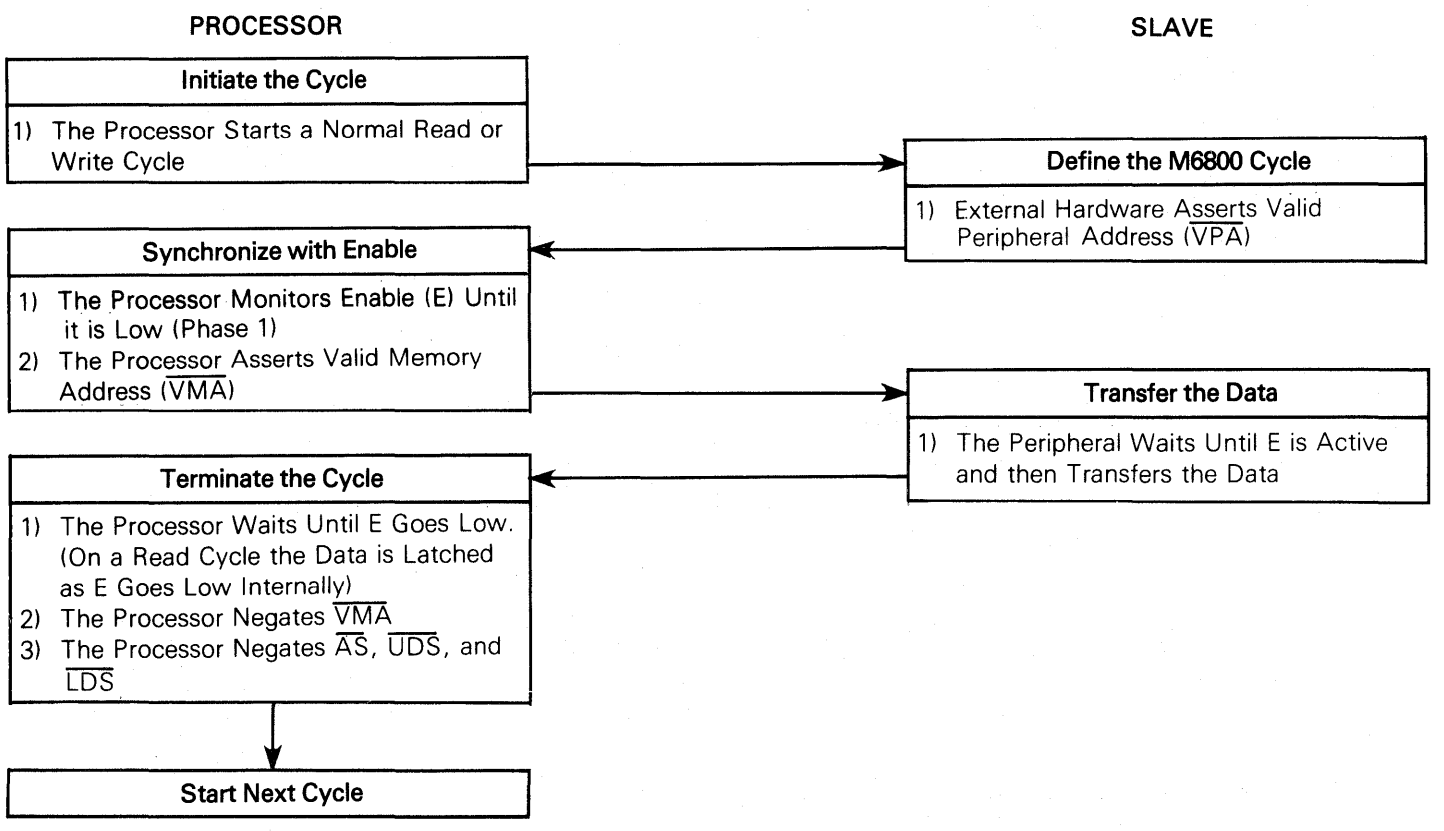


Figure 6-1. M6800 Interfacing Flowchart

6.1 DATA TRANSFER OPERATION

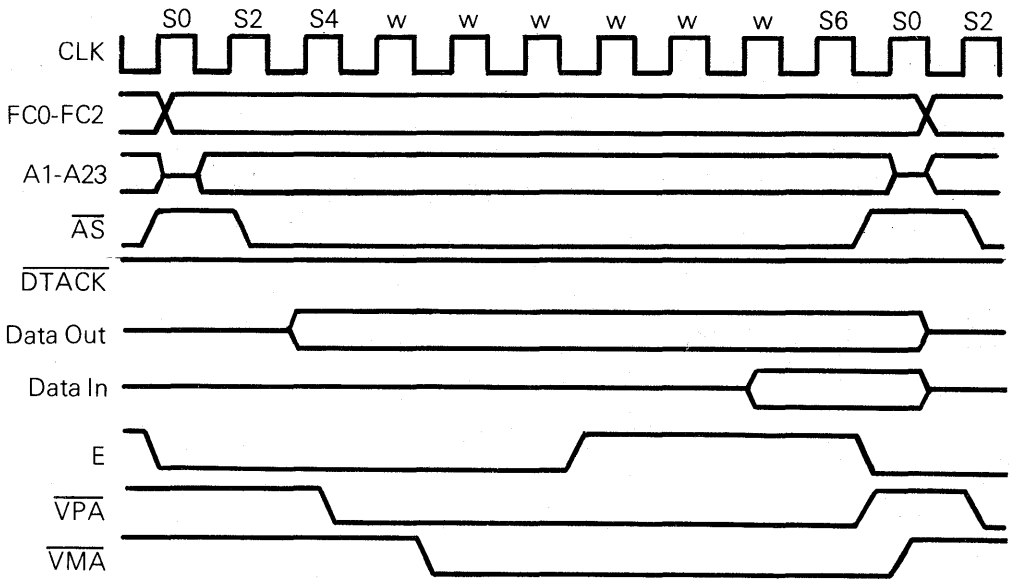
Three signals on the processor provide the M6800 interface. They are: enable (E), valid memory address ($\overline{VMA}$), and valid peripheral address ($\overline{VPA}$). Enable corresponds to the E or phase 2 signal in existing M6800 systems. The bus frequency is one tenth of the incoming MC68010 clock frequency. The timing of E allows 1 megahertz peripherals to be used with an 8 megahertz MC68010. Enable has a 60/40 duty cycle; that is, it is low for six input clocks and high for four input clocks. This duty cycle allows the processor to do successive $\overline{VPA}$ accesses on successive E pulses.

M6800 cycle timing is given in Figure 6-2. At state zero (S0) in the cycle, the address bus is in the high-impedence state. A function code is asserted on the function code output lines. One-half clock later, in state 1, the address bus is released from the high-impedence state.

During state 2, the address strobe ($\overline{AS}$) is asserted to indicate that there is a valid address on the address bus. If the bus cycle is a read cycle, the upper and/or lower data strobes are also asserted in state 2. If the bus cycle is a write cycle, the read/write (R/$\overline{W}$) signal is switched to low (write) during state 2. One-half clock later, in state 3, the write data is placed on the data bus, and in state 4 the data strobes are issued to indicate valid data on the data bus. The processor now inserts wait states until it recognizes the assertion of $\overline{VPA}$.

The $\overline{VPA}$ input signals 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 2 transfer characteristics of the M6800 bus. Valid peripheral address is derived by decoding the address bus, conditioned by the address strobe. Chip select for the M6800 peripherals should be derived by decoding the address bus conditioned by $\overline{VMA}$.

After recognition of $\overline{VPA}$, the processor assures that the enable (E) is low, by waiting if necessary, and subsequently asserts $\overline{VMA}$ two clock cycles before E goes high. $\overline{VMA}$ is then used as part of the chip select equation of the peripheral. This ensures that the M6800 peripherals are selected and deselected at the correct time. The peripheral now runs its cycle during the high portion of the E signal. Figures 6-2 and 6-3 depict the best and worst case M6800 cycle timing; this cycle length is dependent strictly upon when $\overline{VPA}$ is asserted in relationship to the E clock.



1-348

Figure 6-2. MC68010 to M6800 Peripheral Timing Diagram — Best Case

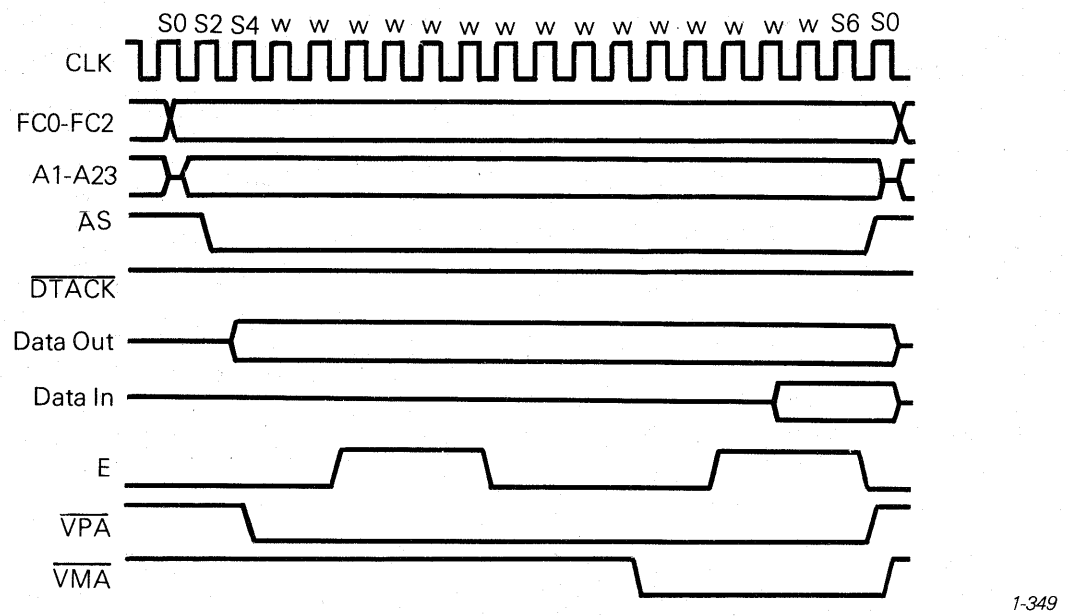


Figure 6-3. MC68010 to M6800 Peripheral Timing Diagram — Worst Case

If we assume that external circuitry asserts $\overline{VPA}$ as soon as possible after the assertion of $\overline{AS}$, then $\overline{VPA}$ will be recognized as being asserted on the falling edge of S4. In this case, no "extra" wait cycles will be inserted prior to the recognition of $\overline{VPA}$ asserted and only the wait cycles inserted to synchronize with the E clock will determine the total length of the cycle. In any case, the synchronization delay will be some integral number of clock cycles within the following two extremes:

1. Best Case — $\overline{VPA}$ is recognized as being asserted on the falling edge three clock cycles before E rises (or three clock cycles after E falls).
2. Worst Case — $\overline{VPA}$ is recognized as being asserted on the falling edge two clock cycles before E rises (or four clock cycles after E falls).

During a read cycle, the processor latches the peripheral data in state 6. For all cycles, the processor negates the address and data strobes one-half clock cycle later in state 7 and the enable signal goes low at this time. Another half clock later, the address bus is put in the high-impedence state. During a write cycle, the data bus is put in the high-impedence state and the read/write signal is switched high. The peripheral logic must remove $\overline{VPA}$ within one clock after the address strobe is negated.

$\overline{DTACK}$ should not be asserted while $\overline{VPA}$ is asserted. Notice that the MC68010 $\overline{VMA}$ is active low, contrasted with the active high M6800 VMA. This allows the processor to put its buses in the high-impedence state on DMA requests without inadvertently selecting the peripherals.

6.2 AC ELECTRICAL SPECIFICATIONS

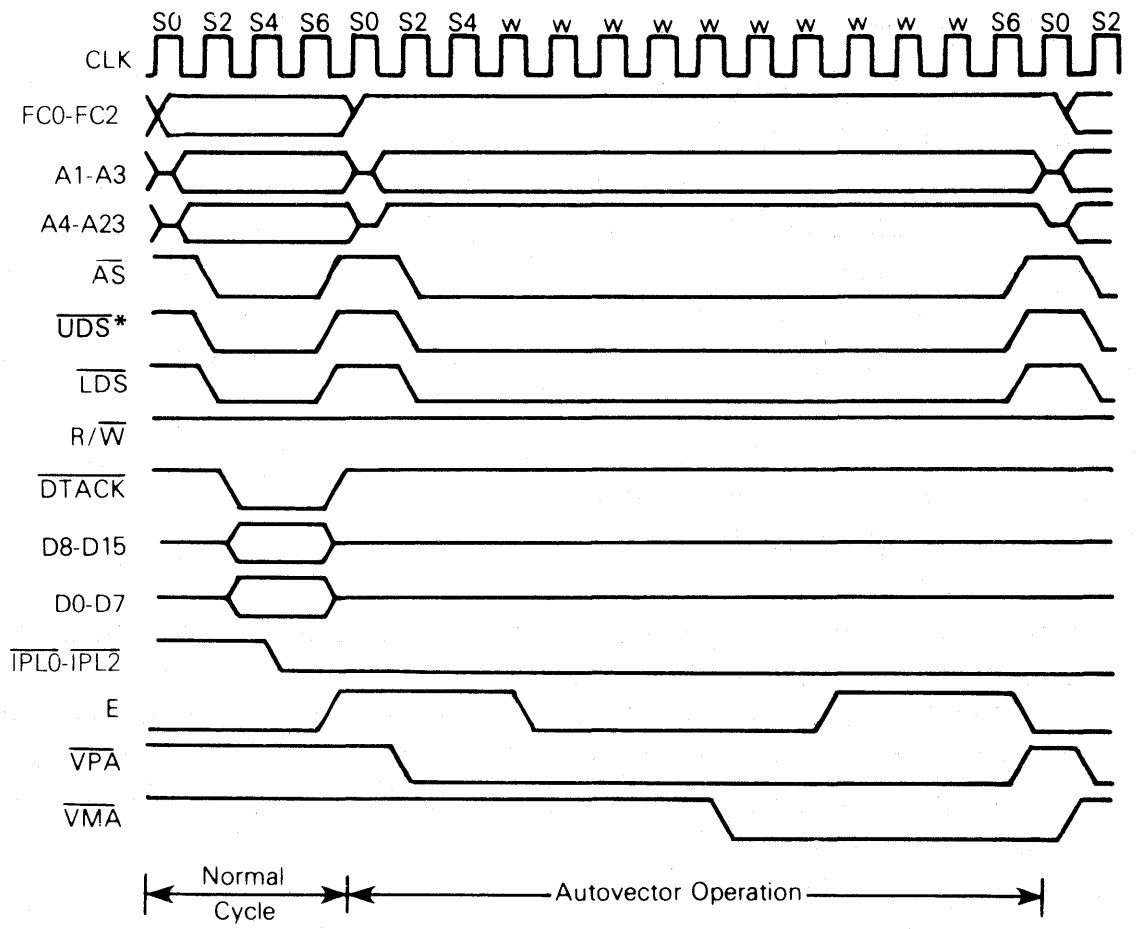
The electrical specifications for interfacing the MC68010 to M6800 Family peripherals are located in Section 8.

6.3 INTERRUPT INTERFACE OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector, if $\overline{VPA}$ is asserted the MC68010 will assert $\overline{VMA}$ and complete a normal M6800 read cycle as shown in Figure 6-4. The processor will then use an internally generated vector that is a function of the interrupt being serviced. This process is known as autovectoring. The seven autovectors are vector numbers 25 through 31 (decimal).

Autovectoring operates in the same fashion (but is not restricted to) the M6800 interrupt sequence. The basic difference is that there are six normal interrupt vectors and one NMI type vector. As with both the M6800 and the MC68010's normal vectored interrupt, the interrupt service routine can be located anywhere in the address space. This is due to the fact that while the vector numbers are fixed, the contents of the vector table entries are assigned by the user.

Since $\overline{VMA}$ is asserted during autovectoring, the M6800 peripheral address decoding should prevent unintended accesses.



*Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D0 through D15 during autovector cycles.

1-350

Figure 6-4. Autovector Operation Timing Diagram

SECTION 7
INSTRUCTION SET AND EXECUTION TIMES

7.1 INSTRUCTION SET

The following paragraphs provide information about the addressing categories and instruction set of the MC68010/MC68012.

7.1.1 Addressing Categories

Effective address modes may be categorized by the ways in which they may be used. The following classifications will be used in the instruction definitions.

Data	If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.
Memory	If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.
Alterable	If an effective address mode may be used to refer to alterable (writeable) operands, it is considered an alterable addressing effective address mode.
Control	If an effective address mode may be used to refer to memory operands without an associated size, it is considered a control addressing effective address mode.

These categories may be combined, so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable. Table 7-1 shows the various categories to which each of the effective address modes belong. Table 7-2 is the instruction set summary.

Table 7-1. Effective Addressing Mode Categories

Effective Address Modes	Mode	Register	Data	Addressing Categories		
				Memory	Control	Alterable
Dn	000	Register Number	X	–	–	X
An	001	Register Number	–	–	–	X
(An)	010	Register Number	X	X	X	X
(An)+	011	Register Number	X	X	–	X
–(An)	100	Register Number	X	X	–	X
d(An)	101	Register Number	X	X	X	X
d(An, ix)	110	Register Number	X	X	X	X
xxx.W	111	000	X	X	X	X
xxx.L	111	001	X	X	X	X
d(PC)	111	010	X	X	X	–
d(PC, ix)	111	011	X	X	X	–
#xxx	111	100	X	X	–	–

1-351

Table 7-2. Instruction Set (Sheet 1 of 2)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
ABCD	Add Decimal with Extend	(Destination)$_{10}$ + (Source)$_{10}$ + X → Destination	*	U	*	U	*
ADD	Add Binary	(Destination) + (Source) → Destination	*	*	*	*	*
ADDA	Add Address	(Destination) + (Source) → Destination	–	–	–	–	–
ADDI	Add Immediate	(Destination) + Immediate Data → Destination	*	*	*	*	*
ADDQ	Add Quick	(Destination) + Immediate Data → Destination	*	*	*	*	*
ADDX	Add Extended	(Destination) + (Source) + X → Destination	*	*	*	*	*
AND	AND Logical	(Destination) Λ (Source) → Destination	–	*	*	0	0
ANDI	AND Immediate	(Destination) Λ Immediate Data → Destination	–	*	*	0	0
ANDI to CCR	AND Immediate to Condition Codes	(Source) Λ CCR → CCR	*	*	*	*	*
ANDI to SR	AND Immediate to Status Register	(Source) Λ SR → SR	*	*	*	*	*
ASL, ASR	Arithmetic Shift	(Destination) Shifted by <count> → Destination	*	*	*	*	*
B$_{CC}$	Branch Conditionally	If $_{CC}$ then PC + d → PC	–	–	–	–	–
BCHG	Test a Bit and Change	~(<bit number>) OF Destination → Z ~(<bit number>) OF Destination → <bit number> OF Destination	–	–	*	–	–
BCLR	Test a Bit and Clear	~(<bit number>) OF Destination → Z 0 → <bit number> → OF Destination	–	–	*	–	–
BRA	Branch Always	(PC) + d → PC	–	–	–	–	–
BSET	Test a Bit and Set	~(<bit number>) OF Destination → Z 1 → <bit number> OF Destination	–	–	*	–	–
BSR	Branch to Subroutine	(PC) → – (SP); (PC) + d → PC	–	–	–	–	–
BTST	Test a Bit	~(<bit number>) OF Destination → Z	–	–	*	–	–
CHK	Check Register Against Bounds	If Dn <0 or Dn> (<ea>) then TRAP	–	*	U	U	U
CLR	Clear and Operand	0 → Destination	–	0	1	0	0
CMP	Compare	(Destination) – (Source)	–	*	*	*	*
CMPA	Compare Address	(Destination) – (Source)	–	*	*	*	*
CMPI	Compare Immediate	(Destination) – Immediate Data	–	*	*	*	*
CMPM	Compare Memory	(Destination) – (Source)	–	*	*	*	*
DB$_{CC}$	Test Condition, Decrement and Branch	If ~$_{CC}$ then Dn – 1 → Dn; if Dn ≠ – 1 then PC + d → PC	–	–	–	–	–
DIVS	Signed Divide	(Destination)/(Source) → Destination	–	*	*	*	0
DIVU	Unsigned Divide	(Destination)/(Source) → Destination	–	*	*	*	0
EOR	Exclusive OR Logical	(Destination) ⊕ (Source) → Destination	–	*	*	0	0
EORI	Exclusive OR Immediate	(Destination) ⊕ Immediate Data → Destination	–	*	*	0	0
EORI to CCR	Exclusive OR Immediate to Condition Codes	(Source) ⊕ CCR → CCR	*	*	*	*	*
EORI to SR	Exclusive OR Immediate to Status Register	(Source ⊕ SR → SR	*	*	*	*	*
EXG	Exchange Register	(Rx) ↔ (Ry)	–	–	–	–	–
EXT	Sign Extend	(Destination) Sign-Extended → Destination	–	*	*	0	0
JMP	Jump	(Destination) → PC	–	–	–	–	–
JSR	Jump to Subroutine	(PC) → – (SP); Destination → PC	–	–	–	–	–
LEA	Load Effective Address	<ea> → An	–	–	–	–	–
LINK	Link and Allocate	(An) → – (SP); (SP) → An; (SP) + d → SP	–	–	–	–	–
LSL, LSR	Logical Shift	(Destination) Shifted by <count> → Destination	*	*	*	0	*
MOVE	Move Data from Source to Destination	(Source) → Destination	–	*	*	0	0
MOVE to CCR	Move to Condition Codes	(Source) → CCR	*	*	*	*	*
MOVE from CCR	Move from Condition Codes	(CCR) → Destination	–	–	–	–	–
MOVE to SR	Move to the Status Register	(Source) → SR	*	*	*	*	*

Λ logical AND
V logical OR
⊕ logical exclusive OR
~ logical complement

* affected
– unaffected
0 cleared
1 set
U undefined

1-352/1

Table 7-2. Instruction Set (Sheet 2 of 2)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
MOVE from SR	Move from the Status Register	(SR) → Destination	–	–	–	–	–
MOVE USP	Move User Stack Pointer	(USP) → An; (An) → USP	–	–	–	–	–
MOVEA	Move Address	(Source) → Destination	–	–	–	–	–
MOVEC	Move Control Register	(Cr) → Rn; (Rn) → Cr	–	–	–	–	–
MOVEM	Move Multiple Registers	(Registers) → Destination (Source) → Registers	–	–	–	–	–
MOVEP	Move Peripheral Data	(Source) → Destination	–	–	–	–	–
MOVEQ	Move Quick	Immediate Data → Destination	–	*	*	0	0
MOVES	Move Alternate Address Space	(Dn) → Destination; (Source) → Dn	–	–	–	–	–
MULS	Signed Multiply	(Destination) X (Source) → Destination	–	*	*	0	0
MULU	Unsigned Multiply	(Destination) X (Source) → Destination	–	*	*	0	0
NBCD	Negate Decimal with Extend	$0-(\text{Destination})_{10}-X \rightarrow$ Destination	*	U	*	U	*
NEG	Negate	0 – (Destination) → Destination	*	*	*	*	*
NEGX	Negate with Extend	0 – (Destination) – X → Destination	*	*	*	*	*
NOP	No Operation	–	–	–	–	–	–
NOT	Logical Complement	~(Destination) → Destination	–	*	*	0	0
OR	Inclusive OR Logical	(Destination) v (Source) → Destination	–	*	*	0	0
ORI	Inclusive OR Immediate	(Destination) v Immediate Data → Destination	–	*	*	0	0
ORI to CCR	Inclusive OR Immediate to Condition Codes	(Source) v CCR → CCR	*	*	*	*	*
ORI to SR	Inclusive OR Immediate to Status Register	(Source) v SR → SR	*	*	*	*	*
PEA	Push Effective Address	<ea> → – (SP)	–	–	–	–	–
RESET	Reset External Device	–	–	–	–	–	–
ROL, ROR	Rotate (Without Extend)	(Destination) Rotated by <count> → Destination	–	*	*	0	*
ROXL, ROXR	Rotate with Extend	(Destination) Rotated by <count> → Destination	*	*	*	0	*
RTD	Return and Deallocate Stack	(SP) + → PC; (SP) + d → SP	–	–	–	–	–
RTE	Return from Exception	(SP) + → SR; (SP) + → PC	*	*	*	*	*
RTR	Return and Restore Condition Codes	(SP) + → CC; (SP) + → PC	*	*	*	*	*
RTS	Return from Subroutine	(SP) + → PC	–	–	–	–	–
SBCD	Subtract Decimal with Extend	$(\text{Destination})_{10}-(\text{Source})_{10}-X \rightarrow$ Destination	*	U	*	U	*
S_{CC}	Set According to Condition	If CC then 1's → Destination else 0's → Destination	–	–	–	–	–
STOP	Load Status Register and Stop	Immediate Data → SR; STOP	*	*	*	*	*
SUB	Subtract Binary	(Destination) – (Source) → Destination	*	*	*	*	*
SUBA	Subtract Address	(Destination) – (Source) → Destination	–	–	–	–	–
SUBI	Subtract Immediate	(Destination) – Immediate Data → Destination	*	*	*	*	*
SUBQ	Subtract Quick	(Destination) – Immediate Data → Destination	*	*	*	*	*
SUBX	Subtract with Extend	(Destination) – (Source) – X → Destination	*	*	*	*	*
SWAP	Swap Register Halves	Register [31:16] ↔ Register [15:0]	–	*	*	0	0
TAS	Test and Set an Operand	(Destination) Tested → CC; 1 → [7] OF Destination	–	*	*	0	0
TRAP	Trap	(PC) → – (SSP); (SR) → – (SSP); (Vector) → PC	–	–	–	–	–
TRAPV	Trap on Overflow	If V set then TRAP	–	–	–	–	–
TST	Test and Operand	(Destination) Tested → CC	–	*	*	0	0
UNLK	Unlink	(An) → SP; (SP) + → An	–	–	–	–	–

[ ] = bit number
Λ logical AND
V logical OR
⊕ logical exclusive OR
~ logical complement

* affected
– unaffected
0 cleared
1 set
U undefined

1-352/2

7.1.2 Instruction Prefetch

The MC68010 uses a two-word tightly-coupled instruction prefetch mechanism to enhance performance. This mechanism is described in terms of the microcode operations involved. If the execution of an instruction is defined to begin when the microroutine for that instruction is entered, some features of the prefetch mechanism can be described.

1. When execution of an instruction begins, the operation word and the word following have already been fetched. The operation word is in the instruction decoder.
2. In the case of multi-word instructions, as each additional word of the instruction is used internally, a fetch is made to the instruction stream to replace it.
3. The last fetch for an instruction from the instruction stream is made when the operation word is discarded and decoding is started on the next instruction.
4. If the instruction is a single-word instruction causing a branch, the second word is not used. But because this word is fetched by the preceding instruction, it is impossible to avoid this superfluous fetch.
5. In the case of an interrupt or trace exception, both prefetched words are not used.
6. The program counter usually points to the last word fetched from the instruction stream.

7.1.3 Loop Mode Operation

The MC68010 has several features that provide efficient execution of program loops. One of these features is the DBcc looping primitive instruction. The DBcc instruction operates on three operands, a loop counter, a branch condition, and a branch displacement. When the DBcc is executed in loop mode, the contents of the low order word of the register specified as the loop counter is decremented by one and compared to minus one. If equal to minus one, the result of the decrement is placed back into the count register and the next sequential instruction is executed, otherwise the condition code register is checked against the specified branch condition. If the condition is true, the result of the decrement is discarded and the next sequential instruction is executed. Finally, if the count register is not equal to minus one and the branch condition is false, the branch displacement is added to the program counter and instruction execution continues at that new address. Note that this is slightly different than non-looped execution; however, the results are the same.

An example of using the DBcc instruction in a simple loop for moving a block of data is shown in Figure 7-1. In this program, the block of data 'LENGTH' words long at address 'SOURCE' is to be moved to address 'DEST' provided that none of the words moved are equal to zero. When the effect of instruction prefetch on this loop is examined it can be seen that the bus activity during the loop execution would be:

1. Fetch the MOVE.W instruction,
2. Fetch the DBEQ instruction,
3. Read the operand where A0 points,
4. Write the operand where A1 points,
5. Fetch the DBEQ branch displacement, and
6. If loop conditions are met, return to step 1.

```
        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)+, (A1)+    Loop To Move The Block Of Data
        DBEQ    D0, LOOP        Stop If Data Word Is Zero
```

1-353

Figure 7-1. DBcc Loop Program Example

During this loop, five bus cycles are executed; however, only two bus cycles perform the data movement. Since the MC68010 has a two word prefetch queue in addition to a one word instruction decode register, it is evident that the three instruction fetches in this loop could be eliminated by placing the MOVE.W word in the instruction decode register and holding the DBEQ instruction and its branch displacement in the prefetch queue. The MC68010 has the ability to do this by entering the loop mode of operation. During loop mode operation, all opcode fetches are suppressed and only operand reads and writes are performed until an exit loop condition is met.

Loop mode operation is transparent to the programmer, with only two conditions required for the MC68010 to enter the loop mode. First, a DBcc instruction must be executed with both branch conditions met and a branch displacement of minus four; which indicates that the branch is to a one word instruction preceding the DBcc instruction. Second, when the processor fetches the instruction at the branch address, it is checked to determine whether it is one of the allowed looping instructions. If it is, the loop mode is entered. Thus, the single word looped instruction and the first word of the DBcc instruction will each be fetched twice when the loop is entered; but no instruction fetches will occur again until the DBcc loop conditions fail.

In addition to the normal termination conditions for a loop, there are several conditions that will cause the MC68010 to exit loop mode operation. These conditions are interrupts, trace exceptions, reset errors, and bus errors. Interrupts are honored after each execution of the DBcc instruction, but not after the execution of the looped instruction. If an interrupt exception occurs, loop mode operation is terminated and can be restarted on return from the interrupt handler. If the T bit is set, trace exceptions will occur at the end of both the loop instruction and the DBcc instruction and thus loop mode operation is not available. Reset will abort all processing, including the loop mode. Bus errors during the loop mode will be treated the same as in normal processing; however, when the RTE instruction is used to continue the execution of the looped instruction, the three word loop will not be re-fetched.

The loopable instructions available on the MC68010 are listed in Table 7-3. These instructions may use the three address register indirect addressing modes to form one word looping instructions; (An), (An)+, and −(An).

Table 7-3. MC68010 Loopable Instructions

Opcodes	Applicable Addressing Modes	
MOVE [BWL]	(Ay) to (Ax)	−(Ay) to (Ax)
	(Ay) to (Ax)+	−(Ay) to (Ax)+
	(Ay) to −(Ax)	−(Ay) to −(Ax)
	(Ay)+ to (Ax)	Ry to (Ax)
	(Ay)+ to (Ax)+	Ry to (Ax)+
	(Ay)+ to −(Ax)	
ADD [BWL] AND [BWL] CMP [BWL] OR [BWL] SUB [BWL]	(Ay) to Dx (Ay)+ to Dx −(Ay) to Dx	
ADDA [WL] CMPA [WL] SUBA [WL]	(Ay) to Ax −(Ay) to Ax (Ay)+ to Ax	
ADD [BWL] AND [BWL] EOR [BWL] OR [BWL] SUB [BWL]	Dx to (Ay) Dx to (Ay)+ Dx to −(Ay)	

Opcodes	Applicable Addressing Modes
ABCD [B] ADDX [BWL] SBCD [B] SUBX [BWL]	−(Ay) to −(Ax)
CMP [BWL]	(Ay)+ to (Ax)+
CLR [BWL] NEG [BWL] NEGX [BWL] NOT [BWL] TST [BWL] NBCD [B]	(Ay) (Ay)+ −(Ay)
ASL [W] ASR [W] LSL [W] LSR [W] ROL [W] ROR [W] ROXL [W] ROXR [W]	(Ay) by #1 (Ay)+ by #1 −(Ay) by #1

NOTE
[B, W, or L] indicate an operand size of byte, word, or long word.

1-354

7.2 INSTRUCTION EXECUTION TIMES

The following paragraphs contain listings of the instruction execution times in terms of external clock (CLK) periods. In this timing data, it is assumed that both memory read and write cycle times are four clock periods. Any wait states caused by a longer memory cycle must be added to the total instruction time. The number of bus read and write cycles for each instruction is also included with the timing data. This data is enclosed in parenthesis following the execution periods and is shown as (r/w) where r is the number of read cycles and w is the number of write cycles.

NOTE

The number of clock periods includes instruction fetches and all applicable operand fetches and stores.

7.2.1 Operand Effective Address Calculation Times

Table 7-4 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand if necessary. Several instructions do not need the operand at an effective address to be fetched and thus require fewer clock periods to calculate a given effective address than the instructions that do fetch the effective address operand. The number of bus read and write cycles is shown in parentheses as (r/w). Note there are no write cycles involved in processing the effective address.

Table 7-4. 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(An)	Address Register Indirect with Displacement	**8**(2/0)	**4**(0/0)	**12**(3/0)	**4**(1/0)
d(An, ix)*	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(PC)	Program Counter with Displacement	**8**(2/0)	–	**12**(3/0)	–
d(PC, ix)	Program Counter with Index	**10**(2/0)	–	**14**(3/0)	–
#xxx	Immediate	**4**(1/0)	–	**8**(2/0)	–

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

1-355

7.2.2 Move Instruction Execution Times

Tables 7-5, 7-6, 7-7, and 7-8 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 7-5. Move Byte and Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
(An)	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
(An)+	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
−(An)	10(2/0)	10(2/0)	14(2/1)	14(2/1)	14(2/1)	18(3/1)	20(3/1)	18(3/1)	22(4/1)
d(An)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d(An, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
xxx.W	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
xxx.L	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	24(5/1)	26(5/1)	24(5/1)	28(6/1)
d(PC)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d(PC, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
#xxx	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)

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

1-356

Table 7-6. Move Byte and Word Instruction Loop Mode Execution Times

	Loop Continued			Loop Terminated					
	Valid Count, cc False			Valid Count, cc True			Expired Count		
	Destination								
Source	(An)	(An)+	−(An)	(An)	(An)+	−(An)	(An)	(An)+	−(An)
Dn	10(0/1)	10(0/1)	—	18(2/1)	18(2/1)	—	16(2/1)	16(2/1)	—
An*	10(0/1)	10(0/1)	—	18(2/1)	18(2/1)	—	16(2/1)	16(2/1)	—
(An)	14(1/1)	14(1/1)	16(1/1)	20(3/1)	20(3/1)	22(3/1)	18(3/1)	18(3/1)	20(3/1)
(An)+	14(1/1)	14(1/1)	16(1/1)	20(3/1)	20(3/1)	22(3/1)	18(3/1)	18(3/1)	20(3/1)
−(An)	16(1/1)	16(1/1)	18(1/1)	22(3/1)	22(3/1)	24(3/1)	20(3/1)	20(3/1)	22(3/1)

* Word only.

1-357

Table 7-7. Move Long Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	14(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)	14(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(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(An, ix)*	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, ix)*	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	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 (ix) does not affect execution time.

1-358

Table 7-8. Move Long Instruction Loop Mode Execution Times

	Loop Continued			Loop Terminated					
	Valid Count, cc False			Valid Count, cc True			Expired Count		
	Destination								
Source	(An)	(An)+	−(An)	(An)	(An)+	−(An)	(An)	(An)+	−(An)
Dn	14(0/2)	14(0/2)	—	20(2/2)	20(2/2)	—	18(2/2)	18(2/2)	—
An	14(0/2)	14(0/2)	—	20(2/2)	20(2/2)	—	18(2/2)	18(2/2)	—
(An)	22(2/2)	22(2/2)	24(2/2)	28(4/2)	28(4/2)	30(4/2)	24(4/2)	24(4/2)	26(4/2)
(An)+	22(2/2)	22(2/2)	24(2/2)	28(4/2)	28(4/2)	30(4/2)	24(4/2)	24(4/2)	26(4/2)
−(An)	24(2/2)	24(2/2)	26(2/2)	30(4/2)	30(4/2)	32(4/2)	26(4/2)	26(4/2)	28(4/2)

1-359

7.2.3 Standard Instruction Execution Times

The number of clock periods shown in Tables 7-9 and 7-10 indicate the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Tables 7-9 and 7-10 the headings have the following meanings: An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand.

Table 7-9. Standard Instruction Execution Times

Instruction	Size	op<ea>, An***	op<ea>, Dn	op Dn, <M>
ADD	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	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	–	–	**42**(1/0)+*	–
MULU	–	–	**40**(1/0)+	–
OR	Byte, Word	–	**4**(1/0)+	**8**(1/1)+
	Long	–	**6**(1/0)+	**12**(1/2)+
SUB	Byte, Word	**8**(1/0)+	**4**(1/0)+	**8**(1/1)+
	Long	**6**(1/0)+	**6**(1/0)+	**12**(1/2)+

NOTES:

\+ add effective address calculation time
* indicates maximum value
** only available addressing mode is data register direct
*** word or long only

1-360

Table 7-10. 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 only.
<ea> may be (An), +(An), or −(An) only. Add two clock periods to the table value if <ea> is −(An).

1-361

7.2.4 Immediate Instruction Execution Times

The number of clock periods shown in Table 7-11 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 7-11, the headings have the following meanings: # = immediate operand, Dn = data register operand, AN = address register operand, and M = memory operand.

Table 7-11. Immediate Instruction Execution Times

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte, Word	**8**(2/0)	—	**12**(2/1) +
	Long	**14**(3/0)	—	**20**(3/2) +
ADDQ	Byte, Word	**4**(1/0)	**4**(1/0)*	**8**(1/1) +
	Long	**8**(1/0)	**8**(1/0)	**12**(1/2) +
ANDI	Byte, Word	**8**(2/0)	—	**12**(2/1) +
	Long	**14**(3/0)	—	**20**(3/1) +
CMPI	Byte, Word	**8**(2/0)	—	**8**(2/0) +
	Long	**12**(3/0)	—	**12**(3/0) +
EORI	Byte, Word	**8**(2/0)	—	**12**(2/1) +
	Long	**14**(3/0)	—	**20**(3/2) +
MOVEQ	Long	**4**(1/0)	—	—
ORI	Byte, Word	**8**(2/0)	—	**12**(2/1) +
	Long	**14**(3/0)	—	**20**(3/2) +
SUBI	Byte, Word	**8**(2/0)	—	**12**(2/1) +
	Long	**14**(3/0)	—	**20**(3/2) +
SUBQ	Byte, Word	**4**(1/0)	**4**(1/0)*	**8**(1/1) +
	Long	**8**(1/0)	**8**(1/0)	**12**(1/2) +

+ add effective address calculation time.
* word only

1-362

7.2.5 Single Operand Instruction Execution Times

Tables 7-12, 7-13, and 7-14 indicate the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7-12. Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
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	4(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
* Use non-fetching effective address calculation time.

Table 7-13. Clear Instruction Execution Times

	Size	Dn	An	(An)	(An)+	−(An)	d(An)	d(An, ix)*	xxx.W	xxx.L
CLR	Byte, Word	4(1/0)	—	8(1/1)	8(1/1)	10(1/1)	12(2/1)	16(2/1)	12(2/1)	16(3/1)
	Long	6(1/0)	—	12(1/2)	12(1/2)	14(1/2)	16(2/2)	20(2/2)	16(2/2)	20(3/2)

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

Table 7-14. 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)

7.2.6 Shift/Rotate Instruction Execution Times

Tables 7-15 and 7-16 indicate the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7-15. 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

1-366

Table 7-16. Shift/Rotate 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)
ASR, ASL	Word	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)
LSR, LSL	Word	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)
ROR, ROL	Word	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)
ROXR, ROXL	Word	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)

1-587

7.2.7 Bit Manipulation Instruction Execution Times

Table 7-17 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7-17. Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	—	8(1/1) +	—	12(2/1) +
	Long	8(1/0)*	—	12(2/0)*	—
BCLR	Byte	—	10(1/1) +	—	14(2/1) +
	Long	10(1/0)*	—	14(2/0)*	—
BSET	Byte	—	8(1/1) +	—	12(2/1) +
	Long	8(1/0)*	—	12(2/0)*	—
BTST	Byte	—	4(1/0) +	—	8(2/0) +
	Long	6(1/0)	—	10(2/0)	—

+ add effective address calculation time
* indicates maximum value

1-367

7.2.8 Conditional Instruction Execution Times

Table 7-18 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7-18. Conditional Instruction Execution Times

Instruction	Displacement	Branch Taken	Branch Not Taken
B_{CC}	Byte	**10**(2/0)	**6**(1/0)
	Word	**10**(2/0)	**10**(2/0)
BRA	Byte	**10**(2/0)	—
	Word	**10**(2/0)	—
BSR	Byte	**18**(2/2)	—
	Word	**18**(2/2)	—
DB_{CC}	CC true	—	**10**(2/0)
	CC false	**10**(2/0)	**16**(3/0)

1-368

7.2.9 JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Table 7-19 indicates the number of clock periods required for the jump, jump-to-subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 7-19. JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Instr	Size	(An)	(An)+	−(An)	d(An)	d(An, ix)+	xxx.W	xxx.L	d(PC)	d(PC, ix)*
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** (4+n/0)	**18+4n** (4+n/0)
	Long	**12+8n** (3+2n/0)	**12+8n** (3+2n/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 (ix) does not affect the instruction's execution time

1-369

7.2.10 Multi-Precision Instruction Execution Times

Table 7-20 indicates the number of clock periods for the multi-precision 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 number of read and write cycles is shown in parenthesis as (r/w).

In Table 7-20, the headings have the following meanings: Dn = data register operand and M = memory operand.

Table 7-20. Multi-Precision Instruction Execution Times

				Loop Mode		
				Continued	Terminated	
		Non-Looped		Valid Count, cc False	Valid Count, cc True	Expired Count
Instruction	**Size**	**op Dn, Dn**	**op M, M***			
ADDX	Byte, Word	**4**(1/0)	**18**(3/10)	**22**(2/1)	**28**(4/1)	**26**(4/1)
	Long	**6**(1/0)	**30**(5/2)	**32**(4/2)	**38**(6/2)	**36**(6/2)
CMPM	Byte, Word	–	**12**(3/0)	**14**(2/0)	**20**(4/0)	**18**(4/0)
	Long	–	**20**(5/0)	**24**(4/0)	**30**(6/0)	**26**(6/0)
SUBX	Byte, Word	**4**(1/0)	**18**(3/1)	**22**(2/1)	**28**(4/1)	**26**(4/1)
	Long	**6**(1/0)	**30**(5/2)	**32**(4/2)	**38**(6/2)	**36**(6/2)
ABCD	Byte	**6**(1/0)	**18**(3/1)	**24**(2/1)	**30**(4/1)	**28**(4/1)
SBCD	Byte	**6**(1/0)	**18**(3/1)	**24**(2/1)	**30**(4/1)	**28**(4/1)

*Source and destination ea is (An)+ for CMPM and –(An) for all others.

7.2.11 Miscellaneous Instruction Execution Times

Table 7-21 indicates the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycle is shown in parenthesis as (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

Table 7-21. 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)
MOVES	Byte, Word	–	–	**16**(2/1) + *	**16**(3/0) + *
	Long	–	–	**20**(2/2) + *	**20**(4/0) + *
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**(27/0)	–	–	–
	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 non-fetching effective address calculation time.
** Source or destination is a memory location for the MOVEP and MOVES instructions and a control register for the MOVEC instruction.

1-371

7.2.12 Exception Processing Execution Times

Table 7-22 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first two instruction words of the handler routine. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 7-22. Exception Processing Execution Times

Exception	
Address Error	**126**(4/26)
Breakpoint Instruction*	**42**(5/4)
Bus Error	**126**(4/26)
CHK Instruction**	**44**(5/4)+
Divide By Zero	**42**(5/4)
Illegal Instruction	**38**(4/4)
Interrupt*	**46**(5/4)
MOVEC, Illegal Cr**	**46**(5/4)
Privilege Violation	**38**(4/4)
Reset***	**40**(6/0)
RTE, Illegal Format	**50**(7/4)
RTE, Illegal Revision	**70**(12/4)
Trace	**38**(4/4)
TRAP Instruction	**38**(4/4)
TRAPV Instruction	**40**(5/4)

\+ add effective address calculation time.

* The interrupt acknowledge and breakpoint cycles are assumed to take four clock periods.

** Indicates maximum value.

*** Indicates the time from when $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are first sampled as negated to when instruction execution starts.

1-372

SECTION 8
ELECTRICAL SPECIFICATIONS

This section contains electrical specifications and associated timing information for the MC68010 and MC68012.

8.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	−0.3 to +7.0	V
Input Voltage	V_{in}	−0.3 to +7.0	V
Operating Temperature Range MC68010/MC68012 MC68010C/MC68012C	T_A	T_L to T_H 0 to 70 −40 to 85	°C
Storage Temperature	T_{stg}	−55 to 150	°C

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either GND or V_{CC}).

8.2 THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Symbol	Value	Rating
Thermal Resistance (Still Air)	θ_{JA}		θ_{JC}		°C/W
Ceramic, Type L/LC		30		15*	
Ceramic, Type R/RC		33		15	
Plastic, Type P		30		15*	
Plastic, Type FN		45*		25*	

* Estimated

8.3 POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \bullet \theta_{JA}) \quad (1)$$

Where:

T_A = Ambient Temperature, °C

θ_{JA} = Package Thermal Resistance, Junction-to-Ambient, °C/W

$P_D = P_{INT} + P_{I/O}$

$P_{INT} = I_{CC} \times V_{CC}$, Watts — Chip Internal Power

$P_{I/O}$ = Power Dissipation on Input and Output Pins — User Determined

For most applications $P_{I/O} < P_{INT}$ and can be neglected.

An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

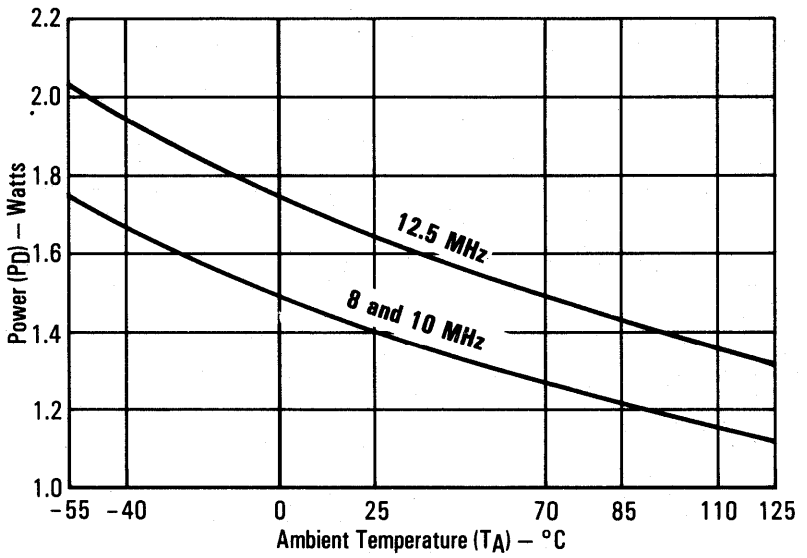
$$P_D = K \div (T_J + 273°C) \quad (2)$$

Solving equations 1 and 2 for K gives:

$$K = T_D \bullet (T_A + 273°C) + \theta_{JA} \bullet P_D^2 \quad (3)$$

Where K is a constant pertaining to the particular part. K can be determined from equation 3 by measuring P_D (at equilibrium) for a known T_A. Using this value of K the values of P_D and T_J can be obtained by solving equations (1) and (2) iteratively for any value of T_A.

The curve shown in Figure 8-1 gives the graphic solution to these equations for the specification power dissipation of 1.50 and 1.75 watts over the ambient temperature range of −55°C to 125°C using a θ_{JA} of 45°C/W for the ceramic (L suffix) package.



1-373

Figure 8-1. MC68010 Power Dissipation (P_D) vs Ambient Temperature (T_A)

The total thermal resistance of a package (θ_{JA}) can be separated into two components, θ_{JC} and θ_{CA}, representing the barrier to heat flow from the semiconductor junction to the package (case) surface (θ_{JC}) and from the case to the outside ambient (θ_{CA}). These terms are related by the equation:

$$\theta_{JA} = \theta_{JC} + \theta_{CA} \quad (4)$$

θ_{JC} is device related and cannot be influenced by the user. However, θ_{CA} is user dependent and can be minimized by such thermal management techniques as heat sinks, ambient air cooling and thermal convention. Thus, good thermal management on the part of the user can significantly reduce θ_{CA} so that θ_{JA} approximately equals θ_{JC}. Substitution of θ_{JC} for θ_{JA} in equation (1) will result in a lower semiconductor junction temperature.

Values for thermal resistance presented in this data sheet, unless estimated, were derived using the procedure described in Motorola Reliability Report 7843, "Thermal Resistance Measurement Method for MC68XX Microcomponent Devices," and are provided for design purposes only. Thermal measurements are complex and dependent on procedure and setup. User derived values for thermal resistance may differ.

Table 8-1. Maximum Power Dissipation by Package Type Modes

Package Type	Temperature (°C)	Maximum Power Dissipation (Watts) per Frequency (MHz)		
		8 MHz	10 MHz	12.5 MHz
L	0 to 70	1.50	1.50	1.75
	−40 to 85	1.65	1.65	—
LC	0 to 70	1.50	1.50	1.75
	−40 to 85	1.65	1.65	—
P	0 to 70	1.50	1.50	—
	−40 to 85	1.65	1.65	—
R	0 to 70	1.50	1.50	1.75
	−40 to 85	1.65	1.65	—
RC	0 to 70	1.50	1.50	1.75
	−40 to 85	1.65	1.65	—
FN	0 to 70	1.50	1.50	—
ILC	0 to 70	1.50	1.50	—
	−40 to 85	1.65	1.65	—
IRC	0 to 70	1.50	1.50	—
	−40 to 85	1.65	1.65	—

8.4 DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 5.0 Vdc ±5%; GND = 0 Vdc; $T_A = T_L$ to T_H; see Figures 8-2, 8-3, and 8-4)

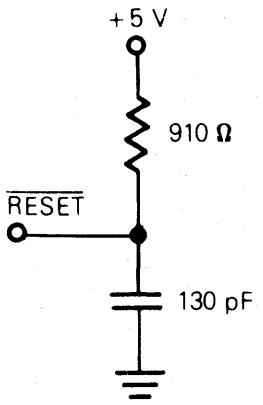
Characteristic		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	GND−0.3	0.8	V
Input Leakage Current @ 5.25 V	$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$,	I_{in}	—	2.5	μA
	CLK, $\overline{IPL0}$-$\overline{IPL2}$, $\overline{VPA}$		—	20	
	$\overline{HALT}$, $\overline{RESET}$				
Three-State (Off State) Input Current @ 2.4 V/0.4 V	$\overline{AS}$, A1-A23, D0-D15, FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	I_{TSI}	—	20	μA
Output High Voltage (I_{OH} = −400 μA)	E*	V_{OH}	V_{CC}−0.75	—	V
	E**, $\overline{AS}$, A1-A23, $\overline{BG}$, D0-D15, FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$		2.4	—	
Output Low Voltage		V_{OL}			V
(I_{OL} = 1.6 mA)	$\overline{HALT}$		—	0.5	
(I_{OL} = 3.2 mA)	A1-A23, $\overline{BG}$, FC0-FC2		—	0.5	
(I_{OL} = 5.0 mA)	$\overline{RESET}$		—	0.5	
(I_{OL} = 5.3 mA)	E, $\overline{AS}$, D0-D15, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$		—	0.5	
Power Dissipation (See Section 8.3)***		P_D	—	—	W
Capacitance (V_{in} = 0 V T_A = 25°C; Frequency = 1 MHz)****		C_{in}	—	20.0	pF

*With external pullup resistor of 1.1 kΩ.

**Without external pullup resistor.

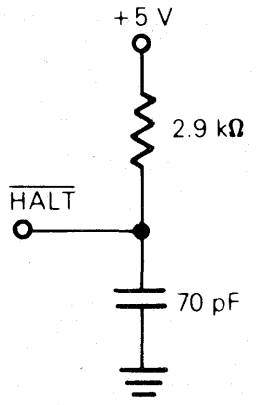
***During normal operation instantaneous V_{CC} current requirements may be as high as 1.5 A.

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



1-374

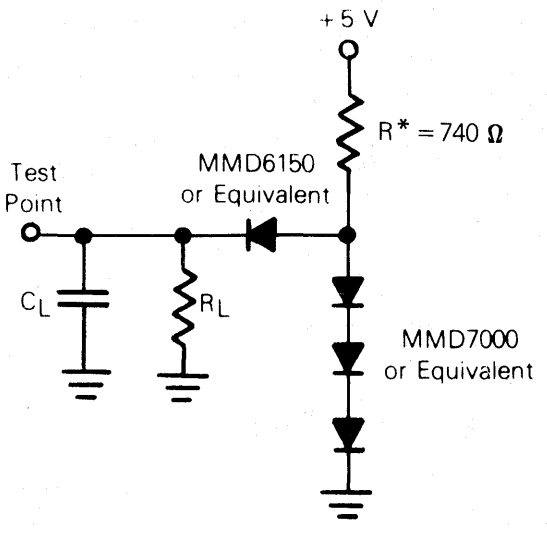
Figure 8-2. $\overline{\text{RESET}}$ Test Load



1-375

Figure 8-3. $\overline{\text{HALT}}$ Test Load

C_L = 130 pF
(Includes all Parasitics)
R_L = 6.0 kΩ for
$\overline{AS}$, A1-A23, $\overline{BG}$, D0-D15, E
FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$
* R = 1.22 kΩ for A1-A23, $\overline{BG}$,
FC0-FC2

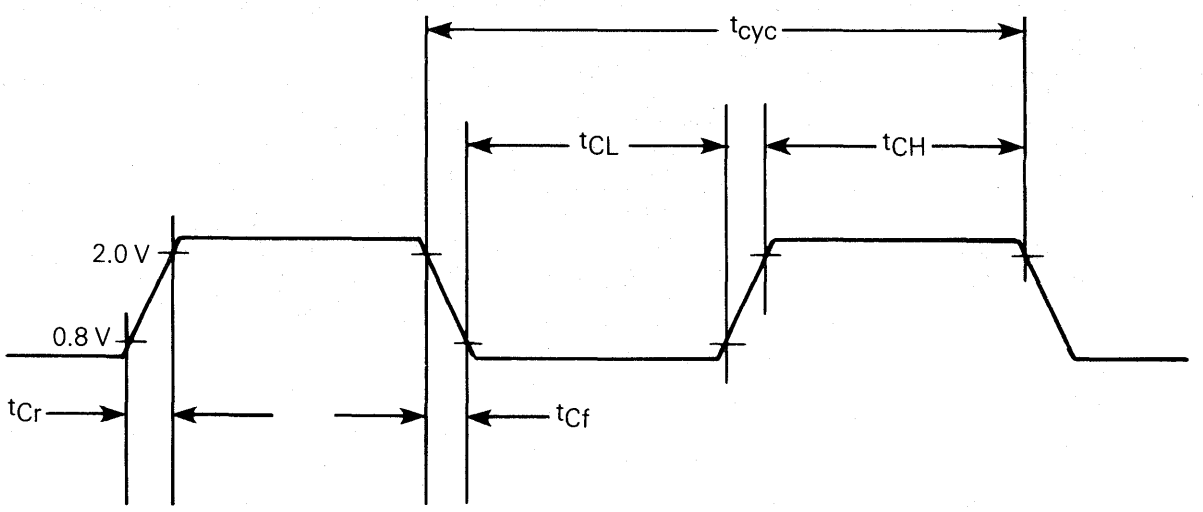


1-376

Figure 8-4. Test Loads

8.5 AC ELECTRICAL SPECIFICATIONS — CLOCK TIMING (See Figure 8-5)

Characteristic	Symbol	8 MHz		10 MHz		12.5 MHz		Unit
		Min	Max	Min	Max	Min	Max	
Frequency of Operation	f	4.0	8.0	4.0	10.0	4.0	12.5	MHz
Cycle Time	t_{cyc}	125	250	100	250	80	250	ns
Clock Pulse Width	t_{CL} t_{CH}	55 55	125 125	45 45	125 125	35 35	125 125	ns
Rise and Fall Times	t_{Cr} t_{Cf}	– –	10 10	– –	10 10	– –	5 5	ns



1-377

Figure 8-5. Clock Input Timing Diagram

8.6 AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES

(V_{CC} = 5.0 Vdc ±5%; GND = 0 Vdc; $T_A = T_L$ to T_H; see Figures 8-6 and 8-7)

Num.	Characteristic	Symbol	8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	
1	Clock Period	t_{cyc}	125	250	100	250	80	250	ns
2	Clock Width Low	t_{CL}	55	125	45	125	35	125	ns
3	Clock Width High	t_{CH}	55	125	45	125	35	125	ns
4	Clock Fall Time	t_{Cf}	–	10	–	10	–	5	ns
5	Clock Rise Time	t_{Cr}	–	10	–	10	–	5	ns
6	Clock Low to Address Valid	t_{CLAV}	–	70	–	55	–	55	ns
6A	Clock High to FC Valid	t_{CHFCV}	–	70	–	55	–	55	ns
7	Clock High to Address, Data Bus High Impedance (Maximum)	t_{CHADZ}	–	80	–	70	–	60	ns
8	Clock High to Address, FC Invalid (Minimum)	t_{CHAFI}	0	–	0	–	0	–	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low	t_{CHSL}	0	60	0	55	0	55	ns
11[2]	Address Valid to $\overline{AS}$, $\overline{DS}$ Low (Read)/ $\overline{AS}$ Low (Write)	t_{AVSL}	30	–	20	–	0	–	ns
11A[2]	FC Valid to $\overline{AS}$, $\overline{DS}$ Low (Read)/ $\overline{AS}$ Low (Write)	t_{FCVSL}	60	–	50	–	40	–	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}	–	70	–	55	–	50	ns
13[2]	$\overline{AS}$, $\overline{DS}$ High to Address/$\overline{FC}$ Invalid	t_{SHAFI}	30	–	20	–	10	–	ns
14[2]	$\overline{AS}$, $\overline{DS}$ Width Low (Read)/$\overline{AS}$ Low (Write)	t_{SL}	240	–	195	–	160	–	ns
14A[2]	$\overline{DS}$ Width Low (Write)	t_{DSL}	115	–	95	–	80	–	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width High	t_{SH}	150	–	105	–	65	–	ns
16	Clock High to Control Bus High Impedance	t_{CHCZ}	–	80	–	70	–	60	ns
17[2]	$\overline{AS}$, $\overline{DS}$ High to R/$\overline{W}$ High (Read)	t_{SHRH}	40	–	20	–	10	–	ns
18[1]	Clock High to R/$\overline{W}$ High	t_{CHRH}	0	70	0	60	0	60	ns
20[1]	Clock High to R/$\overline{W}$ Low	t_{CHRL}	–	70	–	60	–	60	ns
20A[2]	$\overline{AS}$ Low to R/$\overline{W}$ Valid (Write)	t_{ASRV}	–	20	–	20	–	20	ns
21[2]	Address Valid to R/$\overline{W}$ Low (Write)	t_{AVRL}	20	–	0	–	0	–	ns
21A[2]	FC Valid to R/$\overline{W}$ Low (Write)	t_{FCVRL}	60	–	50	–	30	–	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Low (Write)	t_{RLSL}	80	–	50	–	30	–	ns
23	Clock Low to Data Out Valid (Write)	t_{CLDO}	–	70	–	55	–	55	ns
25[2]	$\overline{AS}$, $\overline{DS}$ High to Data Out Invalid (Write)	t_{SHDOI}	30	–	20	–	15	–	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	t_{DOSL}	30	–	20	–	15	–	ns
27[5]	Data in to Clock Low (Setup Time on Read)	t_{DICL}	15	–	10	–	10	–	ns
27A	Late $\overline{BERR}$ Low to Clock Low (Setup Time)	t_{BELCL}	45	–	45	–	45	–	ns
28[2]	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High	t_{SHDAH}	0	245	0	190	0	150	ns
29	$\overline{AS}$, $\overline{DS}$ High to Data In Invalid (Hold Time on Read)	t_{SHDII}	0	–	0	–	0	–	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	t_{SHBEH}	0	–	0	–	0	–	ns
31[2,5]	$\overline{DTACK}$ Low to Data In (Setup Time)	t_{DALDI}	–	90	–	65	–	50	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	$t_{RHr,f}$	0	200	0	200	0	200	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	–	70	–	60	–	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	–	70	–	60	–	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	90 ns +3.5	1.5	80 ns +3.5	1.5	70 ns +3.5	Clk.Per
36[6]	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	90 ns +3.5	1.5	80 ns +3.5	1.5	70 ns +3.5	Clk.Per
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	90 ns +3.5	1.5	80 ns +3.5	1.5	70 ns +3.5	Clk.Per

— Continued

8.6 AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES (Continued)

(V_{CC} = 5.0 Vdc ±5%; GND = 0 Vdc; $T_A = T_L$ to T_H; see Figures 8-6 and 8-7)

Num.	Characteristic	Symbol	8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	
37A[7]	$\overline{BGACK}$ Low to $\overline{BR}$ High	t_{GALBRH}	20	1.5 Clocks	20	1.5 Clocks	20	1.5 Clocks	ns
38	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High)	t_{GLZ}	—	80	—	70	—	60	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	—	1.5	—	1.5	—	Clk. Per.
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}	—	70	—	70	—	70	ns
41	Clock Low to E Transition	t_{CLET}	—	70	—	55	—	45	ns
42	E Output Rise and Fall Time	$t_{Er, f}$	—	25	—	25	—	25	ns
43	$\overline{VMA}$ Low to E High	t_{VMLEH}	200	—	150	—	90	—	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	t_{SHVPH}	0	120	0	90	0	70	ns
45	E Low to Control Address Bus Invalid (Address Hold Time)	t_{ELCAI}	30	—	10	—	10	—	ns
46	$\overline{BGACK}$ Width	t_{GAL}	1.5	—	1.5	—	1.5	—	Clk. Per.
47[5]	Asynchronous Input Setup Time	t_{ASI}	20	—	20	—	20	—	ns
48[2,3]	$\overline{DTACK}$ Low to $\overline{BERR}$ Low	t_{DALBEL}	—	80	—	55	—	35	ns
49[8]	$\overline{AS}$, $\overline{DS}$ High to E Low	t_{SHEL}	−70	70	−55	55	−45	45	ns
50	E Width High	t_{EH}	450	—	350	—	280	—	ns
51	E Width Low	t_{EL}	700	—	550	—	440	—	ns
53	Clock High to Data Out Invalid	t_{CHDOI}	0	—	0	—	0	—	ns
54	E Low to Data Out Invalid	t_{ELDOI}	30	—	20	—	15	—	ns
55	R/$\overline{W}$ to Data Bus Driven	t_{RLDBD}	30	—	20	—	10	—	ns
56[4]	$\overline{HALT}$/$\overline{RESET}$ Pulse Width	t_{HRPW}	10	—	10	—	10	—	Clk. Per.
57	$\overline{BGACK}$ High to Control Bus Driven	t_{GABD}	1.5	—	1.5	—	1.5	—	Clk. Per
58[6,	$\overline{BG}$ High to Control Bus Driven	t_{GHBD}	1.5	—	1.5	—	1.5	—	Clk. Per
59[9]	Clock High to $\overline{RMC}$ Low	t_{CHRL}	—	70	—	60	—	55	ns *
60[9]	$\overline{RMC}$ Low to $\overline{AS}$, $\overline{DS}$ Low (Read)/$\overline{AS}$ Low (Write)	t_{RLSL}	60	—	50	—	40	—	ns
61[9]	Clock High to $\overline{RMC}$ High	t_{CHRH}	0	—	0	—	0	—	ns
62[9]	$\overline{AS}$, $\overline{DS}$ High to $\overline{RMC}$ High	t_{SHRH}	30	—	20	—	10	—	ns

NOTES:

1. For a loading capacitance of less than or equal to 50 picofarads, subtract 5 nanoseconds from the values given in these columns.
2. Actual value depends on clock period.
3. In the absence of $\overline{DTACK}$, $\overline{BERR}$ is an asynchronous input using the asynchronous input setup time (#47)
4. For power up, the MPU must be held in $\overline{RESET}$ state for 100 ms to allow stabilization of on-chip circuitry. After the system is powered up, #56 refers to the minimum pulse width required to reset the system.
5. If the asynchronous setup time (#47) requirements are satisfied, the $\overline{DTACK}$-low to data setup time (#31) and $\overline{DTACK}$-low to $\overline{BERR}$-low setup time (#48) requirements can be ignored. The data must only satisfy the data-in to clock-low setup time (#27) for the following clock cycle, $\overline{BERR}$ must only satisfy the late-$\overline{BERR}$-low to clock-low setup time (#27A) for the following clock cycle.
6. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.
7. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, BG may be reasserted.
8. The falling edge of S6 triggers both the negation of the strobes ($\overline{AS}$ and $\overline{xDS}$) and the falling edge of E. Either of these events can occur first, depending upon the loading on each signal. Specification #49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and the falling edge of the E clock.
9. MC68012 only.

Timing diagrams (Figures 8-6 and 8-7) are located on foldout pages 1 and 2 at the end of this document.

8.7 AC ELECTRICAL SPECIFICATIONS — MC68010 TO M6800 PERIPHERAL CYCLES

(V_{CC} = 5.0 Vdc ±5%; GND = 0 Vdc; $T_A = T_L$ to T_H; refer to Figures 8-8 and 8-9)

Num.	Characteristic	Symbol	8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}	—	70	—	55	—	50	ns
17[2]	$\overline{AS}$, $\overline{DS}$ High to R/$\overline{W}$ High (Read)	t_{SHRH}	40	—	20	—	10	—	ns
20[1]	Clock High to R/$\overline{W}$ Low	t_{CHRL}	—	70	—	60	—	60	ns
23	Clock Low to Data Out Valid (Write)	t_{CLDO}	—	70	—	55	—	55	ns
27	Data In to Clock Low (Setup Time on Read)	t_{DICL}	15	—	10	—	10	—	ns
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}	—	70	—	70	—	70	ns
41	Clock Low to E Transition	t_{CLET}	—	70	—	55	—	45	ns
42	E Output Rise and Fall Time	$t_{Er,f}$	—	25	—	25	—	25	ns
43	$\overline{VMA}$ Low to E High	t_{VMLEH}	200	—	150	—	90	—	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	t_{SHVPH}	0	120	0	90	0	70	ns
45	E Low to Control Address Bus (Address Hold Time)	t_{ELCAI}	30	—	10	—	10	—	ns
47	Asynchronous Input Setup Time	t_{ASI}	20	—	20	—	20	—	ns
49[3]	$\overline{AS}$, $\overline{DS}$ High to E Low	t_{SHEL}	−70	70	−55	55	−45	45	ns
50	E Width High	t_{EH}	450	—	350	—	280	—	ns
51	E Width Low	t_{EL}	700	—	550	—	440	—	ns
54	E Low to Data Out Invalid	t_{ELDOI}	30	—	20	—	15	—	ns

NOTES:

1. For a loading capacitance of less than or equal to 50 picofarads, subtract 5 nanoseconds from the values given in these columns.
2. Actual value depends on clock period.
3. The falling edge of S6 triggers both the negation of the strobes ($\overline{AS}$ and $\overline{xDS}$) and the falling edge of E. Either of these events can occur first, depending upon the loading on each signal. Specification #49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and the falling edge of the E clock.

Timing diagrams (Figures 8-8 and 8-9) are located on foldout pages 3 and 4 at the end of this document.

8.8 AC ELECTRICAL SPECIFICATIONS — BUS ARBITRATION

(V_{CC} = 5.0 Vdc ±5%; GND = 0 Vdc; $T_A = T_L$ to T_H; see Figures 8-10, 8-11, and 8-12)

Num.	Characteristic	Symbol	8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	
7	Clock High to Address, Data Bus High Impedance (Maximum)	t_{CHADZ}	—	80	—	70	—	60	ns
16	Clock High to Control Bus High Impedance	t_{CHCZ}	—	80	—	70	—	60	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	—	70	—	60	—	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	—	70	—	60	—	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	90 ns +3.5	1.5	80 ns +3.5	1.5	70 ns +3.5	Clk. Per.
36[2]	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	90 ns +3.5	1.5	80 ns +3.5	1.5	70 ns +3.5	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	90 ns +3.5	1.5	80 ns +3.5	1.5	70 ns +3.5	Clk. Per.
37A[3]	$\overline{BGACK}$ Low to $\overline{BR}$ High	t_{GALBRH}	20	1.5 Clocks	20	1.5 Clocks	20	1.5 Clocks	ns
38	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High)	t_{GLZ}	—	80	—	70	—	60	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	—	1.5	—	1.5	—	Clk. Per.
46	$\overline{BGACK}$ Width	t_{GAL}	1.5	—	1.5	—	1.5	—	Clk. Per.
47	Asynchronous Input Setup Time	t_{ASI}	20	—	20	—	20	—	ns
57[2]	$\overline{BGACK}$ High to Control Bus Driven	t_{GABD}	1.5	—	1.5	—	1.5	—	Clk. Per.
58[1,2]	$\overline{BG}$ High to Control Bus Driven	t_{GHBD}	1.5	—	1.5	—	1.5	—	Clk. Per.

NOTES:
1. The nanosecond value shown in the specification is the asynchronous input setup time (spec. #47).
2. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.
3. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, $\overline{BG}$ may be reasserted.

Timing diagrams (Figures 8-10, 8-11, and 8-12) are located on foldout pages 5, 6, and 7 at the end of this document.

SECTION 9
ORDERING INFORMATION

This section contains detailed information to be used as a guide when ordering the MC68010 and MC68012.

9.1 PACKAGE TYPES

Suffix	Package Type	Comments
L	Dual-in-Line Ceramic	Slide Braze Package Gold or Select Plate Lead Finish
LC	Dual-in-Line Ceramic	Side Braze Package Gold Lead Finish
P	Dual-in-Line Plastic	Copper Lead Frame Solder Dip Lead Finish
R	Pin Grid Array (PGA) Ceramic	Depopulated Center Pins Solder Dip Lead Finish Standoffs for Soldering
RC	Pin Grid Array (PGA) Ceramic	Depopulated Center Pins Gold Lead Finish No Standoffs
FN	Plastic Leaded Chip Carrier (Quad Pack)	Solder Dip Finish Suitable for Socketing or Surface Mounting

9.2 STANDARD ORDERING INFORMATION

Package Type	Frequency (MHz)	Temperature	Order Number
Ceramic	8.0	0°C to 70°C	MC68010L8
L Suffix	8.0	−40°C to 85°C	MC68010CL8
	10.0	0°C to 70°C	MC68010L10
	10.0	−40°C to 85°C	MC68010CL10
	12.5	0°C to 70°C	MC68010L12

Package Type	Frequency (MHz)	Temperature	Order Number
Ceramic with	8.0	0°C to 70°C	MC68010LC8
Gold Lead Finish	8.0	0°C to 85°C	MC68010ILC8
LC Suffix	8.0	−40°C to 85°C	MC68010CLC8
	10.0	0°C to 70°C	MC68010LC10
	10.0	0°C to 85°C	MC68010ILC10
	10.0	−40°C to 85°C	MC68010CLC10
	12.5	0°C to 70°C	MC68010LC12
Plastic	8.0	0°C to 70°C	MC68010P8
P Suffix	10.0	0°C to 70°C	MC68010P10
Pin Grid Array	8.0	0°C to 70°C	MC68010R8/MC68012R8
with Standoffs	8.0	−40°C to 85°C	MC68010CR8/MC68012CR8
R Suffix	10.0	0°C to 70°C	MC68010R10/MC68012R10
	10.0	−40°C to 85°C	MC68010CR10/MC68012CR10
	12.5	0°C to 70°C	MC68010R12/MC68012R12
Pin Grid Array	8.0	0°C to 70°C	MC68010RC8/MC68012RC8
with Gold Lead	8.0	0°C to 85°C	MC68010IRC8/MC68012IRC8
Finish without	8.0	−40°C to 85°C	MC68010CRC8/MC68012CRC8
Standoffs	10.0	0°C to 70°C	MC68010RC10/MC68012RC10
RC Suffix	10.0	0°C to 85°C	MC68010IRC10/MC68012IRC10
	10.0	−40°C to 85°C	MC68010CRC10/MC68012CRC10
	12.5	0°C to 70°C	MC68010RC12/MC68012RC12
Quad Pack*	8.0	0°C to 70°C	MC68010FN8
FN Suffix	10.0	0°C to 70°C	MC68010FN10

9.3 "BETTER" PROCESSING — STANDARD PRODUCT PLUS

Level IV (Suffix T)

- Available Package Types: L, LC, P, R, RC, and FN
- 100% High Temperature Functional Test at T_A Maximum
- Dynamic Burn-In at 125°C for 96 Hours at 5 Volts, or Equivalent

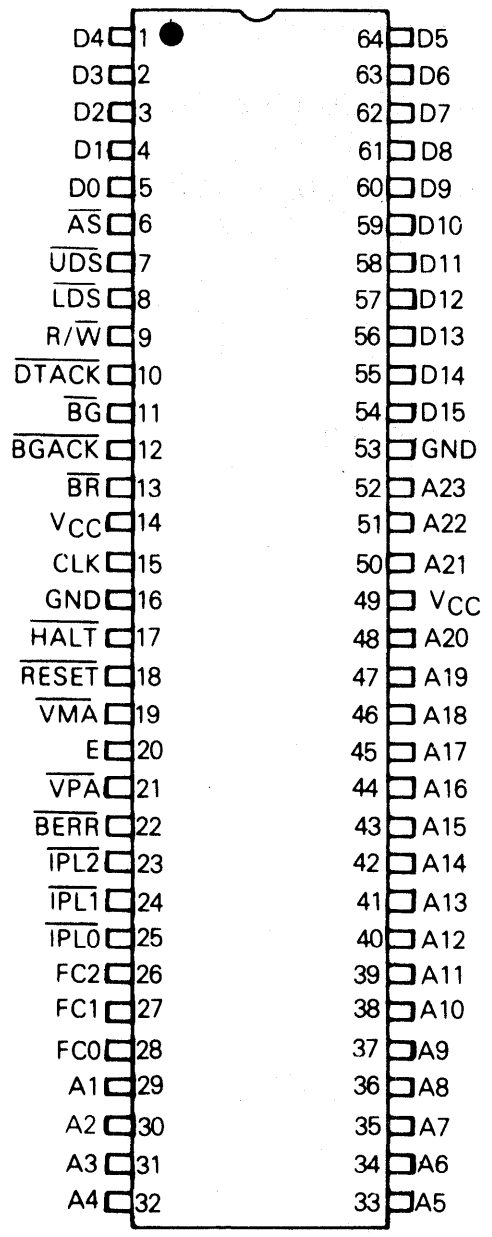
*Contact factory for factory availability.

SECTION 10
MECHANICAL DATA

This section contains the pin assignments and package dimensions for the 64-pin dual-in-line, the 68-pin grid array, and the quad pack versions of the MC68010, and the 84-pin grid array MC68012.

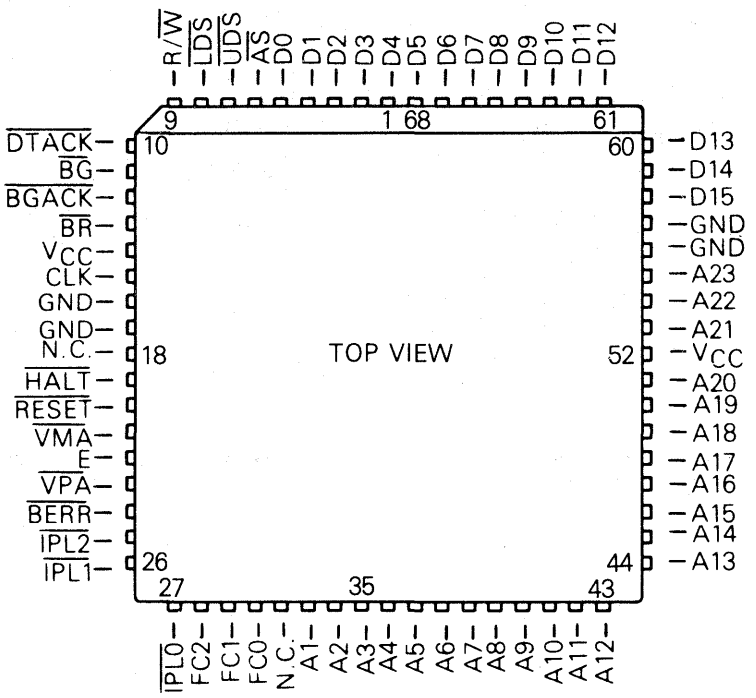
10.1 PIN ASSIGNMENTS

MC68010
64-Pin Dual-In-Line Package

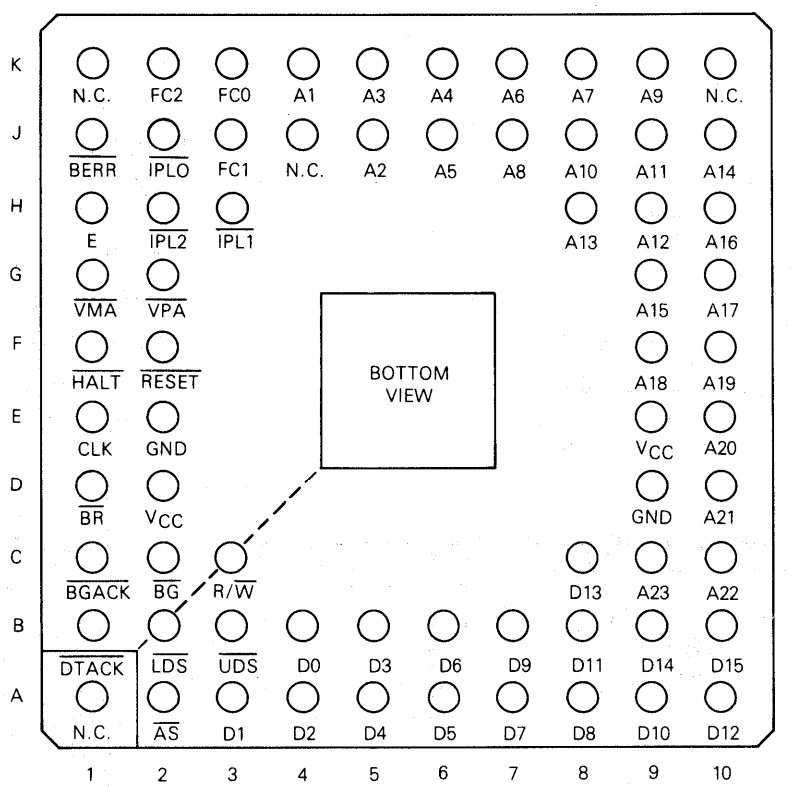


Top View

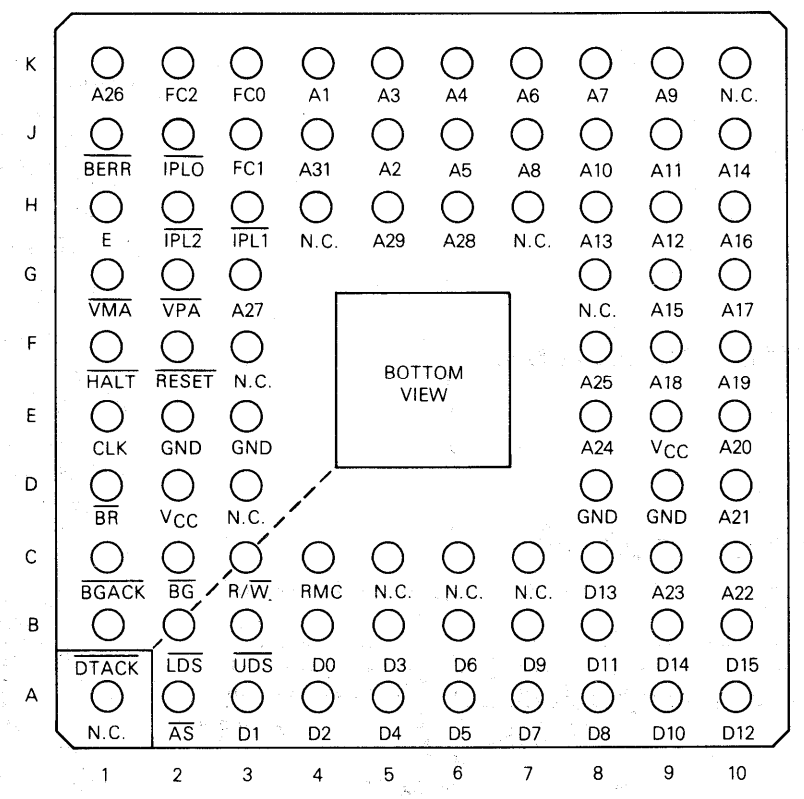
MC68010
68-Pin Quad Pack



MC68010
68-Pin Grid Array

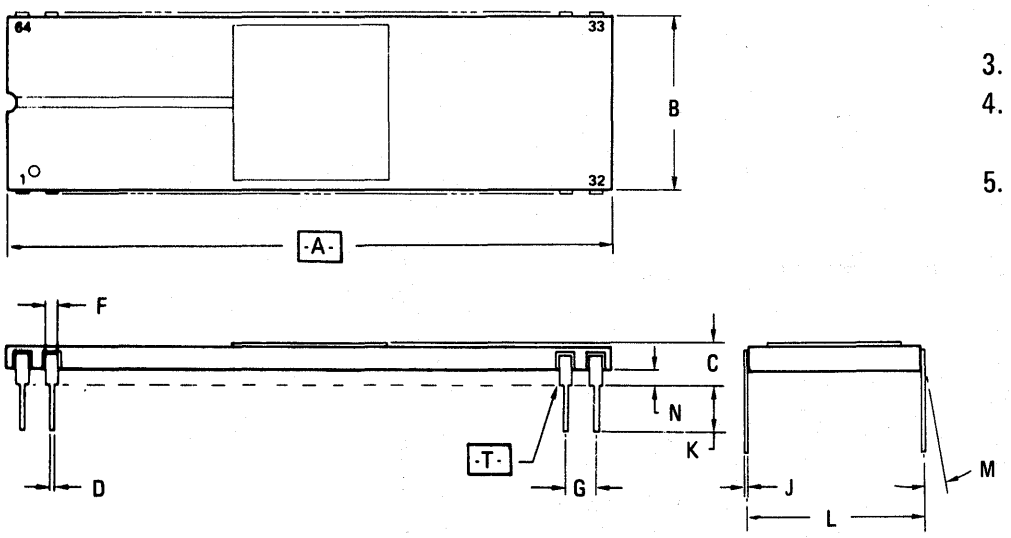


MC68012
84-Pin Grid Array



10. PACKAGE DIMENSIONS

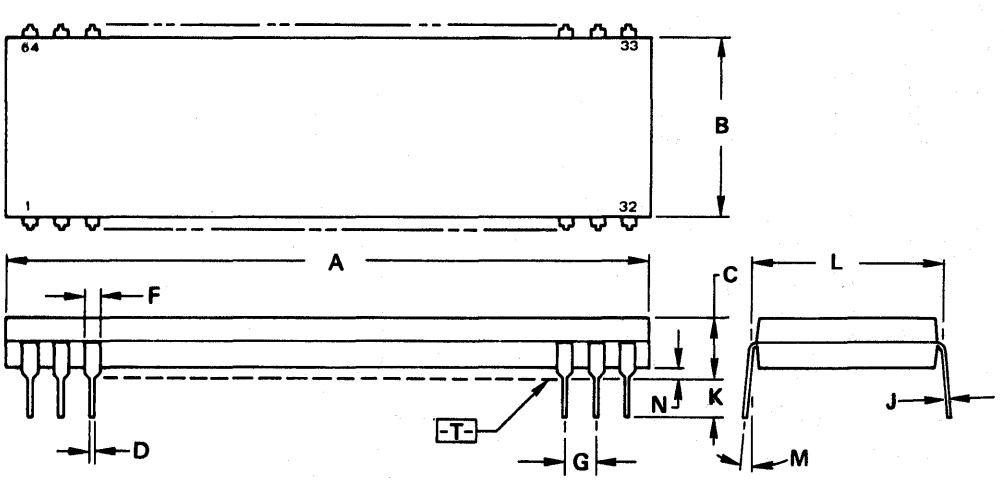
L SUFFIX
CERAMIC PACKAGE
CASE 746-01



NOTES:
1. DIMENSION -A- IS DATUM.
2. POSITIONAL TOLERANCE FOR LEADS:
   ⌖ | 0.25 (0.010) Ⓜ | T | A Ⓜ
3. -T- IS SEATING PLANE.
4. DIMENSION "L" TO CENTER OF LEADS WHEN FORMED PARALLEL.
5. DIMENSIONING AND TOLERANCING PER ANSI Y14.5, 1973.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	80.52	82.04	3.170	3.230
B	22.25	22.96	0.876	0.904
C	3.05	4.32	0.120	0.170
D	0.38	0.53	0.015	0.021
F	0.76	1.40	0.030	0.055
G	2.54 BSC		0.100 BSC	
J	0.20	0.33	0.008	0.013
K	2.54	4.19	0.100	0.165
L	22.61	23.11	0.890	0.910
M	–	10°	–	10°
N	1.02	1.52	0.040	0.060

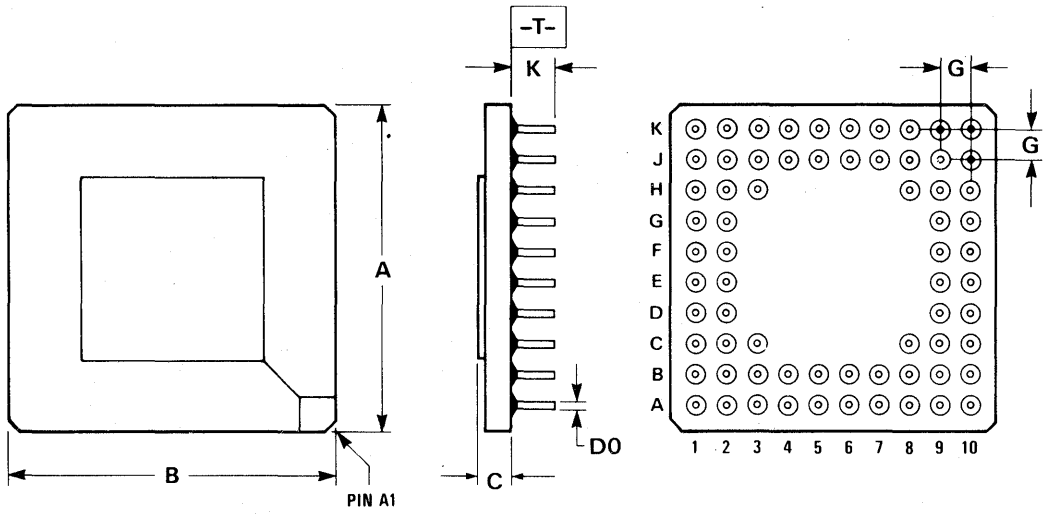
P SUFFIX
PLASTIC PACKAGE
CASE 754-01



NOTES:
1. DIMENSIONS A AND B ARE DATUMS.
2. -T- IS SEATING PLANE.
3. POSITIONAL TOLERANCE FOR LEADS (DIMENSION D):
   ⌖ | Ø 0.25 (0.010) Ⓜ | T | A Ⓜ | B Ⓜ.
4. DIMENSION L TO CENTER OF LEADS WHEN FORMED PARALLEL.
5. DIMENSION B DOES NOT INCLUDE MOLD FLASH.
6. DIMENSIONING AND TOLERANCING PER ANSI Y14.5, 1973.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	81.16	81.91	3.195	3.225
B	20.17	20.57	0.790	0.810
C	4.83	5.84	0.190	0.230
D	0.33	0.53	0.013	0.021
F	1.27	1.77	0.050	0.070
G	2.54 BSC		0.100 BSC	
J	0.20	0.38	0.008	0.015
K	3.05	3.55	0.120	0.140
L	22.86 BSC		0.900 BSC	
M	0°	15°	0°	15°
N	0.51	1.01	0.020	0.040

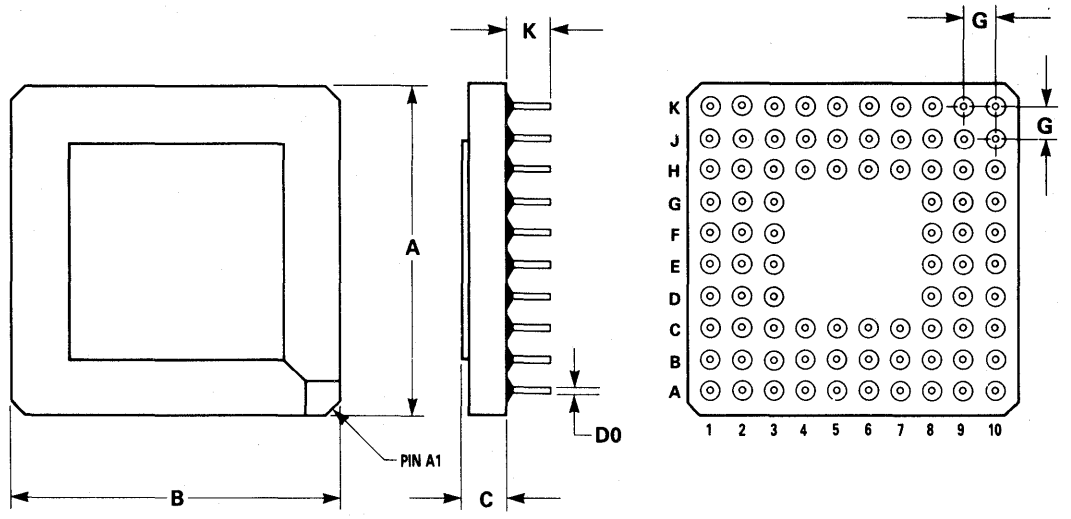
RC SUFFIX (68 PIN)
PIN GRID ARRAY
CASE 765A-03



NOTES:
1. DIMENSIONS A AND B ARE DATUMS AND T IS DATUM SURFACE.
2. POSITIONAL TOLERANCE FOR LEADS (68 PLACES)
   ⌖ | ϕ 0.13 (.005) Ⓜ | T | A Ⓢ | B Ⓢ
3. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
4. CONTROLLING DIMENSION: INCH.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	26.67	27.17	1.050	1.070
B	26.67	27.17	1.050	1.070
C	1.91	2.66	0.075	0.105
D	0.43	0.60	0.017	0.024
G	2.54 BSC		0.100 BSC	
K	4.32	4.82	0.170	0.190

RC SUFFIX (84 PIN)
PIN GRID ARRAY
CASE 793-02

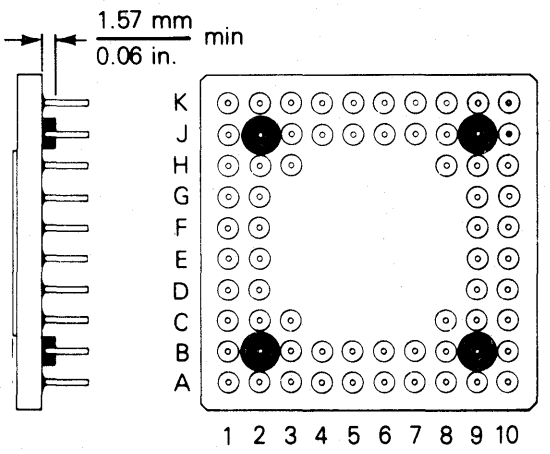


NOTES:
1. DIMENSIONS A AND B ARE DATUMS AND T IS A DATUM SURFACE.
2. POSITIONAL TOLERANCE FOR LEADS: (84 PL)
   ⌖ | ϕ 0.13 (0.005) Ⓛ | T | A Ⓢ | B Ⓢ
3. DIMENSIONING AND TOLERANCING PER Y14.5M, 1982.
4. CONTROLLING DIMENSION: INCH.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	—	27.43	—	1.080
B	—	27.43	—	1.080
C	2.03	2.67	0.080	0.105
D	0.43	0.61	0.017	0.024
G	2.54 BSC		0.100 BSC	
K	3.56	4.95	0.140	0.195

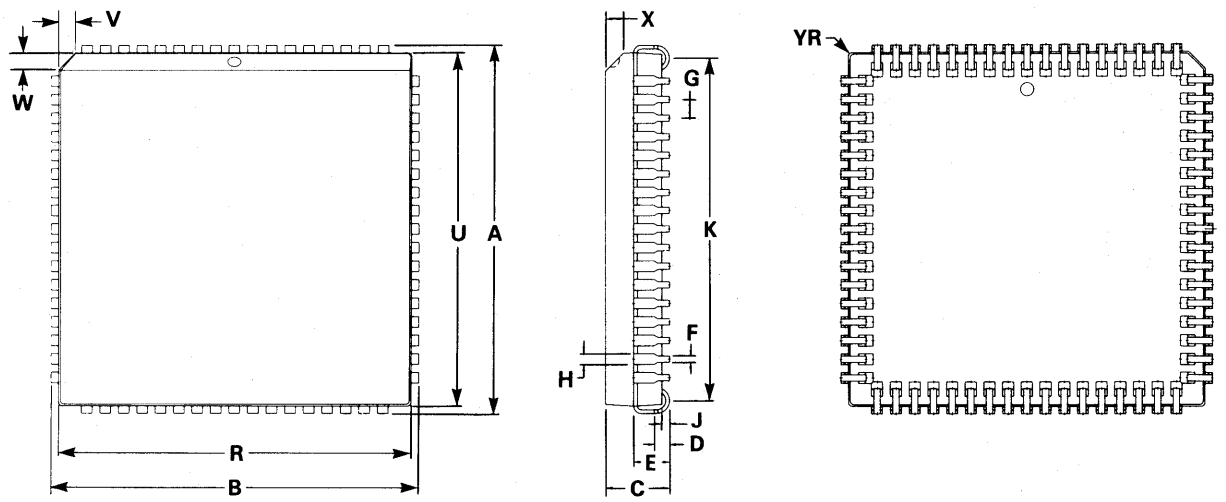
R SUFFIX
PIN GRID ARRAY
WITH STANDOFF

(DIMENSIONS ESSENTIALLY THOSE OF CASES 765A-03 AND 793-02. SEE FOLLOWING ILLUSTRATION FOR STANDOFF DETAIL.)



FN SUFFIX (68 PIN)

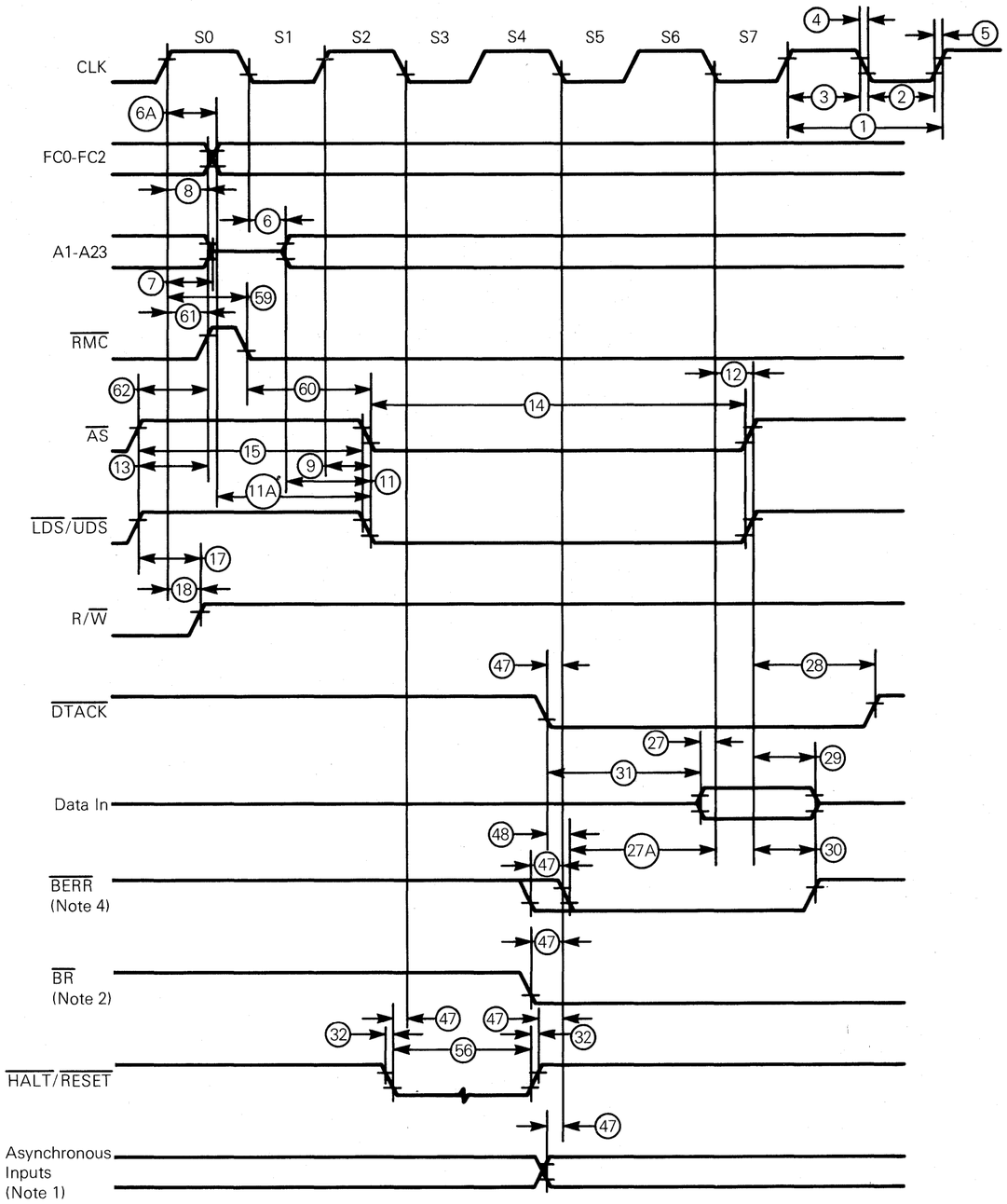
QUAD PACK
CASE 779-01



NOTES:
1. DIMENSIONS R AND U DO NOT INCLUDE MOLD FLASH.
2. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
3. CONTROLLING DIMENSION: INCH

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	25.02	25.27	0.985	0.995
B	25.02	25.27	0.985	0.995
C	4.19	4.57	0.165	0.180
D	0.64	1.01	0.025	0.040
E	2.16	2.79	0.085	0.110
F	0.33	0.53	0.013	0.021
G	1.27 BSC		0.050 BSC	
H	0.66	0.81	0.026	0.032
J	0.38	0.63	0.015	0.025
K	22.61	23.62	0.890	0.930
R	24.13	24.28	0.950	0.956
U	24.13	24.28	0.950	0.956
V	1.07	1.21	0.042	0.048
W	1.07	1.21	0.042	0.048
X	1.07	1.42	0.042	0.056
Y	0.00	0.50	0.000	0.020

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

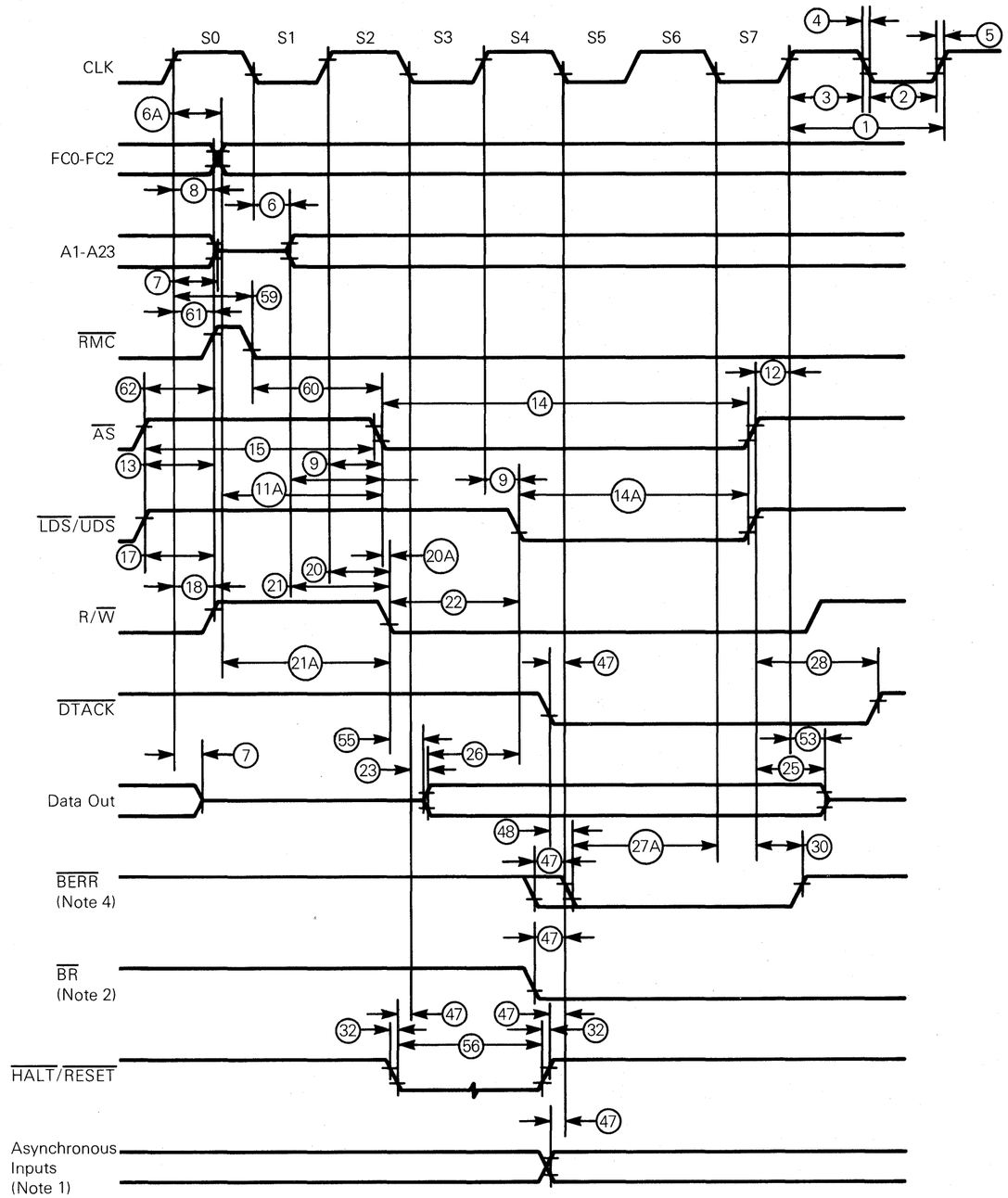


NOTES:
1. Setup time for the asynchronous inputs IPL0/2, IPL1, and VPA guarantees their recognition at the next falling edge of the clock.
2. BR need fall at this time only in order to insure being recognized at the end of this bus cycle.
3. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.
4. The timing for the first falling edge (47) of BERR are for BERR without DTACK, the timings for the second falling edge (27A and 48) are for BERR with DTACK.

1-701

Figure 8-6. Read Cycle Timing Diagram

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

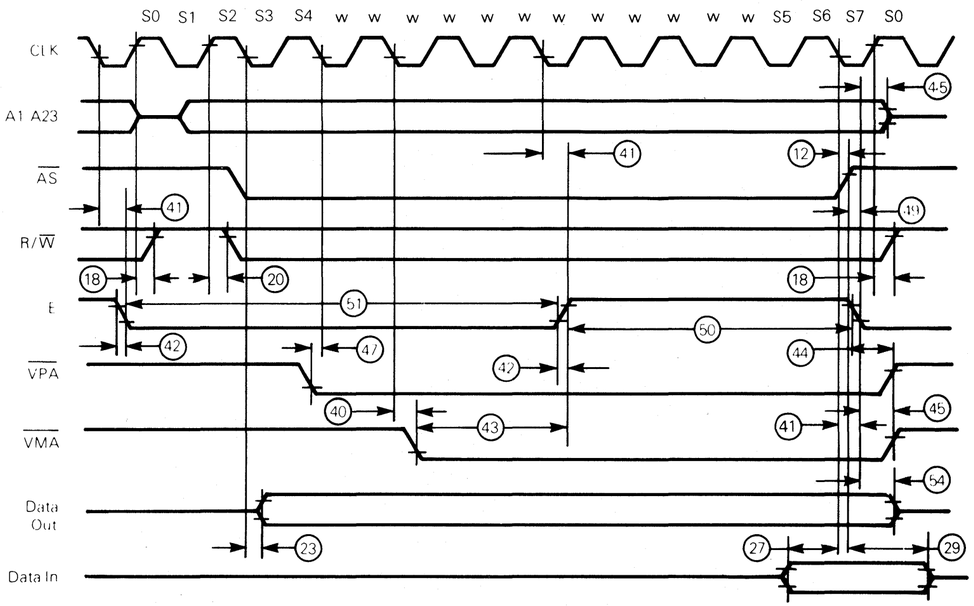


NOTES:
1. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.
2. Because of loading variations, R/W may be valid after AS even though both are initiated by the rising edge of S2 (Specification 20A).
3. The timing for the first falling edge (47) of BERR are for BERR without DTACK; the timings for the second falling edge (27A and 48) are for BERR with DTACK.

1-702

Figure 8-7. Write Cycle Timing Diagram

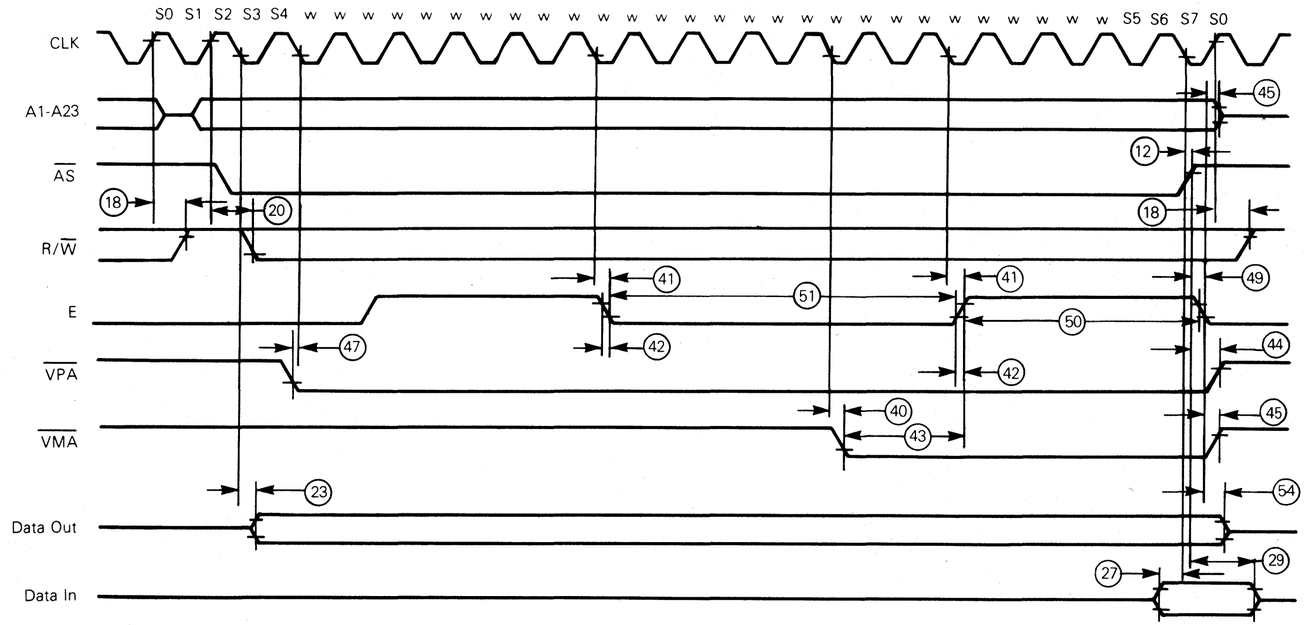
These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



NOTE: This timing diagram is included for those who wish to design their own circuit to generate VMA. It shows the best case possibly attainable.

1-80

Figure 8-8. MC68010 to M6800 Peripheral Timing Diagram — Best Case



NOTE: This timing diagram is included for those who wish to design their own circuit to generate VMA. It shows the worst case possibly attainable.

1-81

Figure 8-9. MC68010 to M6800 Peripheral Timing Diagram — Worst Case

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

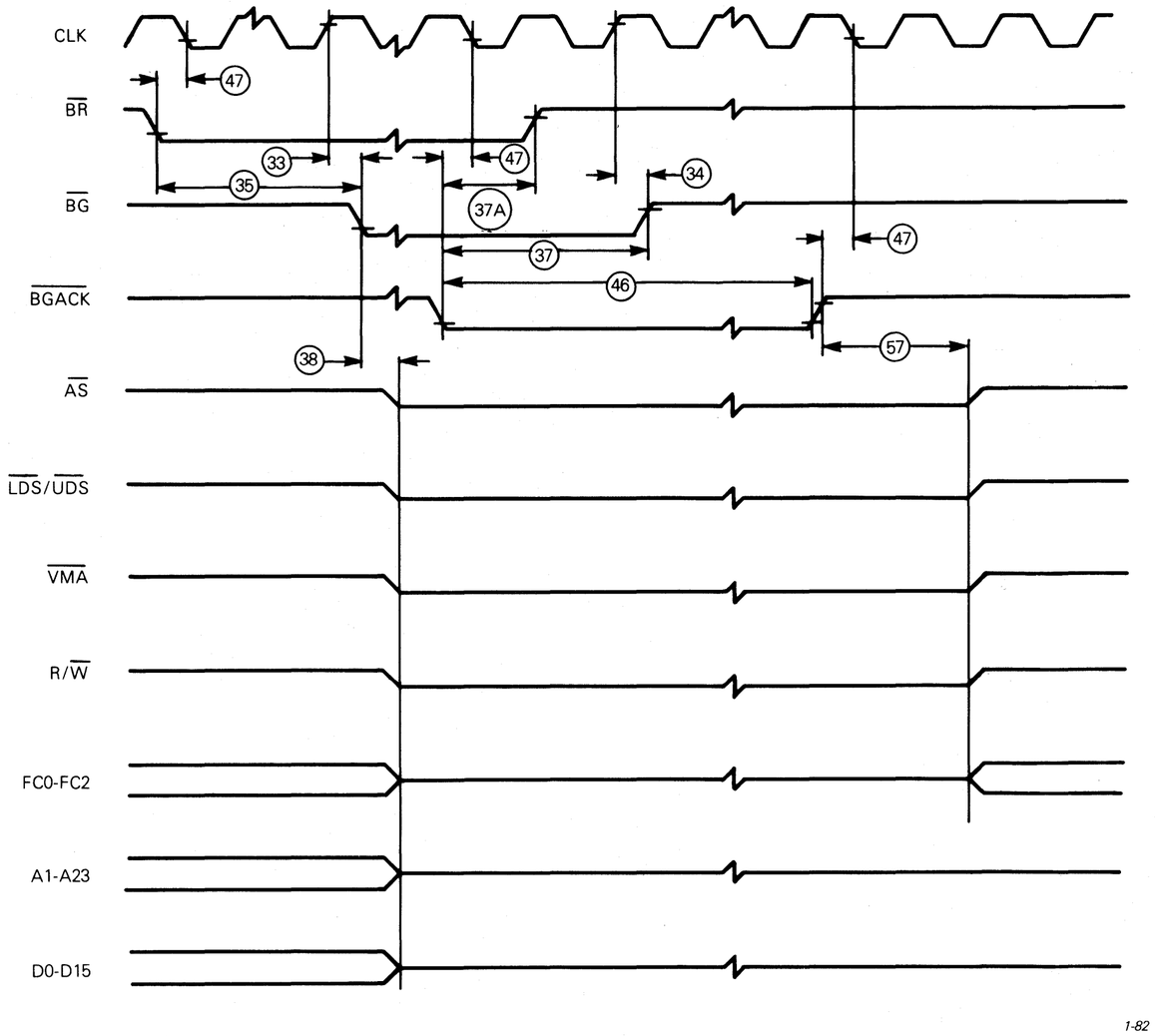


Figure 8-10. Bus Arbitration Timing — Idle Bus Case

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

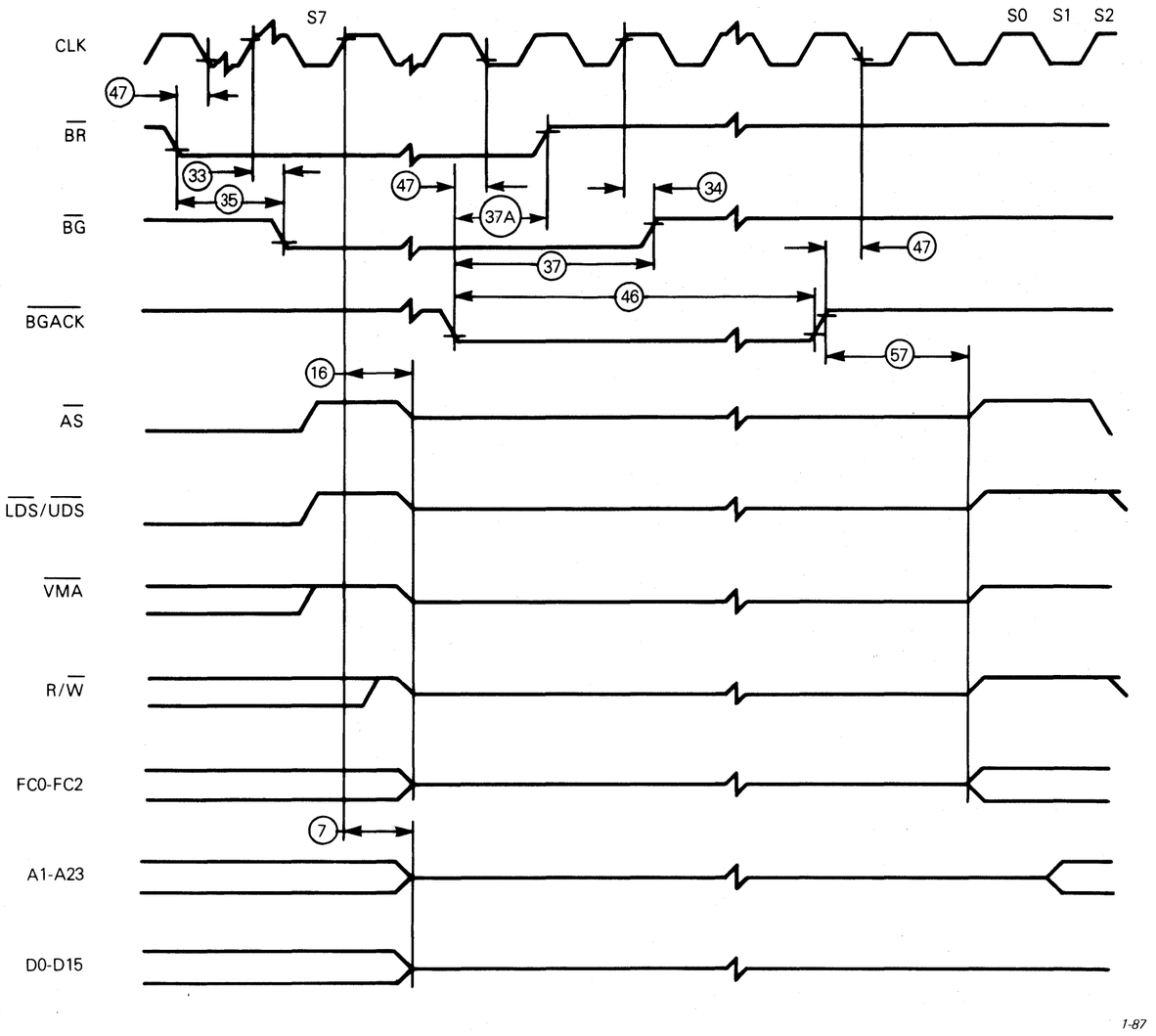


Figure 8-11. Bus Arbitration Timing — Active Bus Case

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

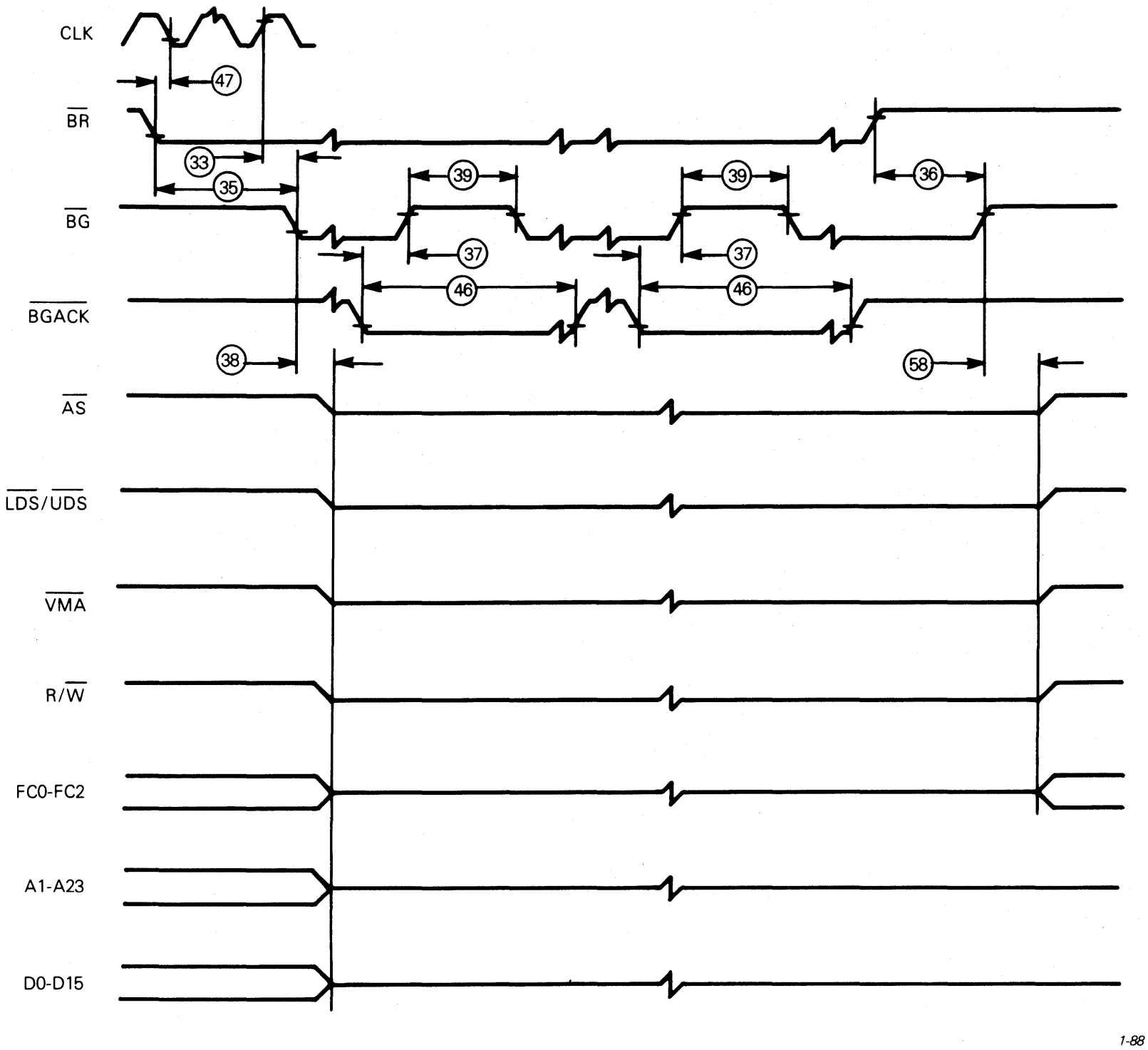


Figure 8-12. Bus Arbitration Timing — Multiple Bus Requests

MOTOROLA ***Semiconductor Products Inc.***

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721 • A SUBSIDIARY OF MOTOROLA INC.

A15344-8 PRINTED IN USA 6-85 IMPERIAL LITHO C31385 18,000

ADI942R2