Signetics

68000
16-/32-Bit Microprocessor

Product Specification

Military
Customer Specific Products

DESCRIPTION

The 68000 is the first implementation of the 68000 16/32 bit microprocessor architecture. The 68000 has a 16-bit data bus and 24-bit address bus, while the full architecture provides for 32-bit address and data buses. It is completely code-compatible with the 68008 8-bit data bus implementation of the 68000 and is downward code-compatible with the 68010 virtual extension and the 68020 32-bit implementation of the architecture. Any user-mode programs written using the 68000 instruction set will run unchanged on the 68008, 68010, and 68020. This is possible because the user programming model is identical for all four processors and the instruction sets are proper sub-sets of the complete architecture.

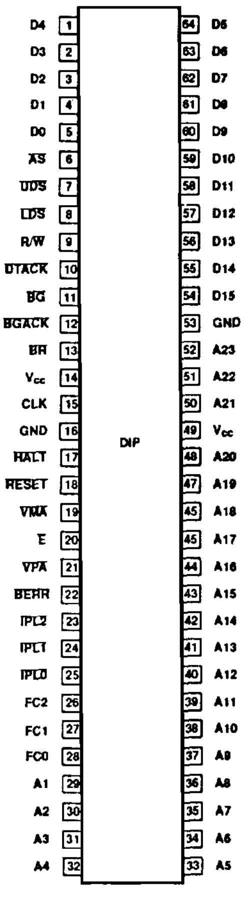
The 68000 possesses an asynchronous bus structure with a 24-bit address bus and a 16-bit data bus.

The resources available to the 68000 user consist of the following:

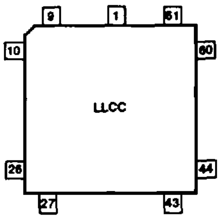
- **17 32-bit data and address registers**
- **16MB direct adressing registers**
- **56 powerful instruction types**
- **Operations on five main data types**
- **Memory-mapped I/O**
- **14 addressing modes**

As shown in the programming model (Figure 1), the 68000 offers sixteen 32-bit registers and a 32-bit program counter. The first eight registers (D0 - D7) are used as data registers for byte (8-bit), word (16-bit), and long-word (32-bit) operations. The second set of seven registers (A0 - A6) and the user stack pointer (USP) may be used as software stack pointers and base address registers. In addition, the registers may be used for word and long-word operations. All of the 16 registers may be used as index registers.

PIN CONFIGURATION



PIN CONFIGURATION



LLCC	Function	LLCC	Function
1	D4	35	A4
2	D3	36	A5
3	D2	37	A6
4	D1	38	A7
5	D0	39	A8
6	AS	40	A9
7	$\overline{\text{UDS}}$	41	A10
8	$\overline{\text{LDS}}$	42	A11
9	R/$\overline{\text{W}}$	43	A12
10	$\overline{\text{DTACK}}$	44	A13
11	$\overline{\text{BG}}$	45	A14
12	$\overline{\text{BGACK}}$	46	A15
13	$\overline{\text{BR}}$	47	A16
14	V_{CC}	48	A17
15	CLK	49	A18
16	GND	50	A19
17	GND	51	A20
18	NC	52	V_{CC}
10	$\overline{\text{HALT}}$	53	A21
20	$\overline{\text{RESET}}$	54	A22
21	$\overline{\text{VMA}}$	55	A23
22	E	56	GND
23	$\overline{\text{VPA}}$	57	GND
24	$\overline{\text{BERR}}$	58	D15
25	$\overline{\text{IPL2}}$	59	D14
26	$\overline{\text{IPL1}}$	60	D13
27	$\overline{\text{IPL0}}$	61	D12
28	FC2	62	D11
29	FC1	63	D10
30	FC0	64	D9
31	NC	65	D8
32	A1	66	D7
33	A2	67	D6
34	A3	68	D5

ORDERING INFORMATION

DESCRIPTION	ORDER CODE	SPEED
64-Pin Ceramic DIP 900mil-wide	68000-6/BXA	6MHz
	68000-8/BXA	8MHz
	68000-10/BXA	10MHz
68-Pin Ceramic LLCC	68000-6/BUC	6MHz
	68000-8/BUC	8MHz
	68000-10/BUC	10MHz

In supervisor mode, the upper byte of the status register and the supervisor stack pointer (SSP) are also available to the programmer. These registers are shown in Figure 2.

The status register, Figure 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 a trace (T) mode and in a supervisor (S) or user state.

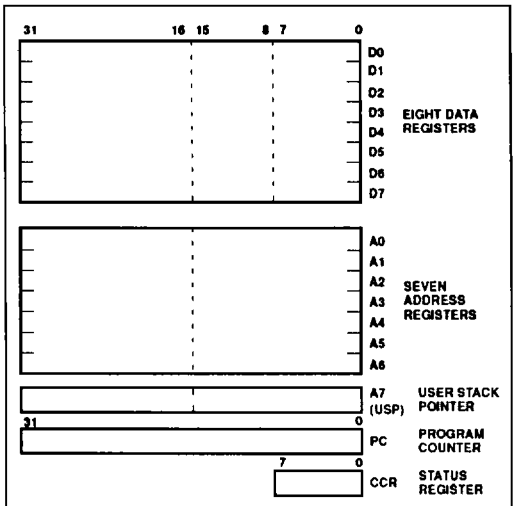


Figure 1. User Programming Model

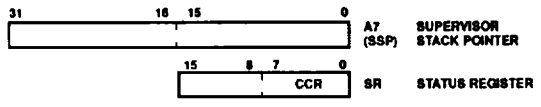


Figure 2. Supervisor Programming Model Supplement

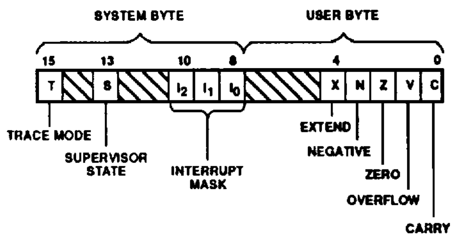


Figure 3. Status Register

Table 1. Addressing Modes

ADDRESSING MODES	SYNTAX
Register direct addressing	
Data register direct	Dn
Address register direct	An
Absolute data addressing	
Absolute short	xxx.W
Absolute long	xxx.L
Program counter relative addressing	
Relative with offset	d_{16}(PC)
Relative with index and offset	d_8(PC,Xn)
Register indirect addressing	
Register indirect	(An)
Postincrement register indirect	(An) +
Predecrement register indirect	– (An)
Register indirect with offset	d_{16}(An)
Indexed register indirect with offset	d_8(An,Xn)
Immediate data addressing	
Immediate	#xxx
Quick immediate	#1 - #8
Implied addressing	
Implied register	SR/USP/SP/PC

NOTES:

Dn	= Data register	SP	= Stack pointer
An	= Address register	USP	= User stack pointer
Xn	= Address or data register used as index register	( )	= Effective Address
SR	= Status register	d_8	= 8-bit offset (displacement)
PC	= Program counter	d_{16}	= 16-bit offset (displacement)
		#xxx	= Immediate data

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, 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 preincrementing, predecrementing, offsetting, and indexing. The program counter relative mode can also be modified via indexing and offsetting.

Table 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}	Test condition, decrement and branch
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
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
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

Instruction Set Overview

The 68000 instruction set is shown in Table 2. Some additional instructions are variations, or subsets of these, and they appear in Table 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. 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).

Table 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 register status
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
MOVE	MOVE	Move source to destination
	MOVEA	Move address
	MOVEM	Move multiple registers
	MOVEP	Move peripheral data
	MOVEQ	Move quick
	MOVE from SR	Move from status register
	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

DATA ORGANIZATION AND ADDRESSING CAPABILITIES

This section contains a description of the registers and the data organization of the 68000.

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.

Data Organization in Registers

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers together with the stack pointers support address operands of 32 bits.

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 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.

Address Registers

Each address register and the stack pointer is 32 bits wide and holds a full 32-bit address. Address registers do not support 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 on 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.

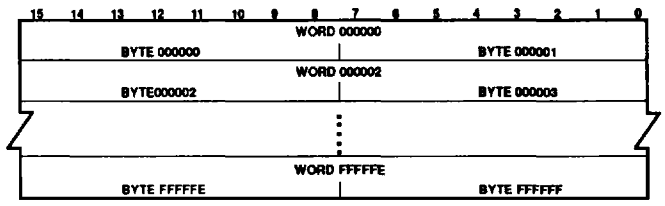


Figure 4. Word Organization in Memory

Data Organization in Memory

Bytes are individually addressable with the high-order byte having an even address the same as the word, as shown in Figure 4. The low-order byte has an odd address that is one count higher than the word address. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long-word datum is located at address n (n even), then the second word of that datum is located at address n + 2.

The data types supported by the 68000 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 5. The numbers indicate the order in which the data would be accessed from the processor.

Addressing

Instructions for the 68000 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.

Instruction Format

Instructions are from one to five words in length as shown in Figure 6. 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.

Program/Data References

The 68000 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. Operand reads are from the data space except in the case of the program counter relative addressing mode. All operand writes are to the data space.

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.

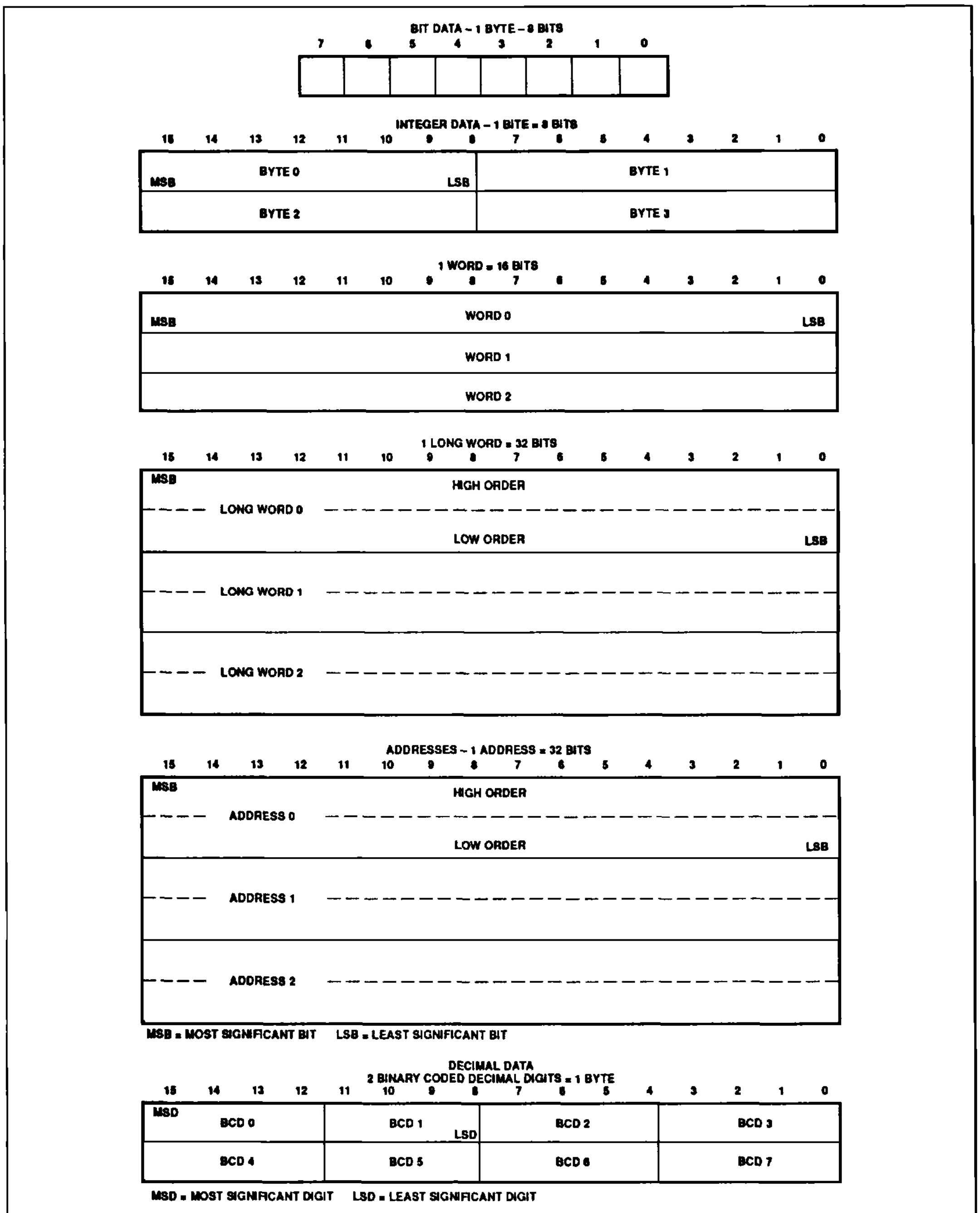


Figure 5. Memory Data Organization

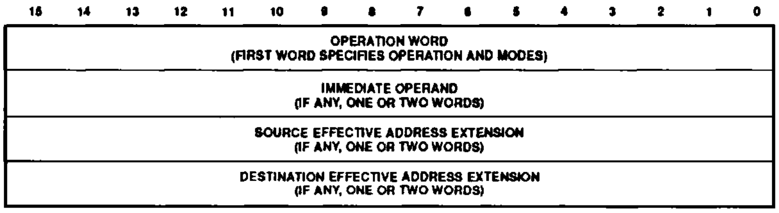

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
OPERATION WORD (FIRST WORD SPECIFIES OPERATION AND MODES)
IMMEDIATE OPERAND (IF ANY, ONE OR TWO WORDS)
SOURCE EFFECTIVE ADDRESS EXTENSION (IF ANY, ONE OR TWO WORDS)
DESTINATION EFFECTIVE ADDRESS EXTENSION (IF ANY, ONE OR TWO WORDS)

Figure 6. Instruction Operation Word General Format

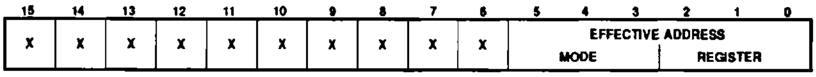


Figure 7. Single-Effective-Address Instruction Operation Word

Effective Address

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 7 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.

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 6. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

Register Direct Modes

These effective addressing modes specify that the operand is in one of 16 multifunction registers.

Data Register Direct — The operand is in the data register specified by the effective address register field.

Address Register Direct — The operand is in the address register specified by the effective address register field.

Memory Address Modes

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

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-routine instructions.

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 on 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.

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.

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.

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 contents of the index register. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

Special Address Modes

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

Absolute Short Address — This addressing mode requires one word of extension. The address of the operand is 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.

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.

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 word. The value in the program counter is the address of the extension word. The reference is classified as a program reference.

Table 4. 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

Table 5. 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	s → (EA)d
MOVEM	16, 32	(EA) → An, Dn Dn → (EA)
MOVEP	16, 32	(EA) → Dn Dn → (EA)
MOVEQ	8	#xxx → Dn
PEA	32	EA → -(SP)
SWAP	32	Dn[31:16] ↔ Dn[15:0]
UNLK	–	An → SP (SP) + → An

NOTES:

s = source [ ] = bit number ( ) + = indirect with postdecrement
d = destination -( ) = indirect with predecrement # = immediate data

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 value in the program counter is the address of the extension word. This reference is classified as a program reference.

Immediate Data — This addressing mode requires either one or two words of extension depending on the size of the operation.

Byte operation – operand is low-order byte of extension word.

Word operation – operand is 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.

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), or the status register (SR). A selected set of instructions may reference the status register by means of the effective address field. These are:

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

Effective Address Encoding Summary

Table 4 is a summary of the effective addressing modes discussed in the previous paragraphs.

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, 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.

INSTRUCTION SET SUMMARY

This section contains an overview of the form and structure of the 68000 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

Data movement	Bit Manipulation
Integer arithmetic	Binary coded decimal
Logical	Program control
Shift and rotate	System control

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

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 move 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 move 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), and move quick (MOVEQ). Table 5 is a summary of the data movement operations.

Table 6. 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) m→ 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

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 sued 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 6 is a summary of the integer arithmetic operations.

Table 7. Logical Operations

INSTRUCTION	OPERAND SIZE	OPERATION
AND	8, 16, 32	Dn ∧ (EA) → Dn (EA) ∧ Dn → (EA) (EA) ∧ #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)

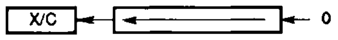
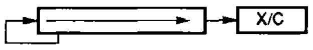
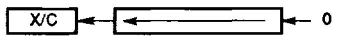
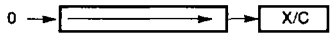
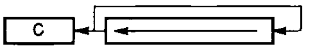
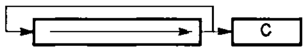
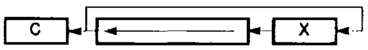
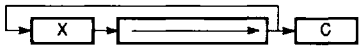
NOTES:

#	=	immediate data
~	=	invert
∧	=	logical AND
V	=	logical OR
⊕	=	logical Exclusive-OR

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 7 is a summary of the logical operations.

Table 8. 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	

Shift and Rotate Operations

Shift operations in both directions are provided by the arithmetic 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 8 is a summary of the shift and rotate operations.

Table 9. 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 E

NOTE: ~ = invert

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 9 is a summary of the bit manipulation operations. (Z is bit 2 of the status register.)

Table 10. 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	$0 - (EA)_{10} - X \rightarrow (EA)$

NOTE: –( ) = indirect with predecrement.

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 10 is a summary of the binary coded decimal operations.

Table 11. 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	
RTR	Return and restore condition codes
RTS	Return from subroutine

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 11.

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

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

Table 12. 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 USP	Move user stack pointer
ORI to SR	Lorical 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
Status 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 SR to EA	Store status register
ORI to CCR	Logical OR to condition codes

System Control Operations

System control operations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the status register. These instructions are summarized in Table 12.

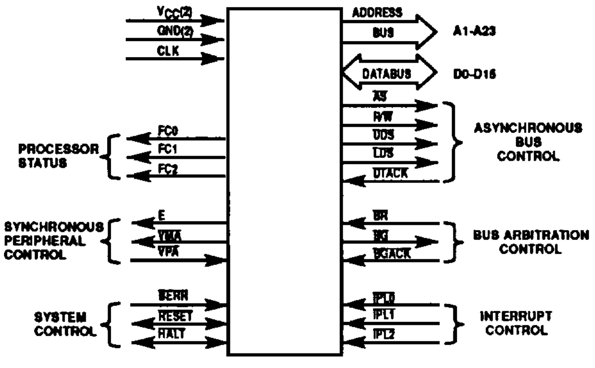


Figure 8. Input and Output Signals

Table 13. Data Strobe Control of Data Bus

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	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*

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

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 give.

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.

Signal Description

The input and output signals can be functionally organized into the groups shown in Figure 8. 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.

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 interrupt cycles. During interrupt cycles, address lines A1, A2, and A3 provide information about what level interrupt is being serviced while address lines A4 through A23 are all set to a logic high.

Data Bus (D0 through D15)

This 16-bit bidirectional, three-state bus is the general purpose data path. It can transfer and accept data in either word or byte length. During an interrupt acknowledge cycle, the external device supplies the vector number on data lines D0 - D7.

Asynchronous Bus Control

Asynchronous data transfers are handled using the following control signals: **address strobe, read/write, upper and lower data strobes, and data transfer acknowledge. These signals are explained in the following paragraphs.**

Address Strobe ($\overline{AS}$) — This signal indicates that there is a valid address on the address bus.

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

Upper and Lower Data Strobe ($\overline{UDS}$, $\overline{LDS}$) — These signals control the flow of data on the data bus, as shown in Table 13. When the R/$\overline{W}$ line is high, the processor will read from the data bus as indicated. When the R/$\overline{W}$ line is low the processor will write to the data bus as shown.

Data Transfer Acknowledge ($\overline{DTACK}$) — This input indicates that the data transfer is completed. When the processor recognizes $\overline{DTACK}$ during a read cycle, data is latched and the bus cycle terminated. When $\overline{DTACK}$ is recognized during a write cycle, the bus cycle is terminated. (Refer to **Asynchronous Versus Synchronous Operation**).

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.

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.

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.

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.

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 given in $\overline{IPL0}$ and the most significant bit is contained in $\overline{IPL2}$. These lines must remain stable until the processor signals interrupt acknowledge (FC0 - FC2 are all high) to insure that the interrupt is recognized.

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.

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 **Bus Error and Halt Operation** for additional information about the interaction of the bus error and halt signals.

Reset ($\overline{RESET}$) — 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 $\overline{RESET}$ 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 $\overline{HALT}$ and $\overline{RESET}$ signals applied at the same time. Refer to **Reset Operation** for further information.

Halt ($\overline{HALT}$) — 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 3-State lines are put in their high-impedance state (refer to Table 15). Refer to **Bus Error and Halt Operation** for additional information about the interaction between the $\overline{HALT}$ and bus error signals.

When the processor has stopped executing instructions, such as in a double bus fault condition (refer to **Double Bus Faults**), the $\overline{HALT}$ line is driven by the processor to indicate to external devices that the processor has stopped.

Peripheral Control
These control signals are used to allow the interfacing of synchronous peripheral devices with the asynchronous 68000. These signals are explained in the following paragraphs.

Enable (E) — This signal is the standard enable signal common to all synchronous type peripheral devices. The period for this output is ten 68000 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.

Valid Peripheral Address ($\overline{VPA}$) —This input indicates that the device or region addressed is a synchronous 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 **Interface with Synchronous Peripherals**.

Valid Memory Address ($\overline{VMA}$) — This output is used to indicate to synchronous peripheral devices that there is a valid address on the address bus and the processor is synchronized to enable. This signal only responds to a valid peripheral address ($\overline{VPA}$) input which indicates that the peripheral is a synchronous family device.

Processor Status (FC0, FC1, FC2)
These function code outputs indicate the state (user or supervisor) and the cycle type currently being executed, as shown in Table 14. The information indicated by the function code outputs is valid whenever address strobe ($\overline{AS}$) is active.

Table 14. Function Code Outputs

FUNCTION CODE OUTPUT			CYCLE TYPE
FC2	FC1	FC0	
Low	Low	Low	(Undefined, reserved)
Low	Low	High	User data
Low	High	Low	User program
Low	High	High	(Undefined, reserved)
High	Low	Low	(Undefined, reserved)
High	Low	High	Supervisor data
High	High	Low	Supervisor program
High	High	High	Interrupt acknowledge

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.

Signal Summary
Table 15 is a summary of all the signals discussed in the previous paragraphs.

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.

Data Transfer Operations

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 68000 for interlocked multiprocessor communications.

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 double 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.

A word read cycle flowchart is given in Figure 9. A byte read cycle flowchart is given in Figure 10. Read cycle timing is given in Figure 11. Figure 12 details word and byte read cycle operations.

Table 15. Signal Summary

SIGNAL NAME	MNEMONIC	INPUT/OUTPUT	ACTIVE STATE	HI-Z	
				on $\overline{HALT}$	on $\overline{BGACK}$
Address bus	A1-A23	Output	High	Yes	Yes
Data bus	D0-D15	Input/output	High	Yes	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 strobes	$\overline{UDS}$, $\overline{LDS}$	Output	Low	No	Yes
Data transfer acknowledge	$\overline{DTACK}$	Input	Low	No	No
Bus request	$\overline{BR}$	Input	Low	No	No
Bus grant	$\overline{BG}$	Output	Low	No	No
Bus grant acknowledge	$\overline{BGACK}$	Input	Low	No	No
Interrupt priority level	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	Input	Low	No	No
Bus error	$\overline{BERR}$	Input	Low	No	No
Reset	$\overline{RESET}$	Input/output	Low	No[1]	No[1]
Halt	$\overline{HALT}$	Input/output	Low	No[1]	No[1]
Enable	E	Output	High	No	No
Valid memory address	$\overline{VMA}$	Output	Low	No	Yes
Valid peripheral address	$\overline{VPA}$	Input	Low	No	No
Function code output	FC0, FC1, FC2	Output	High	No[2]	Yes
Clock	CLK	Input	High	No	No
Power input	V_{CC}	Input	–	–	–
Ground	GND	Input	–	–	–

NOTES:
1. Open Drain
2. Function codes are placed in high-impedance state during HALT.

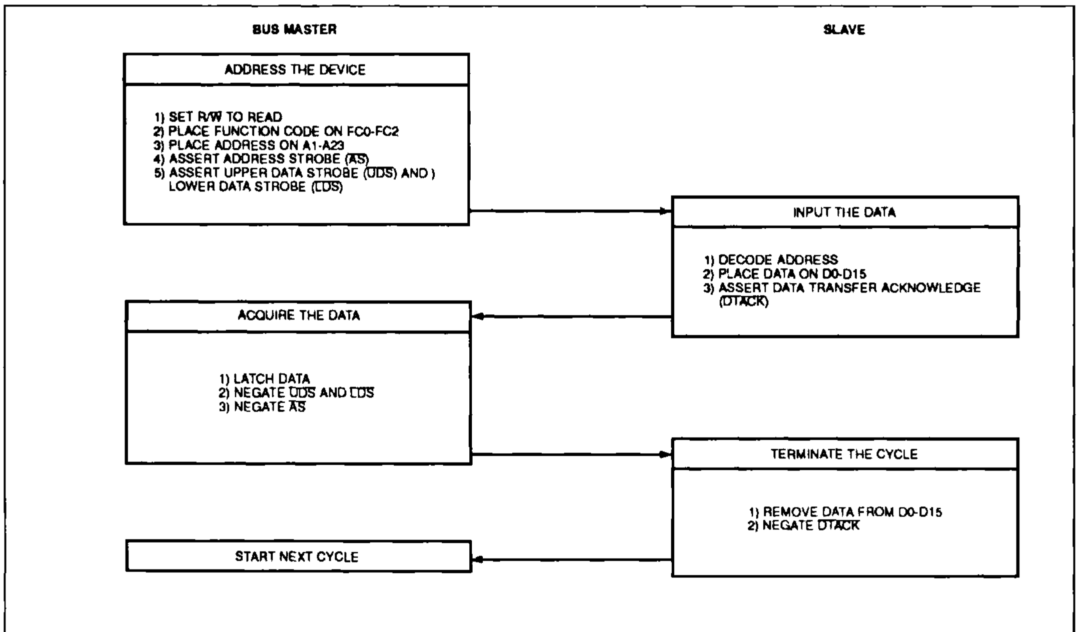

BUS MASTER
SLAVE
ADDRESS THE DEVICE
1) SET R/$\overline{W}$ TO READ
2) PLACE FUNCTION CODE ON FC0-FC2
3) PLACE ADDRESS ON A1-A23
4) ASSERT ADDRESS STROBE ($\overline{AS}$)
5) ASSERT UPPER DATA STROBE ($\overline{UDS}$) AND) LOWER DATA STROBE ($\overline{LDS}$)
INPUT THE DATA
1) DECODE ADDRESS
2) PLACE DATA ON D0-D15
3) ASSERT DATA TRANSFER ACKNOWLEDGE ($\overline{DTACK}$)
ACQUIRE THE DATA
1) LATCH DATA
2) NEGATE $\overline{UDS}$ AND $\overline{LDS}$
3) NEGATE $\overline{AS}$
TERMINATE THE CYCLE
1) REMOVE DATA FROM D0-D15
2) NEGATE $\overline{DTACK}$
START NEXT CYCLE

Figure 9. Word Read Cycle Flowchart

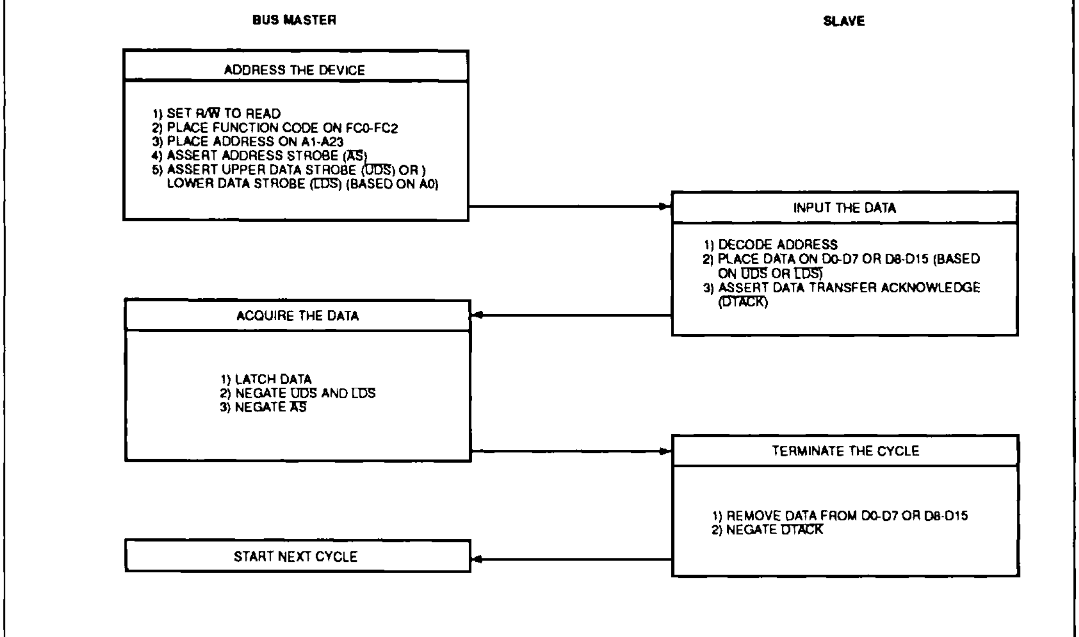

BUS MASTER
SLAVE
ADDRESS THE DEVICE
1) SET R/$\overline{W}$ TO READ
2) PLACE FUNCTION CODE ON FC0-FC2
3) PLACE ADDRESS ON A1-A23
4) ASSERT ADDRESS STROBE ($\overline{AS}$)
5) ASSERT UPPER DATA STROBE ($\overline{UDS}$) OR) LOWER DATA STROBE ($\overline{LDS}$) (BASED ON A0)
INPUT THE DATA
1) DECODE ADDRESS
2) PLACE DATA ON D0-D7 OR D8-D15 (BASED ON $\overline{UDS}$ OR $\overline{LDS}$)
3) ASSERT DATA TRANSFER ACKNOWLEDGE ($\overline{DTACK}$)
ACQUIRE THE DATA
1) LATCH DATA
2) NEGATE $\overline{UDS}$ AND $\overline{LDS}$
3) NEGATE $\overline{AS}$
TERMINATE THE CYCLE
1) REMOVE DATA FROM D0-D7 OR D8-D15
2) NEGATE $\overline{DTACK}$
START NEXT CYCLE

Figure 10. Byte Read Cycle Flowchart

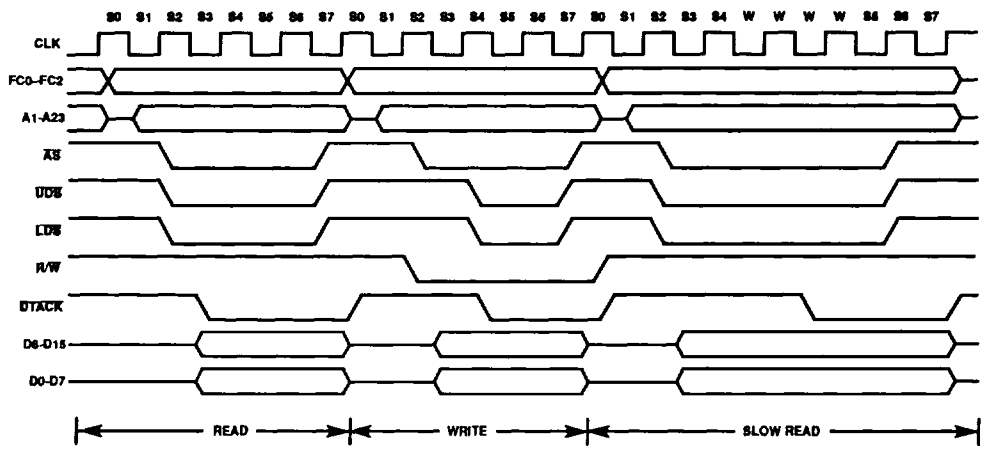

S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 W W W W S4 S6 S7
CLK
FC0-FC2
A1-A23
AS
UDS
LDS
R/W
DTACK
D8-D15
D0-D7
READ
WRITE
SLOW READ

Figure 11. Read and Write Cycle Timing Diagram

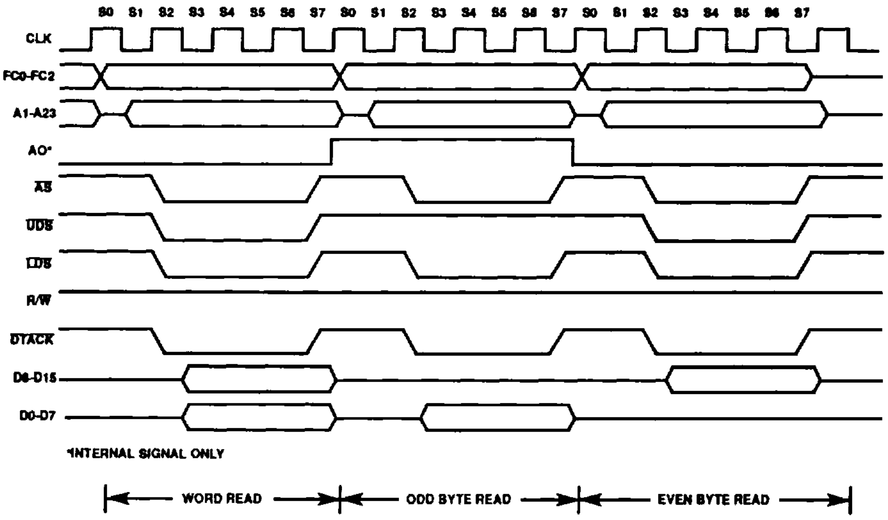


Figure 12. Word and Byte Read Cycle Timing Diagram

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 cycle flowchart is given in Figure 13. A byte write cycle flowchart is given in Figure 14. Write cycle timing is given in Figure 11. Figure 15 details word and byte write cycle operation.

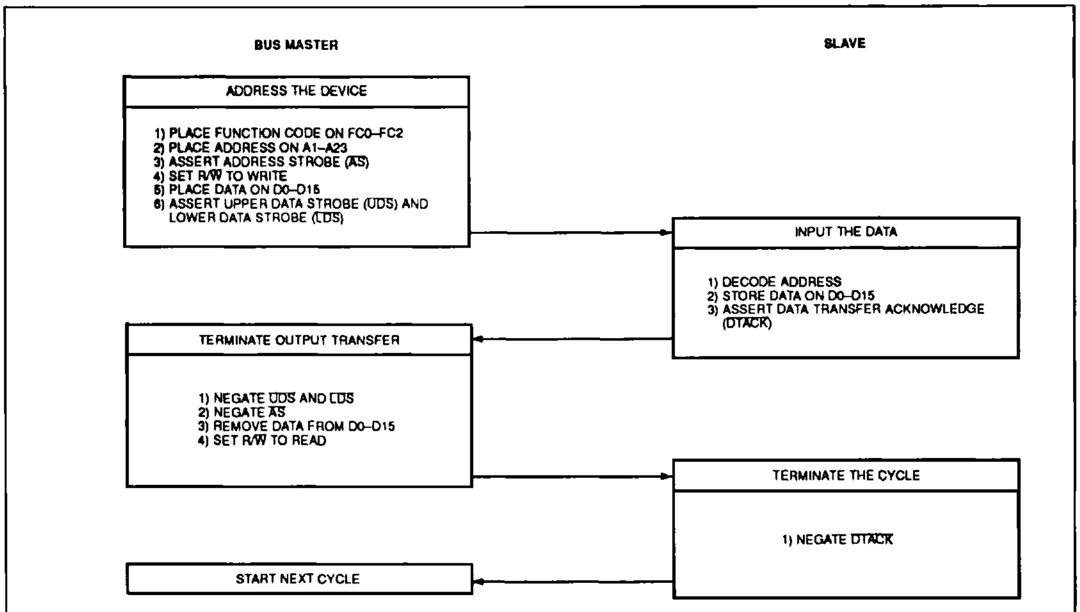

BUS MASTER
SLAVE
ADDRESS THE DEVICE
1) PLACE FUNCTION CODE ON FC0–FC2
2) PLACE ADDRESS ON A1–A23
3) ASSERT ADDRESS STROBE ($\overline{AS}$)
4) SET R/$\overline{W}$ TO WRITE
5) PLACE DATA ON D0–D15
6) ASSERT UPPER DATA STROBE ($\overline{UDS}$) AND LOWER DATA STROBE ($\overline{LDS}$)
INPUT THE DATA
1) DECODE ADDRESS
2) STORE DATA ON D0–D15
3) ASSERT DATA TRANSFER ACKNOWLEDGE ($\overline{DTACK}$)
TERMINATE OUTPUT TRANSFER
1) NEGATE $\overline{UDS}$ AND $\overline{LDS}$
2) NEGATE $\overline{AS}$
3) REMOVE DATA FROM D0–D15
4) SET R/$\overline{W}$ TO READ
TERMINATE THE CYCLE
1) NEGATE $\overline{DTACK}$
START NEXT CYCLE

Figure 13. Word Write Cycle Flowchart

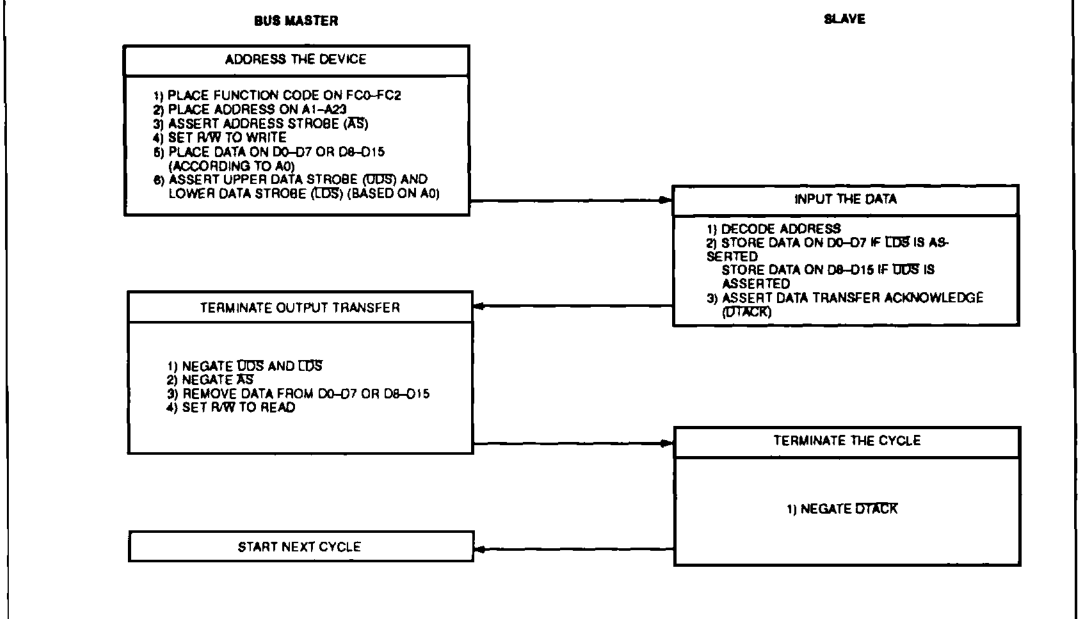

BUS MASTER
SLAVE
ADDRESS THE DEVICE
1) PLACE FUNCTION CODE ON FC0–FC2
2) PLACE ADDRESS ON A1–A23
3) ASSERT ADDRESS STROBE (AS)
4) SET R/W TO WRITE
5) PLACE DATA ON D0–D7 OR D8–D15 (ACCORDING TO A0)
6) ASSERT UPPER DATA STROBE (UDS) AND LOWER DATA STROBE (LDS) (BASED ON A0)
INPUT THE DATA
1) DECODE ADDRESS
2) STORE DATA ON D0–D7 IF LDS IS ASSERTED
STORE DATA ON D8–D15 IF UDS IS ASSERTED
3) ASSERT DATA TRANSFER ACKNOWLEDGE (DTACK)
TERMINATE OUTPUT TRANSFER
1) NEGATE UDS AND LDS
2) NEGATE AS
3) REMOVE DATA FROM D0–D7 OR D8–D15
4) SET R/W TO READ
TERMINATE THE CYCLE
1) NEGATE DTACK
START NEXT CYCLE

Figure 14. Byte Write Cycle Flowchart

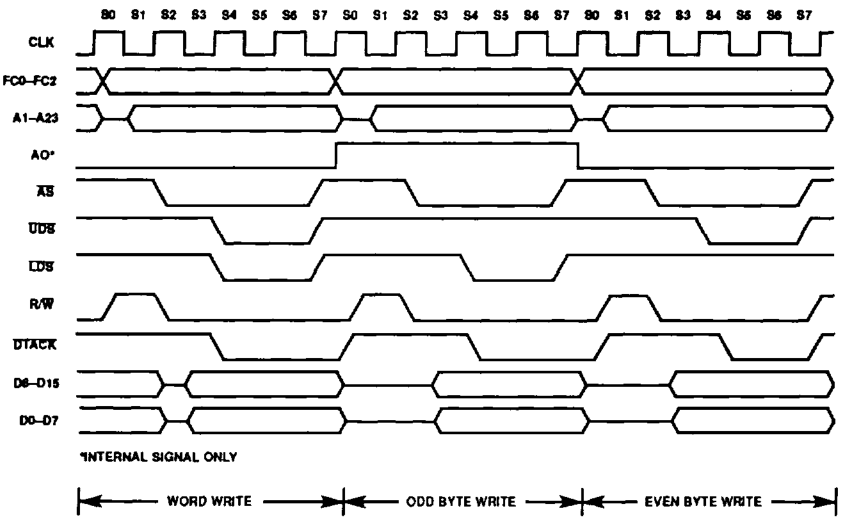


Figure 15. Word and Byte Write Cycle Timing Diagram

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 68000, 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 cycles and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write cycle flowchart is given in Figure 16 and a timing diagram is given in Figure 17.

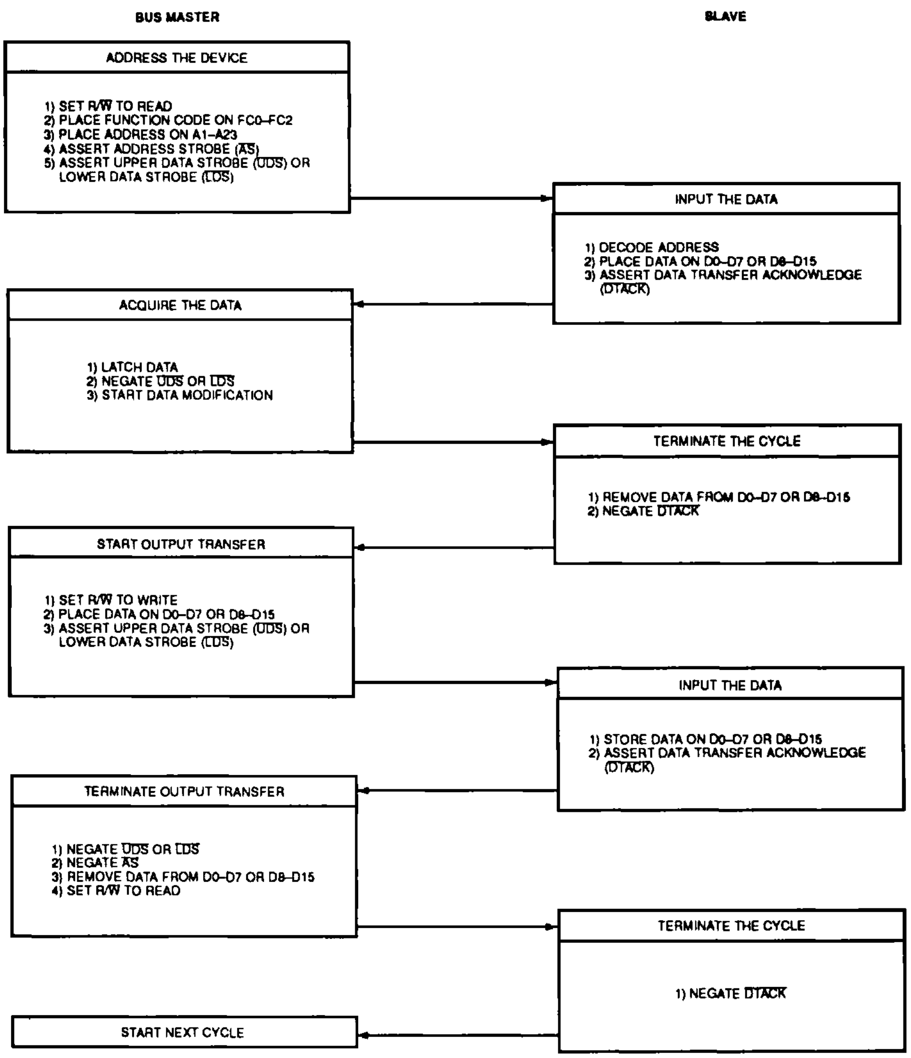

BUS MASTER
SLAVE
ADDRESS THE DEVICE
1) SET R/W̄ TO READ
2) PLACE FUNCTION CODE ON FC0–FC2
3) PLACE ADDRESS ON A1–A23
4) ASSERT ADDRESS STROBE (ĀS̄)
5) ASSERT UPPER DATA STROBE (ŪD̄S̄) OR LOWER DATA STROBE (L̄D̄S̄)
INPUT THE DATA
1) DECODE ADDRESS
2) PLACE DATA ON D0–D7 OR D8–D15
3) ASSERT DATA TRANSFER ACKNOWLEDGE (D̄T̄ĀC̄K̄)
ACQUIRE THE DATA
1) LATCH DATA
2) NEGATE ŪD̄S̄ OR L̄D̄S̄
3) START DATA MODIFICATION
TERMINATE THE CYCLE
1) REMOVE DATA FROM D0–D7 OR D8–D15
2) NEGATE D̄T̄ĀC̄K̄
START OUTPUT TRANSFER
1) SET R/W̄ TO WRITE
2) PLACE DATA ON D0–D7 OR D8–D15
3) ASSERT UPPER DATA STROBE (ŪD̄S̄) OR LOWER DATA STROBE (L̄D̄S̄)
INPUT THE DATA
1) STORE DATA ON D0–D7 OR D8–D15
2) ASSERT DATA TRANSFER ACKNOWLEDGE (D̄T̄ĀC̄K̄)
TERMINATE OUTPUT TRANSFER
1) NEGATE ŪD̄S̄ OR L̄D̄S̄
2) NEGATE ĀS̄
3) REMOVE DATA FROM D0–D7 OR D8–D15
4) SET R/W̄ TO READ
TERMINATE THE CYCLE
1) NEGATE D̄T̄ĀC̄K̄
START NEXT CYCLE

Figure 16. Read-Modify-Write Cycle Flowchart

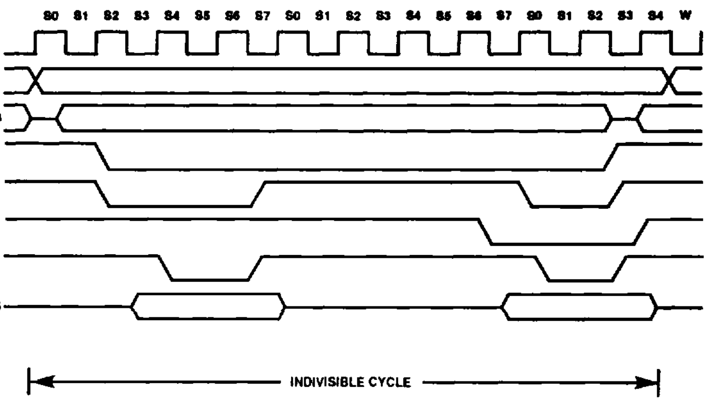

S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 W
CLK
FC0–FC2
A1–A23
AS
UDS OR LDS
R/W
DTACK
D8–D15
INDIVISIBLE CYCLE

Figure 17. Read-Modify-Write Cycle Timing Diagram

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 18 is a flowchart showing the detail involved in a request from a single device. Figure 19 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 (BGACK) signal.

However, if the 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.

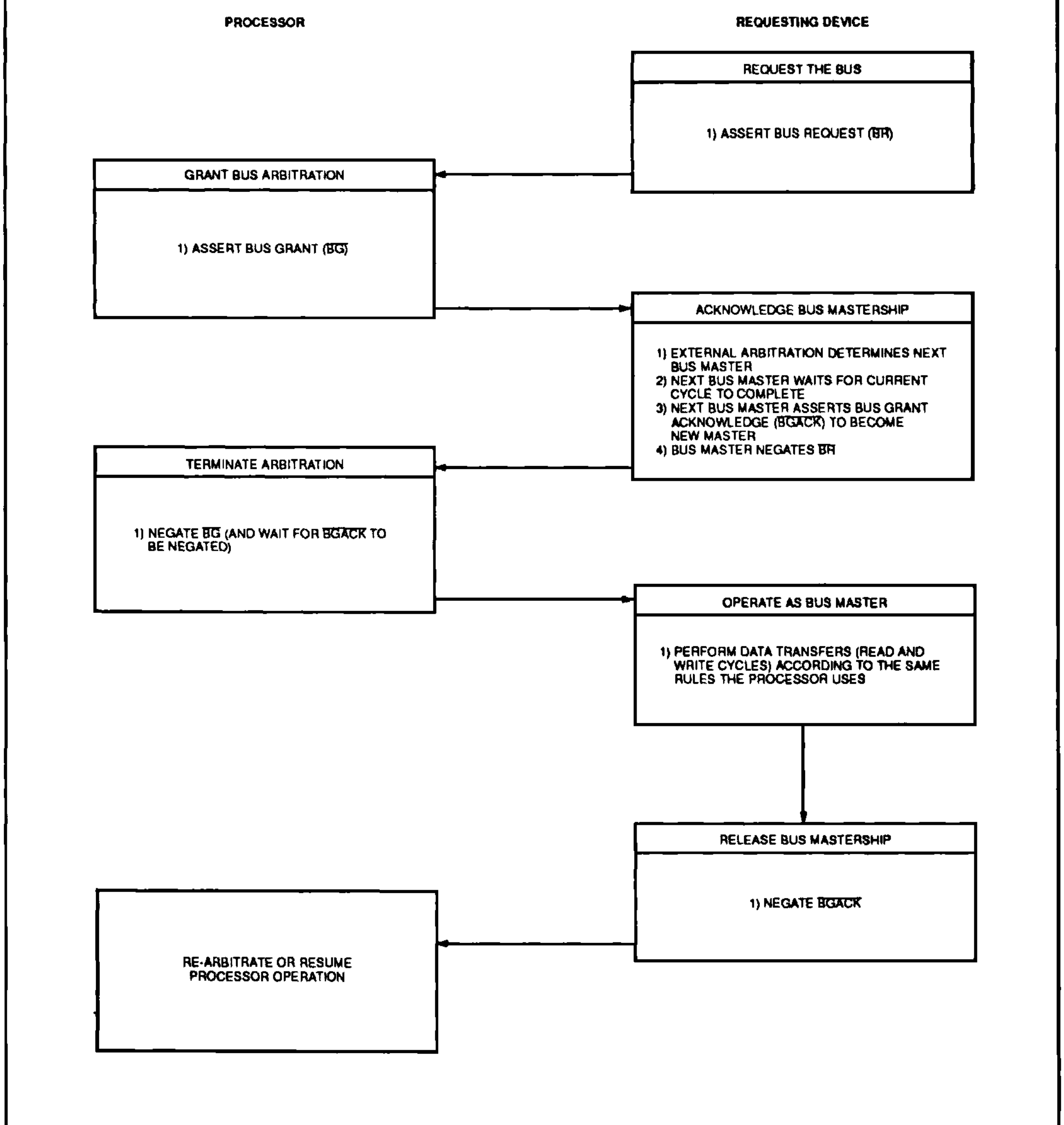

PROCESSOR
REQUESTING DEVICE
REQUEST THE BUS
1) ASSERT BUS REQUEST ($\overline{BR}$)
GRANT BUS ARBITRATION
1) ASSERT BUS GRANT ($\overline{BG}$)
ACKNOWLEDGE BUS MASTERSHIP
1) EXTERNAL ARBITRATION DETERMINES NEXT BUS MASTER
2) NEXT BUS MASTER WAITS FOR CURRENT CYCLE TO COMPLETE
3) NEXT BUS MASTER ASSERTS BUS GRANT ACKNOWLEDGE ($\overline{BGACK}$) TO BECOME NEW MASTER
4) BUS MASTER NEGATES $\overline{BR}$
TERMINATE ARBITRATION
1) NEGATE $\overline{BG}$ (AND WAIT FOR $\overline{BGACK}$ TO BE NEGATED)
OPERATE AS BUS MASTER
1) PERFORM DATA TRANSFERS (READ AND WRITE CYCLES) ACCORDING TO THE SAME RULES THE PROCESSOR USES
RELEASE BUS MASTERSHIP
1) NEGATE $\overline{BGACK}$
RE-ARBITRATE OR RESUME PROCESSOR OPERATION

Figure 18. Bus Arbitration Cycle Flowchart

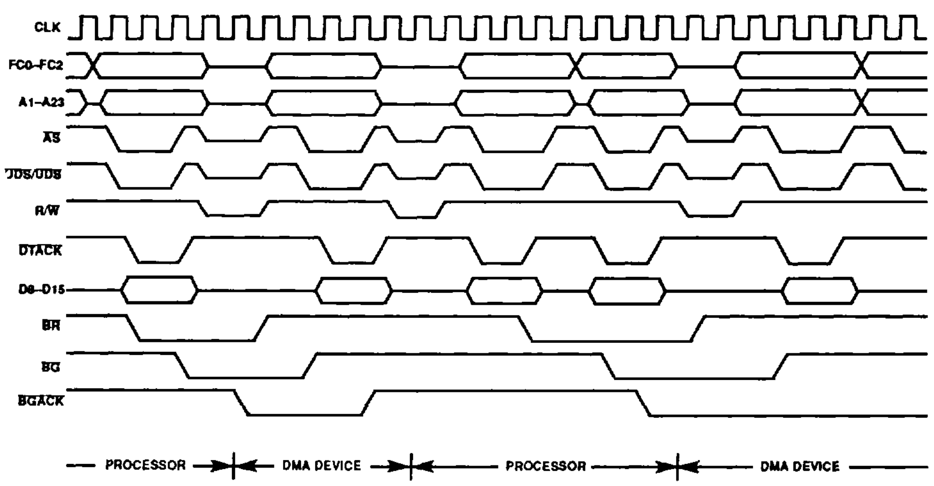


Figure 19. 68000 Bus Arbitration Cycle Timing Diagram

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.

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, but grant will be delayed until $\overline{AS}$ is asserted to indicate external devices that a bus cycle is being executed.

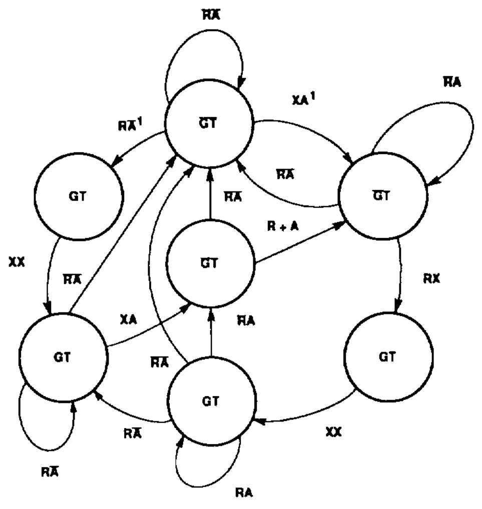
The bus grant signal may be routed through a daisy-chained network or through a specific priority-encoded network. The processor is not affected by the external method of arbitration as long as the protocol is obeyed.

Acknowledgment 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 address strobe indicates that the previous master has completed its cycle; the negation of bus grant acknowledge 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 data transfer acknowledge 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 dropped 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 **Bus Arbitration Control**. Note that the processor does not perform any external bus cycles before it re-asserts bus grant.

Bus Arbitration Control

The bus arbitration control unit in the SCN68000 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 20. All asynchronous signals to the SCN68000 are synchronized before being used internally. This synchronization is accomplished in a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (#47) has been met (see Figure 21). The input signal is sampled on the falling edge of the clock and is valid internally after the next falling edge.



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

Figure 20. 68000 Bus Arbitration Control Unit State Diagram

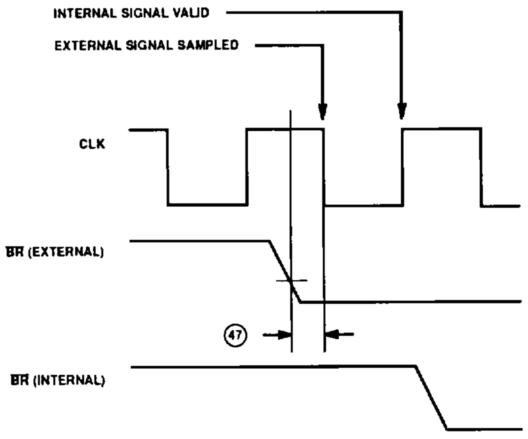


Figure 21. Timing Relationship of External Asynchronous Inputs to Internal Signals

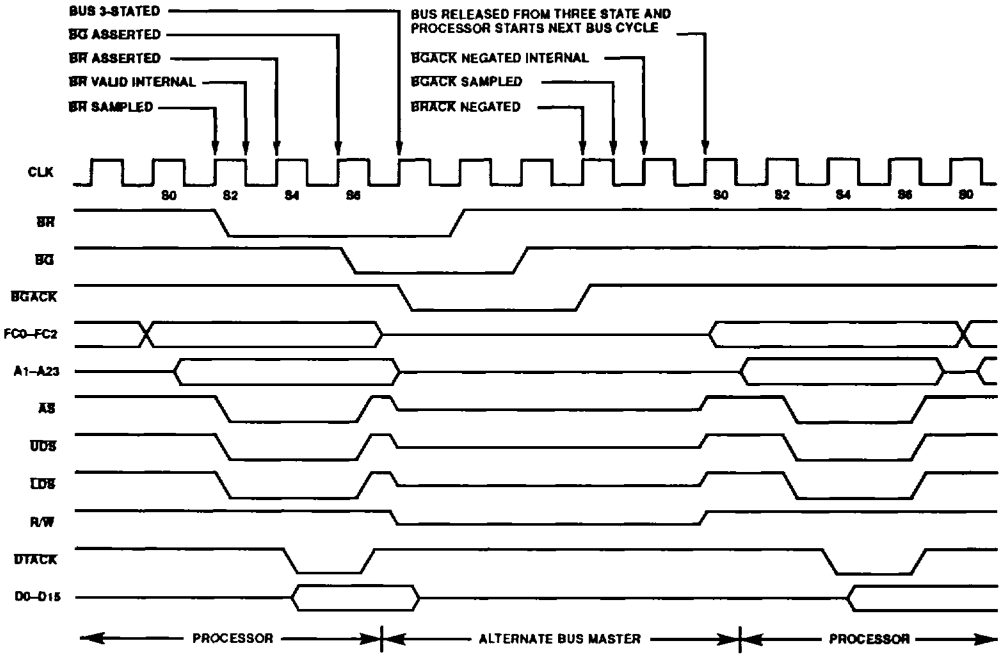


Figure 22. Bus Arbitration Timing Diagram—Processor Active

As shown in Figure 20, 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 3-State control signal 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.

A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 22. the bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 23.

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 24.

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 signal is received, the processor has two options: initiate a bus error exception sequence or try running the bus cycle again.

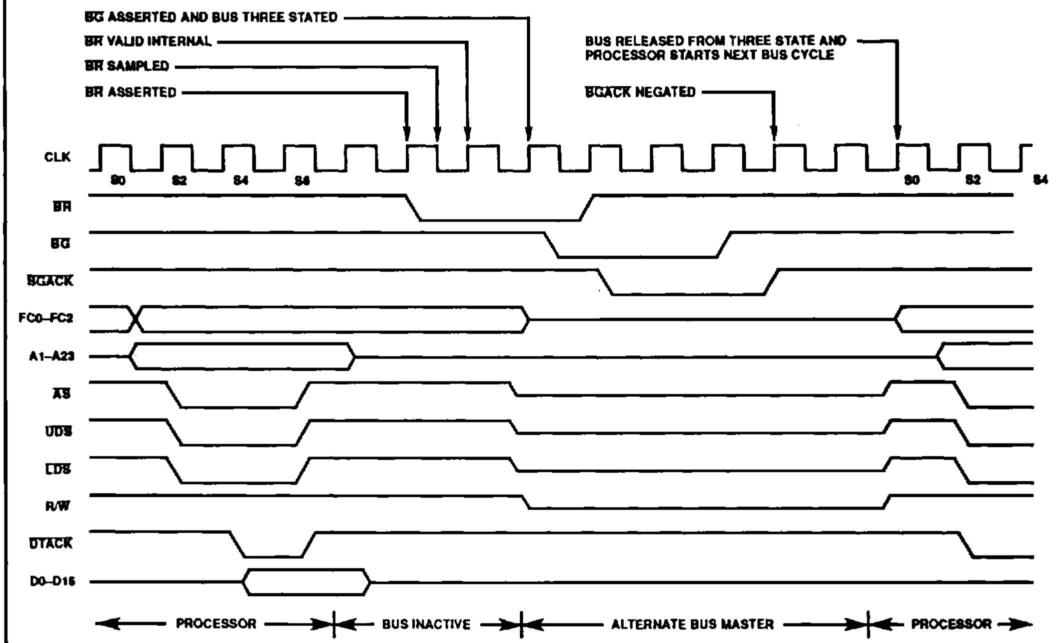


Figure 23. Bus Arbitration Timing Diagram—Bus Inactive

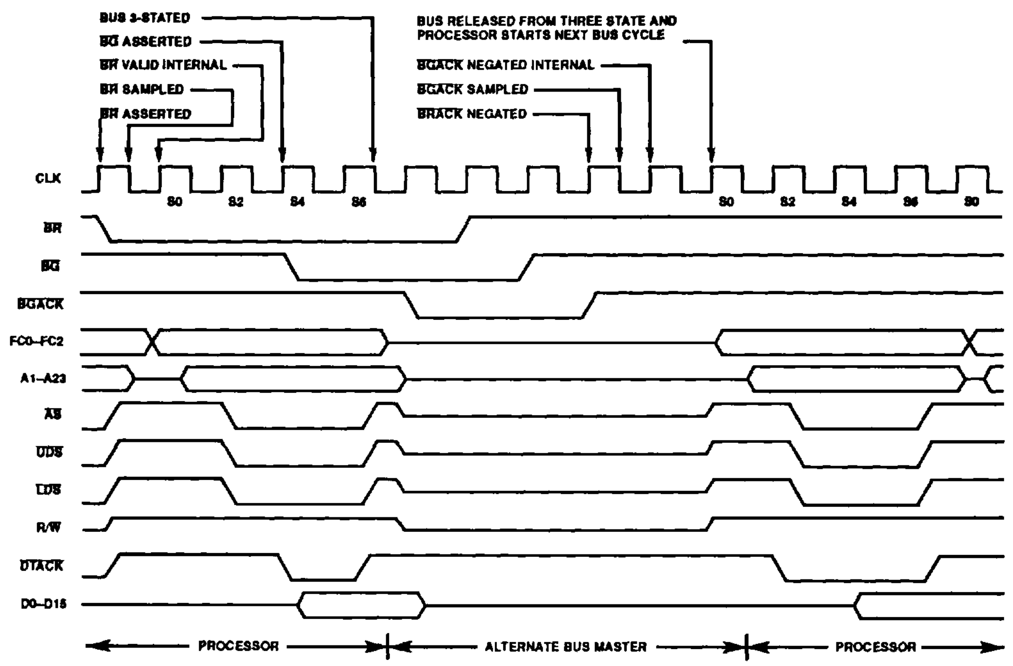


Figure 24. Bus Arbitration Timing Diagram—Special Case

Bus Error Operation — When the bus error signal is asserted, the current bus cycle is terminated. If $\overline{BERR}$ is asserted before the falling edge of S2, $\overline{AS}$ will be negated in S7 in either a read or write cycle. As long as $\overline{BERR}$ remains asserted, the data and address buses will be in the high-impedance state. When $\overline{BERR}$ is negated, the processor will begin stacking for exception processing. Figure 25 is a timing diagram for the exception sequence. The sequence is composed of the following elements:

1. stacking the program counter and status register,
2. stacking the error information,
3. reading the bus error vector table entry, and
4. executing the bus error handler routine.

The stacking of the program counter and the status register is the same as if an interrupt had occurred. Several additional items are stacked when a bus error occurs. These items are used to determine the nature of the error and correct it, if possible. The bus error vector is vector number two located at address \$000008. The processor loads the new program counter from this location. A software bus error handler routine is then executed by the processor. Refer to **Exception Processing** for additional information.

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. Figure 26 is a timing diagram for re-running the bus cycle.

The processor terminates the bus cycle, then puts the address and data output lines in the high-impedance 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 controls. 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.

Halt Operation — The halt input signal to the 68000 performs a halt/run/single-step function in a similar fashion to the synchronous device 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 debut) processor operations one bus cycle at a time.

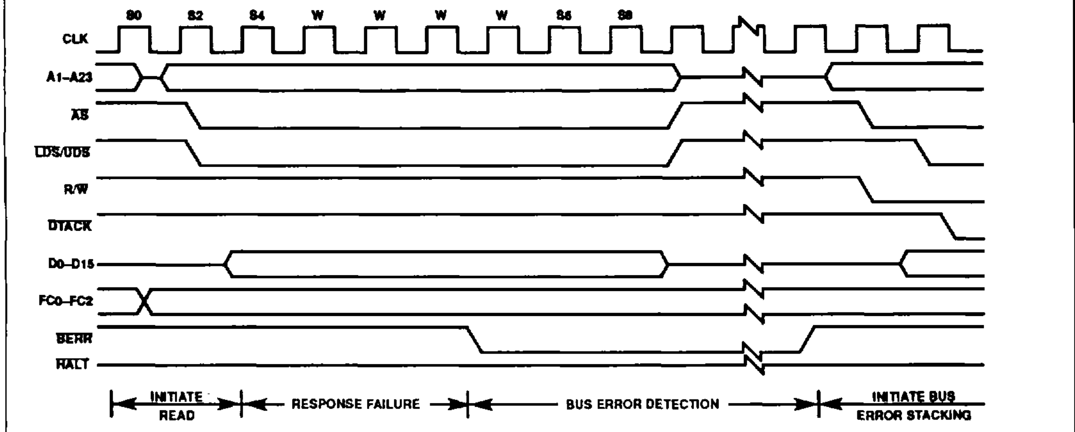


Figure 25. Bus Error Timing Diagram

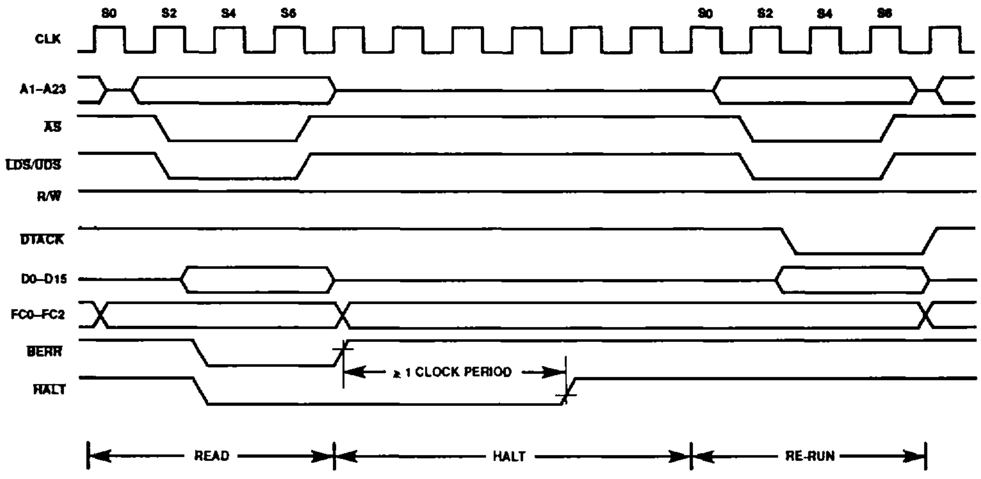


Figure 26. Re-Run Bus Cycle Timing Diagram

Figure 27 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.

When the processor completes a bus cycle after recognizing that the halt signal is active, most 3-State signals are put in the high-impedance 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.

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. 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.

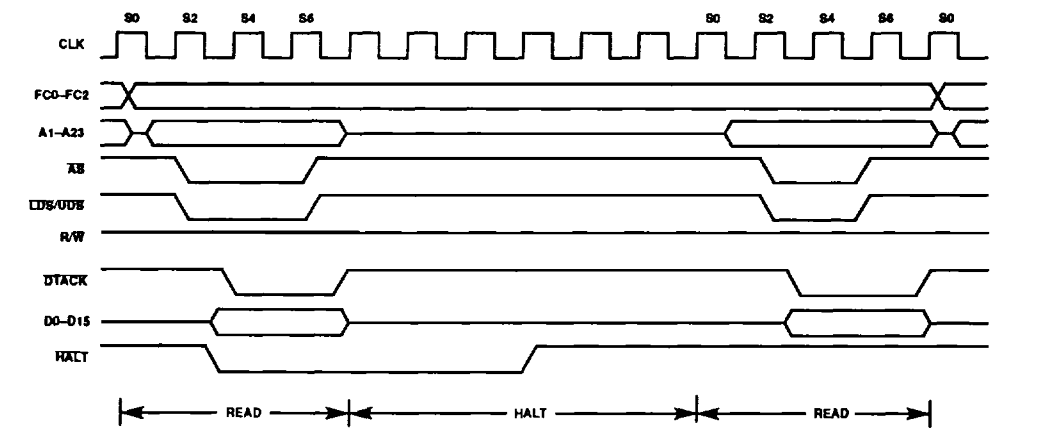


Figure 27. Halt Processor Timing Diagram

Reset Operation

The reset signal is a bidirectional signal that allows either the processor or an external signal to reset the system. Figure 28 is a timing diagram for the reset operation. Both the halt and reset lines must be asserted to ensure total reset of the processor.

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. 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 will be reset at the completion of the reset instruction.

Asserting the reset and 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 at least 100ms.

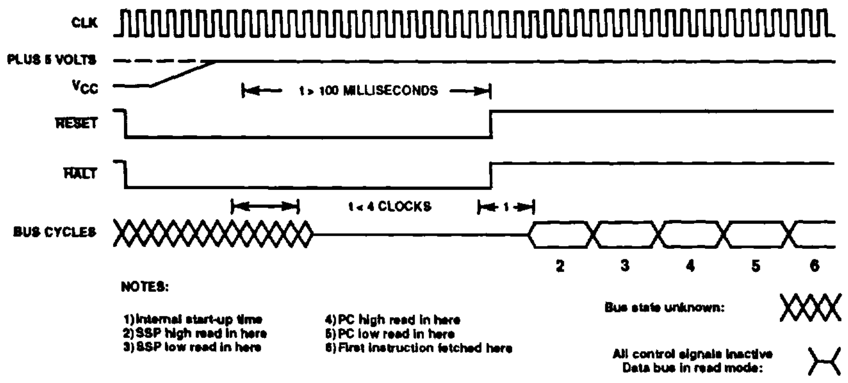


Figure 28. Reset Operation Timing Diagram

Table 16. $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ Assertion Results

CASE NO.	CONTROL SIGNAL	ASSERTED ON RISING EDGE OF STATE		RESULT
		N	N+2	
1	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	A NA NA	S X X	Normal cycle terminate and continue.
2	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	A NA A	S X S	Normal cycle terminate and halt. Continue when $\overline{\text{HALT}}$ removed.
3	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA NA A	A NA S	Normal cycle terminate and halt. Continue when $\overline{\text{HALT}}$ removed.
4	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	X A NA	X S NA	Terminate and take bus error trap.
5	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA A NA	X S A	Terminate and re-run.
6	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	X A A	X S S	Terminate and re-run when $\overline{\text{HALT}}$ removed.
7	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA NA A	X A S	Terminate and re-run when $\overline{\text{HALT}}$ removed.

NOTES:
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

Table 17. $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ Negation Results

CONDITIONS OF TERMINATION	CONTROL SIGNAL	NEGATED ON RISING EDGE OF STATE		RESULTS–NEXT CYCLE
		N	N+2	
Bus Error	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	● or ● or	● ●	Takes bus error trap.
Re-run	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	● or ● or	● ●	Illegal sequence; usually traps to vector number 0.
Re-run	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	● 	 ●	Re-runs the bus cycle.
Normal	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	● ● or	 ●	May lengthen next cycle.
Normal	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	 ● or	● none	If next cycle is started it will be terminated as a bus error.

NOTE:
● = Signal is negated in this bus state.

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 68000 clock. This will assure that when two signals are asserted simultaneously, the required set-up 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 68000. Parameter #48 (see AC Electrical Characteristics for # references) is intended to ensure this operation in a totally asynchronous system, and may be ignored if the above conditions are met.

The preferred bus cycle terminations may be summarized as follows (case numbers refer to Table 16):

Normal Termination:
$\overline{\text{DTACK}}$ occurs first (Case 1).

Halt Termination:
$\overline{\text{HALT}}$ is asserted at the same time or before $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ remains negated (Cases 2 and 3):

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

Re-Run Termination:
$\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ are asserted in lieu of, at the same time, or before $\overline{\text{DTACK}}$ (Cases 6 and 7); $\overline{\text{HALT}}$ must be held at least one cycle after $\overline{\text{BERR}}$. Case 5 indicates $\overline{\text{BERR}}$ may precede $\overline{\text{HALT}}$.

Table 16 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 17 ($\overline{\text{DTACK}}$ is assumed to be negated normally in all cases; for best results, both $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ 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 $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ simultaneously after time out (Case 4).

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 6), 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 4).

Asynchronous Versus Synchronous Operation

Asynchronous Operation
To achieve clock frequency independence at a system level, the 68000 can be used in an asynchronous manner. This entails using only the bus handshake lines ($\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, $\overline{DTACK}$, $\overline{BERR}$, $\overline{HALT}$, and $\overline{VPA}$) to control this data transfer. Using this method, $\overline{AS}$ 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 ($\overline{DTACK}$) to terminate the bus cycle. If no slave responds or the access is invalid, external control logic asserts the $\overline{BERR}$, or $\overline{BERR}$ and $\overline{HALT}$ signal to abort or rerun the bus cycle.

The $\overline{DTACK}$ signal is allowed to be asserted before the data from a slave device is valid on a read cycle. the length of time that $\overline{DTACK}$ 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 $\overline{AS}$ to the assertion of $\overline{DTACK}$. This is because the MPU will insert wait cycles of one clock period each until $\overline{DTACK}$ is recognized.

Synchronous Operation
To allow for those systems which use the system clock as a signal to generate $\overline{DTACK}$ and other asynchronous inputs, the asynchronous input setup time is given as parameter #47. If this setup is met on an input, such as $\overline{DTACK}$, 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 $\overline{DTACK}$ 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 $\overline{DTACK}$ 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.

NOTE: *During an active bus cycle,* $\overline{VPA}$ *and* $\overline{BERR}$ is sampled on every falling edge of the clock starting with S2. $\overline{DTACK}$ 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 $\overline{BERR}$ is asserted in the absence of $\overline{DTACK}$, in which case it will terminate one clock cycle later in S9. $\overline{VPA}$ is sampled only on the third falling edge of the system clock before the rising edge of the E clock.

PROCESSING STATES

This section describes the actions of the 68000 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 68000 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. A special case of the normal state is the stopped state which the processor enters when a stop instruction is executed. In this state, no further 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.

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 be 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 state programs.

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 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.

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 state register are privileged. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE to USP) and move from user stack pointer (MOVE from USP) 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.

Table 18. Bus Cycle Classification

FUNCTION CODE OUTPUT			REFERENCE CLASS
FC2	FC1	FC0	
0	0	0	(Unassigned)
0	0	1	User data
0	1	0	User program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor data
1	1	0	Supervisor program
1	1	1	Interrupt acknowledge

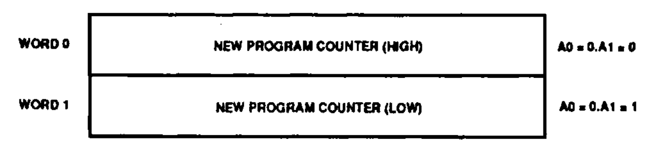
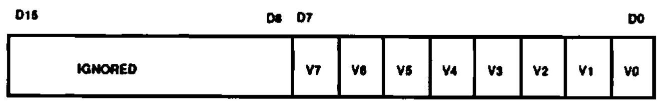


Figure 29. Format of Vector Table Entries



Where:

V7 is the MSB of the vector number.
V0 is the LSB of the vector number.

Figure 30. Vector Number Format

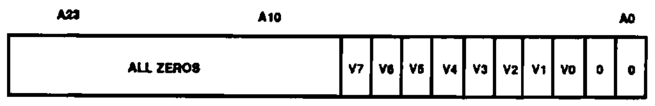


Figure 31. Exception Vector Address Calculation

Privilege State Changes
Once the processor is in the user state and executing instructions, only exceptional processing can change the privilege state. During exception processing, the current 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.

Reference Classification
When the processor makes a reference, it classifies the kind of reference being made, 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 18 lists the classification of references.

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 switches to instruction processing.

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 29), 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 address 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 30) to the processor on data bus lines D0 through D7. The processor translates the vector number into a full 24–bit address, shown in Figure 31. The memory layout for exception vectors is given in Table 19.

Table 19. Exception Vector Table

VECTOR NUMBER(S)	ADDRESS			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	(Unassigned, reserved)
15	60	03C	SD	Uninitialized interrupt vector
16–23*	64	04C	SD	(Unassigned, reserved)
	95	05F		–
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
	191	0BF		–
48–63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		–
64–255	256	100	SD	User interrupt vectors
	1023	3FF		–

NOTE:
* Vector numbers 12, 13, 14, 16 through 23, and 48 through 63 are reserved for future enhancements by Signetics. No user peripheral devices should be assigned these numbers.

As shown in Table19, the memory layout is 512 words long (1024 bytes). It starts at address 0 and proceeds through address 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 64 entries, so user interrupt vectors may overlap at the discretion of the systems designer.

Kinds of Exceptions
Exceptions can be generated by either internal or external causes. The externally generated exceptions are the interrupts and the bus error and reset requests. The interrupts are requests from peripheral devices for processor action while the bus error and reset interrupts 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 fetches from odd addresses, and privilege violations cause exceptions. Tracing behaves like a very high–priority internally–generated interrupt after each instruction execution.

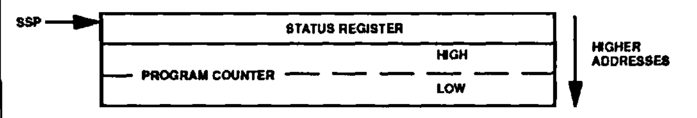


Figure 32. Exception Stack Order (Groups 1 and 2)

Table 20. 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	Exception processing is started by normal instruction execution

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 and classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer as shown in Figure 32. 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 from the address of the instruction which caused the error. 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. 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.

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 preempt 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 bus error and then address 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 aborted. 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 20.

Exception Processing Detailed Discussion

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.

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 and the trace state is forced off. The processor interrupt priority mask is set at level seven. 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.

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, with 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 are detected between instruction executions. 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 sent to the supervisor stack, tracing is suppressed, and the processor priority level is set to the level of the interrupt 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 an 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 program counter and status register on the supervisor stack. 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 33, a timing diagram is given in Figure 34, and the interrupt processing sequence is shown in Figure 35.

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.

Uninitialized Interrupt

An interrupt device asserts $\overline{\text{VPA}}$ or provides an interrupt during an interrupt acknowledge cycle to the 68000. If the vector register has not been initialized, the responding 68000 family peripheral will provide vector 15, the uninitialized interrupt vector. This provides a uniform way to recover from a programming error.

Spurious Interrupt

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

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.

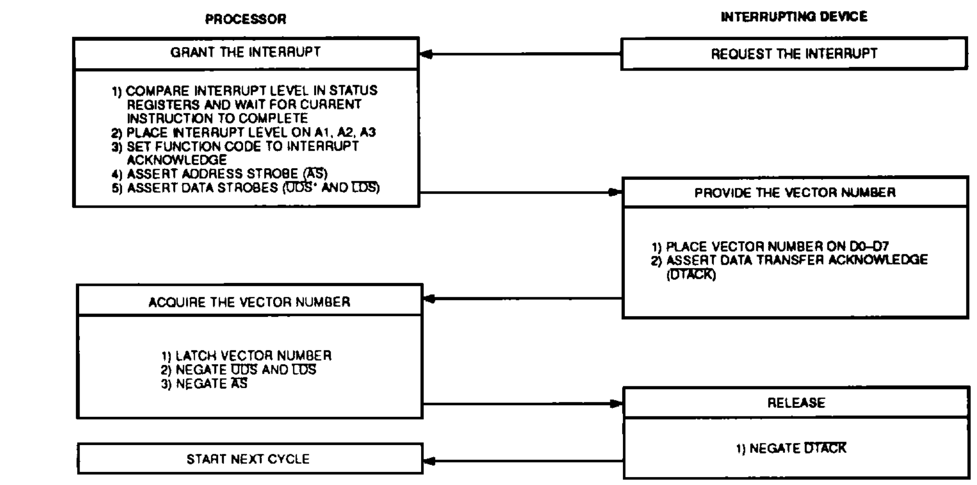
Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a runtime error, which may be an arithmetic overflow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

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

Illegal and Unimplemented Instructions

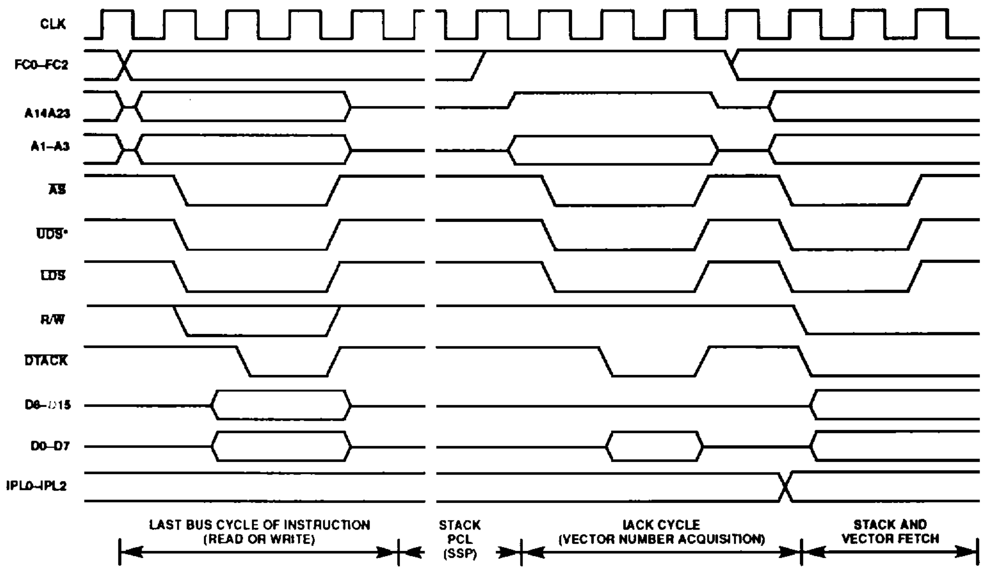
"Illegal instruction" is the term used to refer to any of the word bit patterns which are not the bit pattern of the first word of a legal instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs. Signetics reserves the right to define instructions whose opcodes may be any of the illegal instructions. Three bit patterns will always force an illegal instruction trap on all 68000 family compatible microprocessors. They are: $\overline{\text{4AFA}}$, $\overline{\text{4AFB}}$, and $\overline{\text{4AFC}}$. Two of the patterns, $\overline{\text{4AFA}}$ and $\overline{\text{4AFB}}$, are reserved for Signetics systems products. The third pattern, $\overline{\text{4AFC}}$, is reserved for customer use.

Word patterns with bits 15 through 12 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 in software.


PROCESSOR
INTERRUPTING DEVICE
GRANT THE INTERRUPT
1) COMPARE INTERRUPT LEVEL IN STATUS REGISTERS AND WAIT FOR CURRENT INSTRUCTION TO COMPLETE
2) PLACE INTERRUPT LEVEL ON A1, A2, A3
3) SET FUNCTION CODE TO INTERRUPT ACKNOWLEDGE
4) ASSERT ADDRESS STROBE (AS)
5) ASSERT DATA STROBES (UDS* AND LDS)
REQUEST THE INTERRUPT
PROVIDE THE VECTOR NUMBER
1) PLACE VECTOR NUMBER ON D0–D7
2) ASSERT DATA TRANSFER ACKNOWLEDGE (DTACK)
ACQUIRE THE VECTOR NUMBER
1) LATCH VECTOR NUMBER
2) NEGATE UDS AND LDS
3) NEGATE AS
RELEASE
1) NEGATE DTACK
START NEXT CYCLE

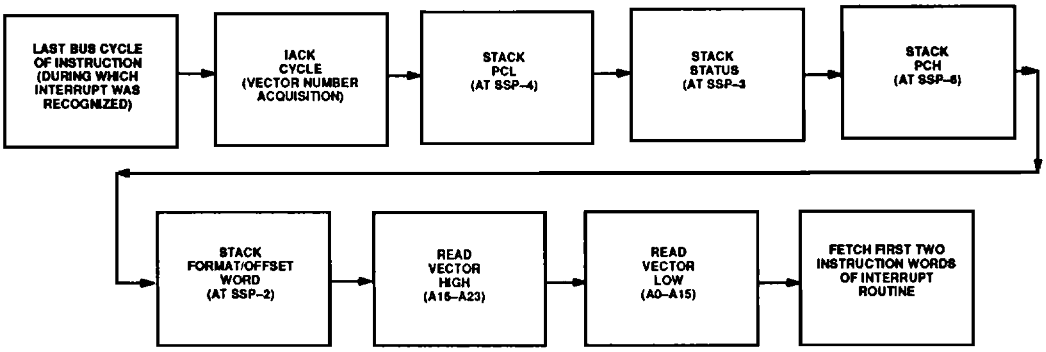
NOTE:
* Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception. The processor does not recognize anything on data lines D8 through D15 at this time.

Figure 33. Vector Acquisition Flowchart



NOTE:
* 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.

Figure 34. Interrupt Acknowledge Cycle Timing Diagram


LAST BUS CYCLE OF INSTRUCTION (DURING WHICH INTERRUPT WAS RECOGNIZED)
IACK CYCLE (VECTOR NUMBER ACQUISITION)
STACK PCL (AT SSP–4)
STACK STATUS (AT SSP–3)
STACK PCH (AT SSP–6)
STACK FORMAT/OFFSET WORD (AT SSP–2)
READ VECTOR HIGH (A16–A23)
READ VECTOR LOW (A0–A15)
FETCH FIRST TWO INSTRUCTION WORDS OF INTERRUPT ROUTINE

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

Figure 35. Interrupt Processing Sequence

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:

STOP	AND Immediate to SR
RESET	EOR Immediate to SR
RTE	OR Immediate to SR
MOVE to SR	MOVE to USP

Tracing
To aid in program development, the 68000 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 but 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 after 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.

Bus Error
Bus error exceptions occur when the external logic requests that a bus error be processed by an exception. The current bus cycle which the processor is making is then aborted. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing.

Exception processing for the bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus error exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. The value saved for the program counter is advanced by some amount, one to five words beyond the address of the first word of the instruction which made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved: whether it was a read or a write, whether or not the processor was processing an instruction, and the classification displayed on the function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a Group 2 exception; the processor is not processing an instruction if it is processing a Group 0 or a Group 1 exception. Figure 36 illustrates how this information is organized on the supervisor stack. Although this information is not sufficient in general to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in vector number two. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, address error, or reset, the processor is halted and all processing ceases. This simplifies the detection of catastrophic system failure, since the processor removes itself from the system rather than destroy any memory contents. Only the $\overline{\text{RESET}}$ pin can restart a halted processor.

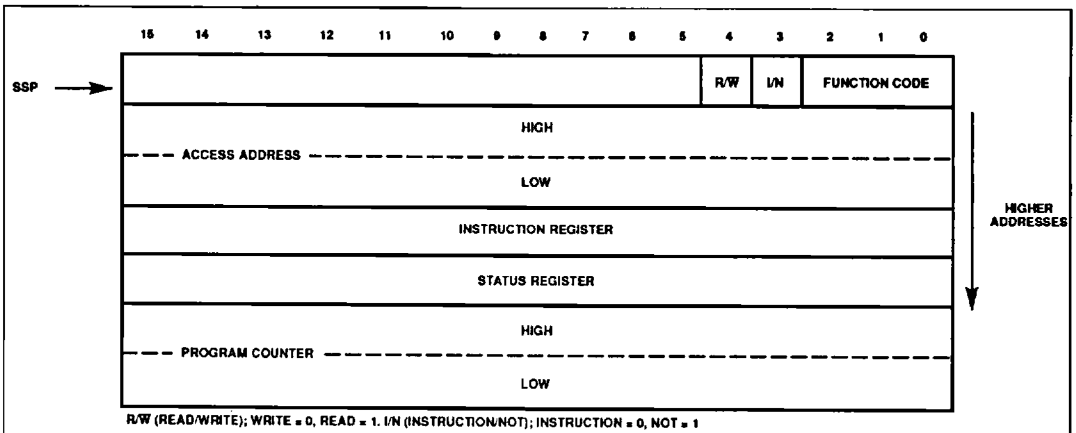
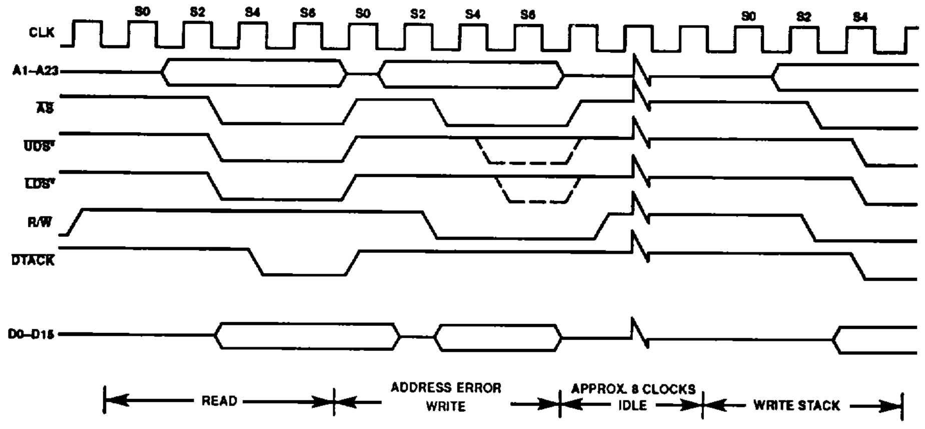


Figure 36 Exception Stack Order (Group 0)

Address Error
Address error exceptions occur when the processor attempts to access a word or a 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 ceases whatever processing it is currently doing and begins exception processing. After the exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted. As shown in figure 37, an address error will execute a short bus cycle followed by exception processing.

INTERFACE WITH SYNCHRONOUS PERIPHERALS

To interface the synchronous peripherals with the asynchronous 68000, the processor modifies its bus cycle to meet the synchronous cycle requirements whenever a synchronous device address is detected. This is possible since both processors use memory mapped I/O. Figure 38 is a flowchart of the interface operation between the processor and synchronous devices.



NOTE:
*UDS and LDS are asserted on mask sets R9M, 8F4, T6E, DL6, CC1, and GN7.

Figure 37. Address Error Timing Diagram

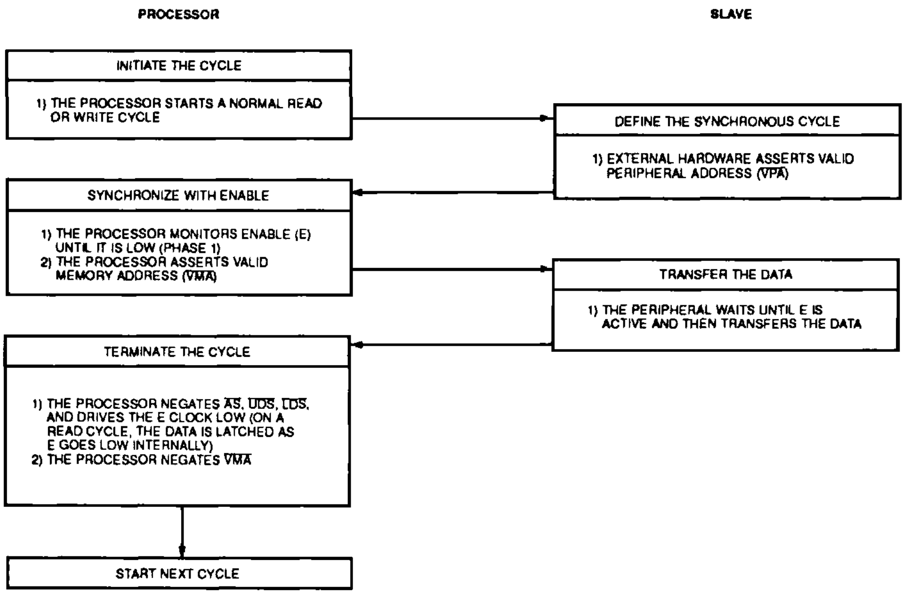

PROCESSOR
SLAVE
INITIATE THE CYCLE
1) THE PROCESSOR STARTS A NORMAL READ OR WRITE CYCLE
DEFINE THE SYNCHRONOUS CYCLE
1) EXTERNAL HARDWARE ASSERTS VALID PERIPHERAL ADDRESS ($\overline{VPA}$)
SYNCHRONIZE WITH ENABLE
1) THE PROCESSOR MONITORS ENABLE (E) UNTIL IT IS LOW (PHASE 1)
2) THE PROCESSOR ASSERTS VALID MEMORY ADDRESS ($\overline{VMA}$)
TRANSFER THE DATA
1) THE PERIPHERAL WAITS UNTIL E IS ACTIVE AND THEN TRANSFERS THE DATA
TERMINATE THE CYCLE
1) THE PROCESSOR NEGATES $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, AND DRIVES THE E CLOCK LOW (ON A READ CYCLE, THE DATA IS LATCHED AS E GOES LOW INTERNALLY)
2) THE PROCESSOR NEGATES $\overline{VMA}$
START NEXT CYCLE

Figure 38. Synchronous Interfacing Flowchart

Data Transfer Operation

Three signals on the processor provide the synchronous 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 synchronous systems. The bus frequency is one tenth of the incoming 68000 clock frequency. The timing of E allows 1MHz peripherals to be used with 8MHz 68000s. 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.

Synchronous cycle timing is given in figures 39, 40, 48, and 49. At state zero (S0) in the cycle, the address bus is in the high–impedance 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–impedance 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 a synchronous device (or an area reserved for synchronous devices) and that the bus should conform to the phase 2 transfer characteristics of the synchronous bus. Valid peripheral address is derived by decoding the address bus, conditioned by the address strobe. Chip select for the synchronous 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 $\overline{VMA}$. Valid memory address is then used as part of the chip select equation of the peripheral. This ensures that the synchronous peripherals are selected and deselected at the correct time. The peripheral now runs its cycle during the high portion of the E signal. Figures 39 and 40 depict the best and worst case synchronous cycle timing. This cycle length is dependent strictly on when $\overline{VPA}$ is asserted in relationship to the E clock.

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–impedance state. During a write cycle, the data bus is put in the high–impedance 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.

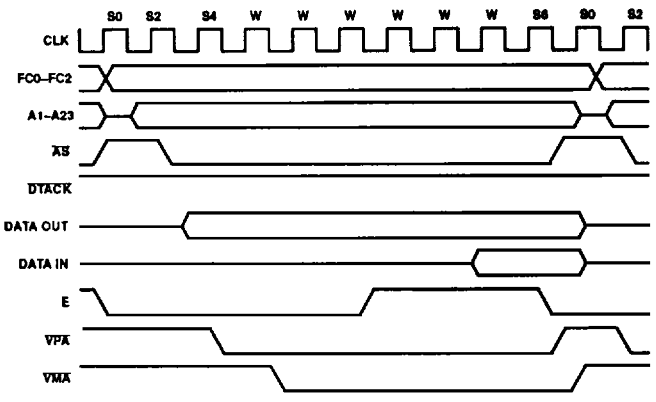


Figure 39. 68000 to Synchronous Peripheral Timing — Best Case

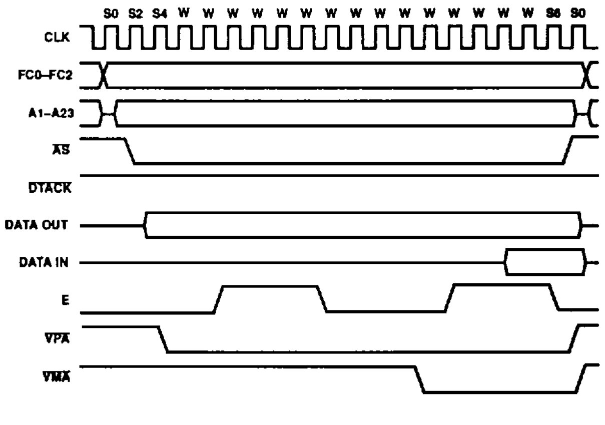


Figure 39. 68000 to Synchronous Peripheral Timing — Worst Case

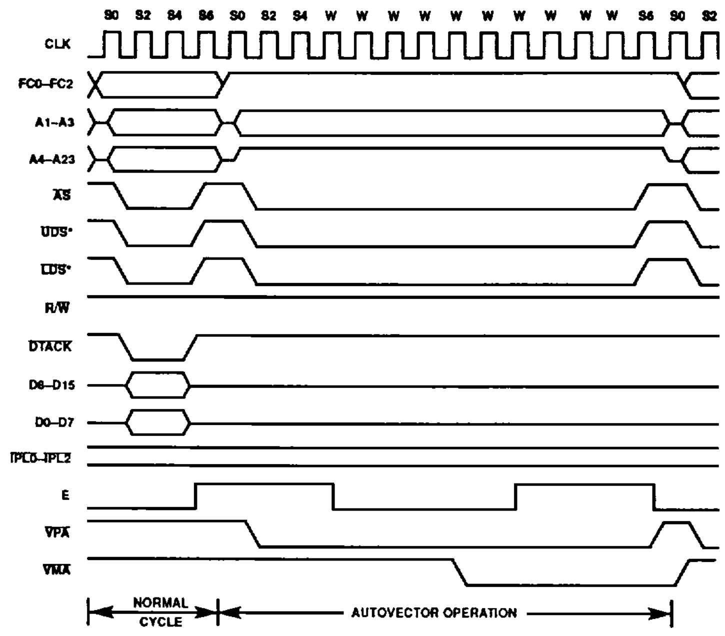
$\overline{\text{DTACK}}$ should not be asserted while $\overline{\text{VPA}}$ is asserted. Notice that the 68000 $\overline{\text{VMA}}$ is active low, contrasted with the active high synchronous $\overline{\text{VMA}}$. This allows the processor to put its buses in the high–impedance state on DMA requests without inadvertently selecting the peripherals.

Interrupt Interface Operation

During an interrupt cycle while the processor is fetching the vector, the $\overline{\text{VPA}}$ is asserted, the 68000 will assert $\overline{\text{VMA}}$ and complete a normal synchronous read cycle as shown in Figure 41. 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 synchronous interrupt sequence. The basic difference is that there are six normal interrupt vectors and one NMI type vector. As with both the synchronous and the 68000'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 $\overline{\text{VMA}}$ is asserted during autovectoring, care should be taken to insure the synchronous peripheral address decoding prevents unintended accesses.



NOTE:
*Although UDS and LDS are asserted, no data is read from the bus during the autovector cycle. The vector number is generated internally.

Figure 41. Autovector Operation Timing Diagram

INSTRUCTION SET AND EXECUTION TIMES

Instruction Set

The following paragraphs provide information about the addressing categories and instruction set of the 68000.

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.

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 21 shows the various categories to which each of the effective address modes belong. Table 22 is the instruction set summary.

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 (writable) 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.

Table 21. Effective Addressing Mode Categories

EFFECTIVE ADDRESS MODES	MODE	REGISTER	ADDRESSING CATEGORIES			
			Data	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	–	–

Table 22. Instruction Set

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 ∧ (Source) → Destination	–	•	•	0	0
ANDI	AND immediate	(Destination) ∧ Immediate Data → Destination	–	•	•	0	0
ANDI to CCR	AND immediate to condition codes	(Source) ∧ CCR → CCR	•	•	•	•	•
ANDI to SR	AND immediate to status register	(Source) ∧ 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 an operand	0 → Destination	–	0	1	0	0
CMP	Compare	(Destination)–(Source)	–	•	•	•	•
CMPA	Compare address	(Destination)–(Source)	–	•	•	•	•
CMPI	Compare immediate	(Destination)–Immediate Data	–	•	•	•	•

Table 22. Instruction Set (Continued)

MNEMONIC	DESCRIPTION	OPERATION	CONDITION CODES				
			X	N	Z	V	C
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	Destination → An	–	–	–	–	–
LINK	Link and allocate	An → –(SP); SP → An; SP + Displacement → 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 code	(Source) → CCR	●	●	●	●	●
MOVE to SR	Move to status register	(Source) → SR	●	●	●	●	●
MOVE from SR	Move from status register	SR → Destination	–	–	–	–	–
MOVE USP	Move user stack pointer	USP → An; An → USP	–	–	–	–	–
MOVEA	Move address	(Source) → Destination	–	–	–	–	–
MOVEM	Move multiple registers	Registers → Destination (Source) → Registers	–	–	–	–	–
MOVEP	Move peripheral data	(Source) → Destination	–	–	–	–	–
MOVEQ	Move quick	Immediate Data → Destination	–	●	●	0	0
MULS	Signed multiply	(Destination) X (Source) → Destination	–	●	●	0	0
MULU	Unsigned multiply	(Destination) X (Source) → Destination	–	●	●	0	0
NBCD	Negate decimal with extend	0–(Destination)$_{10}$–X → 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) ∨ (Source) → Destination	–	●	●	0	0
ORI	Inclusive–OR immediate	(Destination) ∨ Immediate Data → Destination	–	●	●	0	0
ORI to CCR	Inclusive–OR immediate to conditions codes	(Source) ∨ CCR → CCR	●	●	●	●	●
ORI to SR	Inclusive–OR immediate to status register	(Source) ∨ SR → SR	●	●	●	●	●
PEA	Push effective address	Destination → –(SP)	–	–	–	–	–

Table 22. Instruction Set (Continued)

MNEMONIC	DESCRIPTION	OPERATION	CONDITION CODES				
			X	N	Z	V	C
RESET	Reset external devices	–	–	–	–	–	–
ROL, ROR	Rotate (without extend)	(Destination) Rotated by <count> $\rightarrow$ Destination	–	•	•	0	•
ROXL, ROXR	Rotate with extend	(Destination) Rotated by <count> $\rightarrow$ Destination	•	•	•	0	•
RTE	Return from exception	(SP) + $\rightarrow$ SR; (SP) + $\rightarrow$ PC	•	•	•	•	•
RTR	Return and restore condition codes	(SP) + $\rightarrow$ CC; (SP) + $\rightarrow$ PC	•	•	•	•	•
RTS	Return from subroutine	(SP) + $\rightarrow$ 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 $\rightarrow$ Destination else 0's $\rightarrow$ Destination	–	–	–	–	–
STOP	Load status register and stop	Immediate Data $\rightarrow$ SR; STOP	•	•	•	•	•
SUB	Subtract binary	(Destination)–(Source) $\rightarrow$ Destination	•	•	•	•	•
SUBA	Subtract address	(Destination)–(Source) $\rightarrow$ Destination	–	–	–	–	–
SUBI	Subtract immediate	(Destination)–Immediate Data $\rightarrow$ Destination	•	•	•	•	•
SUBQ	Subtract quick	(Destination)–Immediate Data $\rightarrow$ Destination	•	•	•	•	•
SUBX	Subtract with extend	(Destination)–(Source)–X $\rightarrow$ Destination	•	•	•	•	•
SWAP	Swap register halves	Register [31:16]$\leftrightarrow$Register [15:0]	–	•	•	0	0
TAS	Test and set an operand	(Destination) Tested $\rightarrow$ CC; 1 $\rightarrow$ [7] OF Destination	–	•	•	0	0
TRAP	Trap	PC $\rightarrow$ –(SSP); SR $\rightarrow$ –(SSP); (Vector) $\rightarrow$ PC	–	–	–	–	–
TRAPV	Trap on overflow	If V then TRAP	–	–	–	–	–
TST	Test and operand	(Destination) Tested $\rightarrow$ CCC	–	•	•	0	0
UNLK	Unlink	An $\rightarrow$ SP; (SP) + $\rightarrow$ An	–	–	–	–	–

NOTES:

[ ] = bit number	• affected
$\oplus$ logical exclusive OR	– unaffected
$\wedge$ logical AND	0 cleared
$\vee$ logical OR	1 set
~ logical complement	U undefined

Instruction Prefetch

The 68000 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 words are not used.
6. The program counter usually points to the last word fetched from the instruction stream.

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. The timing data is enclosed in parentheses 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 periods includes instruction fetch and all applicable operand fetches and stores.

Effective Address Operand Calculation Timing

Table 23 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. 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 23. Effective Address Calculation Times

ADDRESSING MODE		BYTE,WORD	LONG
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)	8(2/0)
(An)+	Address register indirect with postincrement	4(1/0)	8(2/0)
–(An)	Address register indirect with predecrement	6(1/0)	10(2/0)
d(An)	Address register indirect with displacement	8(2/0)	12(3/0)
d(An,ix)*	Address register indirect with index	10(2/0)	14(3/0)
xxx.W	Absolute short	8(2/0)	12(3/0)
xxx.L	Absolute long	12(3/0)	16(4/0)
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)

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

Move Instruction Execution Times

Tables 24 and 25 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 parentheses as (r/w).

Table 24. Move Byte 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)

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

Table 25. 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)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
(An)	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
(An)+	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
-(An)	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
d(An)	16(4/0)	1(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)

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

Standard Instruction Execution Times
The number of clock periods shown in Table 26 indicates 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 parentheses 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 26 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 26. 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	–	–	158(1/0)+*	–
DIVU	–	–	140(1/0)+*	–
EOR	byte, word	–	4(1/0)***	8(1/1)+
	long	–	8(1/0)***	12(1/2)+
MULS	–	–	70(1/0)+*	–
MULU	–	–	70(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.
◊ Word or long word only.
* Indicates maximum value.
** The base time of 6 clock periods is increased to 8 if the effective address mode is register direct or immediate (effective) address time should also be added).
*** Only available effective address mode is data register direct.
DIVS, DIVU – The divide algorithm used by the 68000 provides less than 10% difference between the best and worst case timings.
MULS, MULU – The multiply algorithm requires 38 + 2n clocks where n is defined as:
MULU: n= the number of ones in the <ea>.
MULS: n = concatenate the <ea> with a zero as the LSB; n is the resultant number of 10 or 01 patterns in the 17-bit source; i.e., worst case happens when the source is $5555.

Immediate Instruction Execution Times

The number of clock periods shown in Table 27 includes the time to fetch immediate operands, perform the operations, store the results, and read the next 9operation. The number of bus read and write cycles is shown in parentheses 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 27, the headings have the following meanings: # = immediate operand, DN = data register operand, An = address register operand, M = memory operand, and SR = status register.

Table 27. Immediate Instruction Execution Times

INSTRUCTION	SIZE	op#, Dn	op#, An	op#, M
ADDI	byte, word	8(2/0)	–	12(2/1)+
	long	16(3/0)	–	20(3/2)+
ADDQ	byte, word	4(1/0)	8(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	16(3/0)	–	20(3/1)+
CMPI	byte, word	8(2/0)	–	8(2/0)+
	long	14(3/0)	–	12(3/0)+
EORI	byte, word	8(2/0)	–	12(2/1)+
	long	16(3/0)	–	20(3/2)+
MOVEQ	long	4(1/0)	–	–
ORI	byte, word	8(2/0)	–	12(2/1)+
	long	16(3/0)	–	20(3/2)+
SUBI	byte, word	8(2/0)	–	12(2/1)+
	long	16(3/0)	–	20(3/2)+
SUBQ	byte, word	4(1/0)	8(1/0)*	8(1/1)+
	long	8(1/0)	8(1/0)	12(1/2)+

NOTE:
+ add effective address calculation time
* word only

Single Operand Instruction Execution Times

Table 28 indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parentheses 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 28. Single Operand Instruction Execution Times

INSTRUCTION	SIZE	REGISTER	MEMORY
CLR	byte, word	4(1/0)	8(1/1)+
	long	6(1/0)	12(1/2)+
NBCD	byte	6(1/0)	8(1/1)+
NEG	byte, word	4(1/0)	8(1/1)+
	long	6(1/0)	12(1/2)+
NEGX	byte, word	4(1/0)	8(1/1)+
	long	6(1/0)	12(1/2)+
NOT	byte, word	4(1/0)	8(1/1)+
	long	6(1/0)	12(1/2)+
Scc	byte, false	4(1/0)	8(1/1)+
	byte, true	6(1/0)	8(1/1)+
TAS	byte	4(1/0)	10(1/1)
TST	byte, word	4(1/0)	4(1/0)+
	long	4(1/0)	4(1/0)+

NOTE:
+ add effective address calculation time

Shift/Rotate Instruction Execution Times

Table 29 indicates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parentheses 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 29. 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)	–

NOTE:
+ add effective address calculation time
n is the shift or rotate count

Bit Manipulation Instruction Execution Times

Table 30 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parentheses 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 30. 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	–	8(1/1)+	–	12(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)	–

NOTES:
+ add effective calculation time
* indicates maximum value

Conditional Instruction Execution Times

Table 31 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parentheses 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 31. Conditional Instruction Execution Times

INSTRUCTION	DISPLACEMENT	BRANCH TAKEN	BRANCH NOT TAKEN
B_{CC}	byte	10(2/0)	8(1/0)
	word	10(2/0)	12(2/0)
BRA	byte	10(2/0)	–
	word	10(2/0)	–
BSR	byte	18(2/2)	–
	word	18(2/2)	–
DB_{CC}	CC true	–	12(2/0)
	CC false	10(2/0)	14(3/0)

Table 32. JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

INSTRUCTION	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)	– –	– –

NOTES:
n is the number of registers to move
* is the size of the index register (ix) and does not affect the instruction's execution time

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

Table 32 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 parentheses as (r/w).

Multi-Precision Instruction Execution Times

Table 33 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 parentheses as (r/w).

In Table 33, the headings have the following meanings: Dn = data register operand and M = memory operand.

Table 33. Multi-Precision Instruction Times

INSTRUCTION	SIZE	op Dn, Dn	op M, M
ADDX	byte, word	4(1/0)	18(3/1)
	long	8(1/0)	30(5/2)
CMPM	byte, word	–	12(3/0)
	long	–	20(5/0)
SUBX	byte, word	4(1/0)	18(3/1)
	long	8(1/0)	30(5/2)
ABCD	byte	6(1/0)	18(3/1)
SBCD	byte	6(1/0)	18(3/1)

Table 34. Miscellaneous Instruction Execution Times

INSTRUCTION	SIZE	REGISTER	MEMORY
ANDI to CCR	byte	20(3/0)	–
ANDI to SR	word	20(3/0)	–
CHK	–	10(1/0)+	–
EORI to CCR	byte	20(3/0)	–
EORI to SR	word	20(3/0)	–
ORI to CCR	byte	20(3/0)	–
ORI to SR	word	20(3/0)	–
MOVE from SR	–	6(1/0)	8(1/1)+
MOVE to CCR	–	12(2/0)	12(2/0)+
MOVE to SR	–	12(2/0)	12(2/0)+
EXG	–	6(1/0)	–
EXT	word	4(1/0)	–
	long	4(1/0)	–
LINK	–	16(2/2)	–
MOVE from USP	–	4(1/0)	–
MOVE to USP	–	4(1/0)	–
NOP	–	4(1/0)	–
RESET	–	132(1/0)	–
RTE	–	20(5/0)	–
RTR	–	20(5/0)	–
RTS	–	16(4/0)	–
STOP	–	4(0/0)	–
SWAP	–	4(1/0)	–
TRAPV	–	4(1/0)	–
UNLK	–	12(3/0)	–

NOTE:
+ add effective address calculation time

Table 35. Move Peripheral Instruction Execution Times

INSTRUCTION	SIZE	REGISTER→ MEMORY	MEMORY → REGISTER
MOVEP	word	16(2/2)	16(4/0)
	long	24(2/4)	25(6/0)

Miscellaneous Instruction Execution Times
Tables 34 and 35 indicate the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycles is shown in parentheses 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.

Exception Processing Execution Times
Table 36 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 parentheses as (r/w).

Table 36. Exception Processing Execution Times

EXCEPTION	PERIODS
Address error	50(4/7)
Bus error	50(4/7)
CHK instruction	44(5/4)+
Divide by zero	42(5/4)
Illegal instruction	34(4/3)
Interrupt	44(5/3)*
Privilege violation	34(4/3)
$\overline{\text{RESET}}$**	40(6/0)
Trace	34(4/3)
TRAP instruction	38(4/4)
TRAPV instruction	34(4/3)

NOTES:
+ add effective address calculation time.
* The interrupt acknowledge cycle is assumed to take four clock periods.
** Indicates the time from when $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are first sampled as negated to when instruction execution starts.

ABSOLUTE MAXIMUM RATINGS

SYMBOL	PARAMETER	RATING	UNIT
V_{CC}	Supply voltage range	–0.3 to +7.0	V
T_{STG}	Storage temperature range	–65 to +150	°C
P_D	Maximum power dissipation, (P_D)	1.75	W
	Lead temperature (soldering 5 seconds)	270	°C
T_J	Junction temperature	150	°C
θ_{JC}	Thermal resistance, junction to case	15, dual-in-line	°C/W
θ_{JA}	Thermal resistance, junction to ambient	30, dual-in-line 50, LCC	°C/W

NOTE:
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}).

RECOMMENDED OPERATING CONDITIONS

SYMBOL	PARAMETER	RATING	UNIT
V_{CC} V_{SS}	Supply voltage	4.75 to 5.25 0	V V
V_{IH}	High level input voltage (logic inputs)	2.0 to V_{CC}	V
V_{IL}	Low level input voltage (logic inputs)	GND to 0.8	V
	Minimum high level output voltage	2.4	V
	Maximum low level output voltage	0.5	V
	Frequency of operation: 68000–6 68000–8 68000–10	 4.0 to 6.0 4.0 to 8.0 4.0 to 10.0	 MHz MHz MHz
T_C	Case operating temperature range (T_C)	−55°C to +110°C	°C

DC ELECTRICAL CHARACTERISTICS V_{CC} = 5.0VDC ±5%; T_C = −55°C TO +110°C (see figures 42, 43, and 44)

SYMBOL	PARAMETER	TEST CONDITIONS −55°C<T_C<+110°C, V_{CC}=5V ±5%		LIMITS Min	Max	UNIT
V_{OH}	High-level output voltage all outputs	I_{OH} = −400µA		2.4		V
V_{OH}	High-level output voltage enable only	R pullup = 1.1 kΩ I_{OH} −400µA		V_{CC}−.75		V
V_{OL}	Low-level output voltage A23-1, FCO-2, $\overline{BG}$	I_{OL} = 3.2mA	V_{CC} = 4.75V		0.5	V
V_{OL}	Low-level output voltage $\overline{HALT}$	I_{OL} = 1.6mA	V_{CC} = 4.75V		0.5	V
V_{OL}	Low-level output voltage $\overline{AS}$, R/$\overline{W}$, D-15-0, $\overline{UDS}$, $\overline{LDS}$, $\overline{VMA}$, E	I_{OL} = 5.3mA	V_{CC} = 4.75V		0.5	V
V_{OL}	Low-level output voltage $\overline{RESET}$	I_{OL} = 5.0mA	V_{CC} = 4.75V		0.5	V
I_{OHZ}	High-impedance (off-state) output current (HIGH)	V_O = 2.4V			20	µA
I_{OLZ}	High-impedance (off-state) output current (LOW)	V_O = 0.4V			−20	µA
I_{IH}	High-level input current; all inputs[1]	V_{IN} = 5.25V			2.5	µA
I_{IL}	Low-level input current; all inputs[1]	V_{IN} = 0			2.5	µA
I_{CC}	Supply current[2]	V_{CC} = 5.25V			333	mA
C_{IN}	Capacitance[3]	V_{IN} = 0V frequency = 1MHz			20	pF

Power Considerations

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

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

$P_D = K \div (T_J + 273°C)$ (2)

Solving equations 1 and 2 for K gives:

$K = T_D \bullet (T_A + 273°C) + \bullet_{JA} \bullet P_D 2$ (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.

Figure 45 illustrates the graphic solution to the equations, given above, for the specification power dissipations of 1.50 and 1.75 watts over the ambient temperature range of −55°C to 125°C using an average θ_{JA} of 40°C/W to represent various 68000 packages.

However, actual θ_{JA}'s in the range of 30°C to 50°C/W only change the curves slightly.

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:

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

θ_{JC} is a 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.

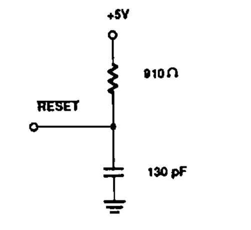


Figure 42. RESET Test Load

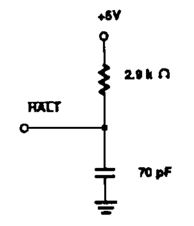


Figure 43. HALT Test Load

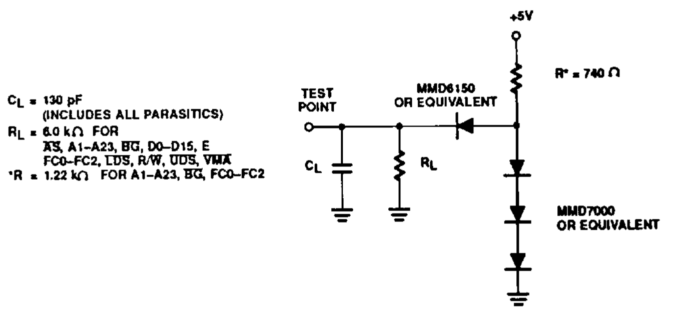

+5V
R* = 740 Ω
TEST POINT
MMD6150 OR EQUIVALENT
C_L = 130 pF (INCLUDES ALL PARASITICS)
R_L = 6.0 kΩ FOR AS, A1–A23, BG, D0–D15, E FC0–FC2, LDS, R/W, UDS, VMA
*R = 1.22 kΩ FOR A1–A23, BG, FC0–FC2
C_L
R_L
MMD7000 OR EQUIVALENT

Figure 44. Test Loads

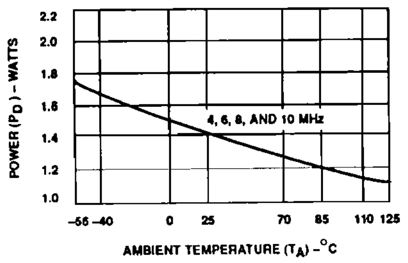


Figure 45. 68000 Power Dissipation (P_D) vs Ambient Temperature (T_A)

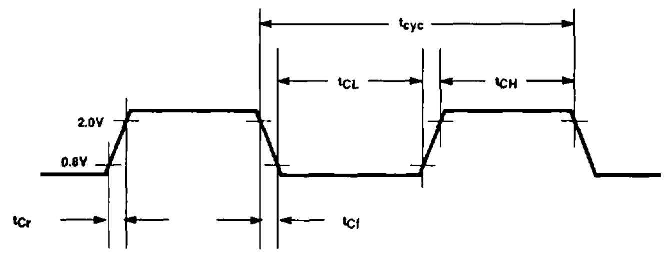
Values for thermal resistance presented in this data sheet are provided for design purposes only. Thermal measurements are complex and dependent on procedure and setup. User derived values for thermal resistance may differ.

AC ELECTRICAL CHARACTERISTICS — Clock Timing (see Figure 46)

CHARACTERISTIC	SYMBOL	6MHz		8MHz		10MHz		UNIT
		Min	Max	Min	Max	Min	Max	
Frequency of operation	F		6.0		8.0		10.0	MHz
Cycle time	t_{cyc}	167	250	125	250	100	250	ns
Clock pulse width	t_{CL} t_{CH}	75 75	125 125	55 55	125 125	45 45	125 125	ns ns
Rise and fall times	t_{Cr} t_{Cf}	– –	10 10	– –	10 10	– –	10 10	ns ns

37. Maximum Power Dissipation by Package Type Modes

PACKAGE TYPE	TEMPERATURE (°C)	MAXIMUM POWER DISSIPATION (WATTS) PER FREQUENCY (MHz)	
		6/8MHz	10MHz
Ceramic	0 to 70 –40 to 85	1.50 1.65	1.50 1.65
Plastic	0 to 70 –40 to 85	1.50 1.65	1.50 1.65
Pin grid array	0 to 70 –40 to 85	1.50 1.65	1.50 1.65
Plastic LCC	0 to 70	1.50	1.50



NOTE:

Timing measurements are referenced to and from a low voltage of 0.8V and a high voltage of 2.0V unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8V and 2V.

Figure 46. Clock Input Timing Diagram

AC ELECTRICAL CHARACTERISTICS[1]—Read and Write Cycles $-55°C \le T_C \le +110°C$ (See Figures 47 and 48)

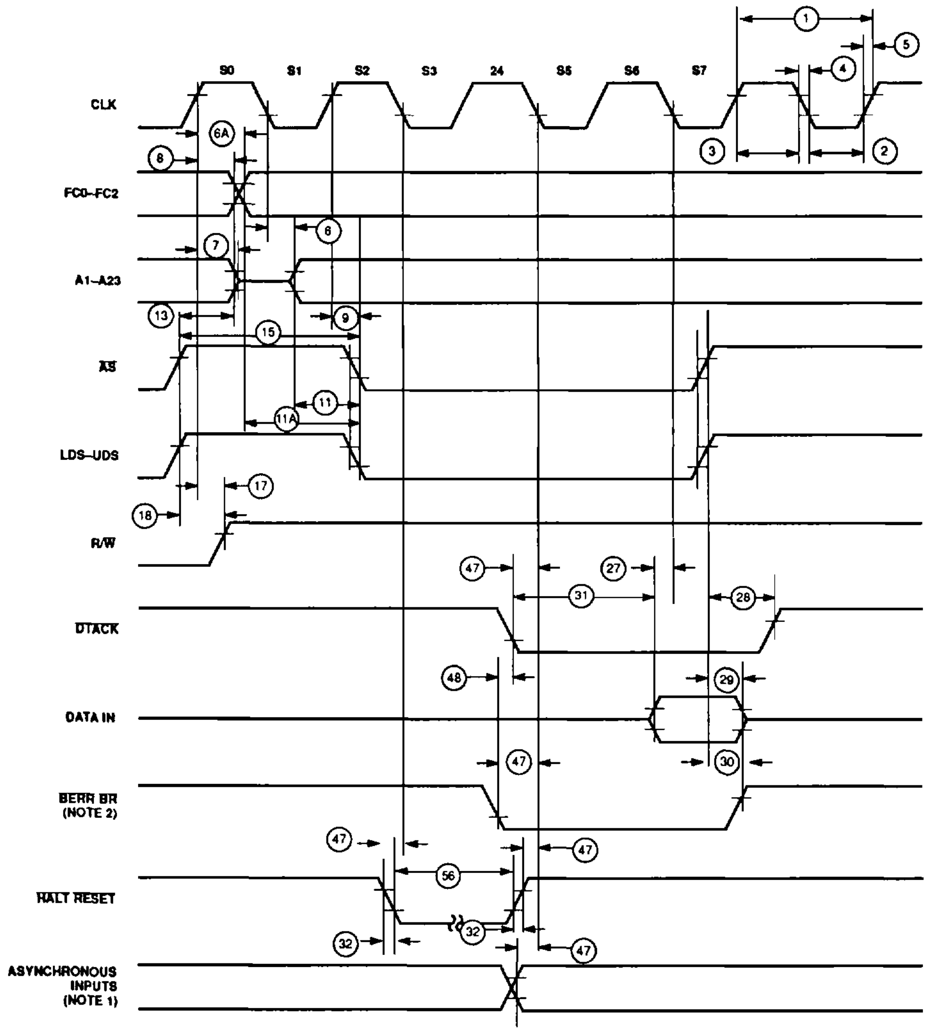
NO.	SYMBOL	CHARACTERISTIC	6MHz		8MHz		10MHz		UNIT
			Min	Max	Min	Max	Min	Max	
1	t_{CYC}	Clock Period	167	250	125	250	100	250	ns
2	t_{CL}	Clock width low	75	125	55	125	45	125	ns
3	t_{CH}	Clock width high	75	125	55	125	45	125	ns
4	t_{Cf}	Clock fall time	–	10	–	10	–	10	ns
5	t_{Cr}	Clock rise time	–	10	–	10	–	10	ns
6	t_{CLAV}	Clock low to address valid	–	80	–	70	–	60	ns
6A	t_{CHFCV}	Clock high to FC valid	–	80	–	70	–	60	ns
7[7]	t_{CHAZx}	Clock high to address data high impedance (maximum)	–	100	–	80	–	70	ns
8[8]	t_{CHAZn}	Clock high to address/FC invalid (minimum)	0	–	0	–	0	–	ns
9[9]	t_{CHSLx}	Clock high to $\overline{AS}$, $\overline{DS}$ low (maximum)	–	70	0	60	0	55	ns
10[10]	t_{CHSLn}	Clock high to $\overline{AS}$, $\overline{DS}$ low (minimum)	–	–	–	–	–	–	ns
11[11]	t_{AVSL}	Address valid to $\overline{AS}$, $\overline{DS}$ (read) low/$\overline{AS}$ write	35	–	30	–	20	–	ns
11A[7,11]	t_{FCVSL}	FC valid to $\overline{AS}$, $\overline{DS}$ (read) low/$\overline{AS}$ (write)	70	–	60	–	50	–	ns
12[9]	t_{CLSH}	Clock low to $\overline{AS}$, $\overline{DS}$ high	–	80	–	70	–	55	ns
13[7,11]	t_{SHAZ}	$\overline{AS}$, $\overline{DS}$ high to address/FC invalid	40	–	30	–	20	–	ns
14[7,11]	t_{SL}	$\overline{AS}$, $\overline{DS}$ width low (read)/$\overline{AS}$ (write)	337	–	240	–	195	–	ns
14A[7,11]	t_{DWPW}	$\overline{DS}$ width low (write)	170	–	115	–	95	–	ns
15[7,11]	t_{SH}	$\overline{AS}$, $\overline{DS}$ width high	180	–	150	–	105	–	ns
16[7]	t_{CHSZ}	Clock high to $\overline{AS}$,$\overline{DS}$ high impedance	–	100	–	80	–	70	ns
17[7,11]	t_{SHRH}	$\overline{AS}$, $\overline{DS}$ high to R/$\overline{W}$ high	50	–	40	–	20	–	ns
18[9]	t_{CHRHx}	Clock high to R/$\overline{W}$ high (maximum)	–	80	–	70	–	60	ns
19[10]	t_{CHRHn}	Clock high to R/$\overline{W}$ high (minimum)	–	–	–	–	–	–	ns
20[9]	t_{CHRL}	Clock high to R/$\overline{W}$ low	–	80	–	70	–	60	ns
20A[7,15]	t_{ASRV}	$\overline{AS}$ low to R/$\overline{W}$ valid	–	20	–	20	–	20	ns
21[7,11]	t_{AVRL}	Address valid to R/$\overline{W}$ low	25	–	20	–	0	–	ns
21A[7,11]	t_{FCVRL}	FC valid to R/$\overline{W}$ low	70	–	60	–	50	–	ns
22[7,11]	t_{RLSL}	R/$\overline{W}$ low to $\overline{DS}$ low (write)	140	–	80	–	50	–	ns
23	t_{CLDO}	Clock low to data out valid	–	80	–	70	–	55	ns
24[13]	t_{CHRZ}	Clock high to R/$\overline{W}$, $\overline{VMA}$ high impedance	–	–	–	–	–	–	ns
25[7,11]	t_{SHDO}	$\overline{DS}$ high to data out invalid	40	–	30	–	20	–	ns
26[11]	t_{DOSL}	Data out valid to $\overline{DS}$ low (write)	35	–	30	–	20	–	ns
27[7,17]	t_{DICL}	Data in to clock low (setup time)	25	–	15	–	10	–	ns
28[7,11]	t_{SHDAH}	$\overline{AS}$, $\overline{DS}$ high to $\overline{DTACK}$ high	0	167	0	125	0	100	ns
29[7]	t_{SHDI}	$\overline{DS}$ high to data invalid (hold time)	0	–	0	–	0	–	ns
30[7]	t_{SHBEH}	$\overline{AS}$, $\overline{DS}$ high to $\overline{BERR}$ high	0	–	0	–	0	–	ns
31[7,11,17]	t_{DALDI}	$\overline{DTACK}$ low to data valid (Setup time)	–	120	–	90	–	65	ns
32[7]	$t_{RH,f}$	$\overline{HALT}$ and $\overline{RESET}$ input transition time	0	200	0	200	0	200	ns
33	t_{CHGL}	Clock high to $\overline{BG}$ low	–	80	–	70	–	60	ns
34[7]	t_{CHGH}	Clock high to $\overline{BG}$ high	–	80	–	70	–	60	ns
35[7]	t_{BRLGL}	$\overline{BR}$ low to $\overline{BG}$ low	1.5	3.5	1.5	3.5	1.5	3.5	Clk. per.

AC ELECTRICAL CHARACTERISTICS[1]—Read and Write Cycles (Continued)

NO.	SYMBOL	CHARACTERISTIC	6MHz		8MHz		10MHz		UNIT
			Min	Max	Min	Max	Min	Max	
36[7,18]	t_{BRHGH}	$\overline{BR}$ high to $\overline{BG}$ high	1.5	+100	1.5	+90	1.5	+80	Clk. per.
37[7]	t_{GALGH}	$\overline{BGACK}$ low to $\overline{BG}$ low	1.5	NS	1.5	NS	1.5	NS	Clk. per.
37A[7]	t_{BGKBR}	$\overline{BGACK}$ low to $\overline{BR}$ high (to prevent rearbitration)	20ns	1.5	20ns	1.5	20ns	1.5	Clk. per.
38[7]	t_{GLZ}	$\overline{BG}$ low to bus high impedance (with $\overline{AS}$ high)	–	100	–	80	–	70	ns
39[7]	t_{GH}	$\overline{BG}$ width high	1.5	–	1.5	–	1.5	–	Clk. per.
40	t_{CLVML}	Clock low to $\overline{VMA}$ low	–	80	–	70	–	70	ns
41	t_{CLC}	Clock low to E transition	–	85	–	70	–	55	ns
42[7]	$t_{Er,f}$	E output rise and fall time	–	25	–	25	–	25	ns
43[7]	t_{VMLEH}	$\overline{VMA}$ low to E high	240	–	200	–	150	–	ns
44[7]	t_{SHVPH}	$\overline{AS}$, $\overline{DS}$ high to $\overline{VPA}$ high	0	160	0	120	0	90	ns
45[7]	t_{ELAI}	E low to address /$\overline{VMA}$/FC invalid	35	–	30	–	10	–	ns
46[7]	t_{BGL}	$\overline{BGACK}$ width	1.5	–	1.5	–	1.5	–	Clk. per.
47[17]	t_{ASI}	Asynchronous input setup time	25	–	20	–	20	–	ns
48[7,14]	t_{BELDAL}	$\overline{BERR}$ low to $\overline{DTACK}$ low	25	–	20	–	20	–	ns
49[7]	t_{ELSI}	E low to $\overline{AS}$, $\overline{DS}$ invalid	–80	+80	–70	+70	–55	+55	ns
50[7]	t_{EH}	E width high	600	–	450	–	350	–	ns
51[7]	E width low	t_{EL}	900	–	700	–	550	–	ns
52[7,16]	t_{CIEHX}	E extended rise time	–	–	–	–	–	–	ns
53[7]	t_{CHDO}	Data hold from clock high	0	–	0	–	0	–	ns
54[5,7]	t_{ELDOZ}	Data hold from E low (write)	40	–	30	–	20	–	ns
55[7]	t_{RLDO}	R/$\overline{W}$ to data bus impdeance change	35	–	30	–	20	–	ns
56[7,12]	t_{HRPW}	$\overline{HALT}$/$\overline{RESET}$ pulse width	10	–	10	–	10	–	Clk. per.
57	t_{GABD}	$\overline{BGACK}$ high to control bus driven	1.5	–	1.5	–	1.5	–	Clk. per.
58[18]	t_{GHBD}	$\overline{BG}$ high to control bus driven	1.5	–	1.5	–	1.5	–	Clk. per.

NOTES:
1. After V_{CC} has been applied for 100 ms.
2. All outputs unloaded except for load capacitance. Clock should be either 4MHz or F_{MAX}. Low; HALT, RST, (Part is held in reset). High; $\overline{DTACK}$, BR, BGACK, IPLQ-2, VPA, BERR.
3. Guaranteed but not tested.
4. $V_{CC} = 5V \pm 5\%$
5. After V_{CC} has been applied for 100ms.
6. All outputs unloaded except for load capacitance. Clock should be either 4MHz or F_{MAX}. Low; $\overline{HALT}$, $\overline{RST}$, (Part is held in reset). High; $\overline{DTACK}$, BR, BGACK, IPLQ–2, VPA, BERR.
7. As a minimum, tested initially and for process or design changes only.
8. For invalid, as a minimum, tested initially and for process and design changes only.
9. For a loading capacitance of less than or equal to 50pF, subtract 5ns from the values given in the maximum column.
10. Combined with the above parameter. Previous specification of 0ns was theoretical and not attainable.
11. Actual value depends on clock period.
12. For power up, the MPU must be held in RESET state for 100ms to all stabilization of on–chip circuitry. After the system is powered up, 56 refers to the minimum pulse width required to reset the system.
13. Combined with 16, control bus specification
14. If 47 is satisfied for both $\overline{DTACK}$ and $\overline{BERR}$, 48 may be 0ns.
15. When $\overline{AS}$ and R/$\overline{W}$ are equally loaded (±20%), subtract 10ns from the values given in these columns.
16. Deleted, useful only if E clock used to drive clock input on MC6809 Microprocessor.
17. If the asynchronous setup time (47) requirements are satisfied, the $\overline{DTACK}$ low to data setup time (#31) requirement can be ignored. the data must only satisfy the data in to clock–low setup time (#27) for the following cycle.
18. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BTACK}$.

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.

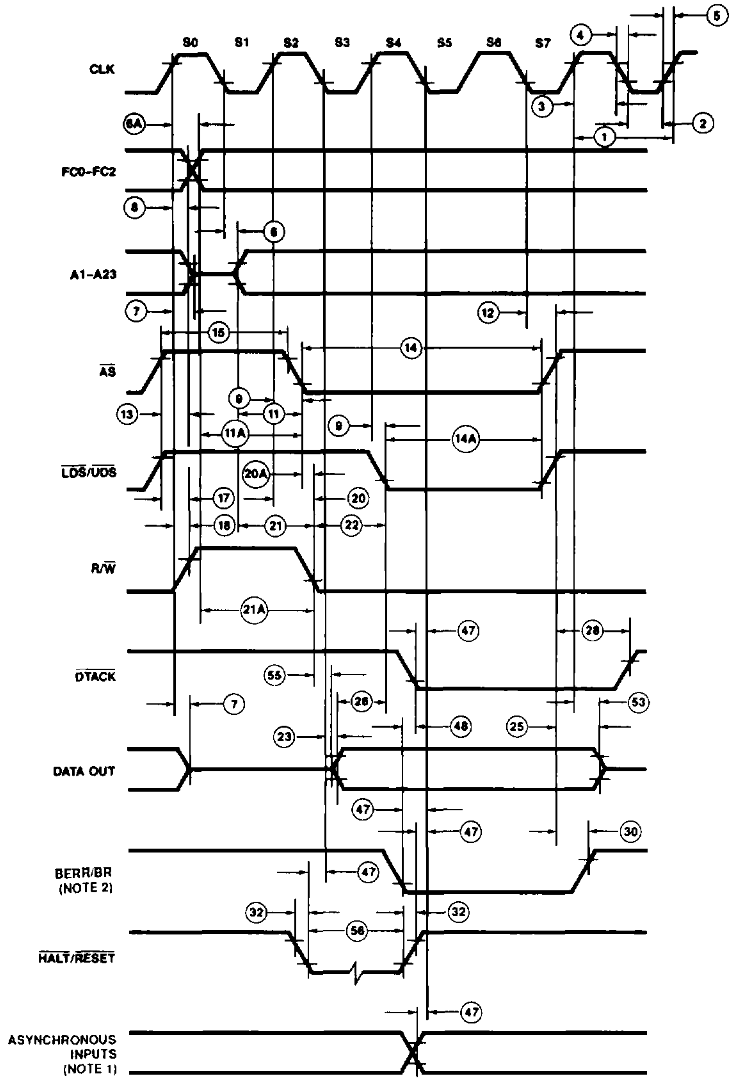


NOTES:

1. Setup time for the asynchronous inputs $\overline{\text{BGACK}}$, $\overline{\text{IPL0–2}}$, and $\overline{\text{VPA}}$ 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 0.8V and a high voltage of 2.0V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8V and 2.0V.

Figure 47. Write Cycle Timing Diagram

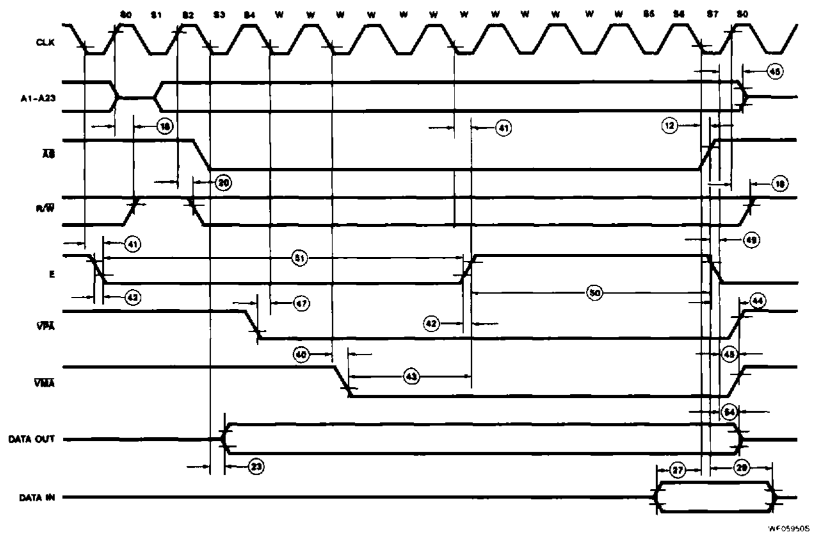
These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



NOTES:

1. Timing measurements are referenced to and from a low voltage of 0.8V and a high voltage of 2.0V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8V and 2.0V.
2. Because of loading variations, R/W̄ may be valid after ĀS̄ even though both are initiated by the rising edge of S2 (Specification 20A).

Figure 48. Write Cycle Timing Diagram (Continued)



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

Figure 49. 68000 to M6800 Peripheral Timing Diagram — Best Case

Figures 50, 51, and 52 depict the three bus arbitration cases that can arise. Figure 50 shows the timing where $\overline{AS}$ is negated when the processor asserts $\overline{BG}$ (idle bus case). Figure 51 shown the timing where $\overline{AS}$ is asserted when the processor asserts $\overline{BG}$ (active bus case). Figure 52 shows the timing where more than one bus master are requesting the bus. Refer to **Bus Arbitration** for a complete discussion of bus arbitration.

The waveforms shown in Figures 50, 51, and 52 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.

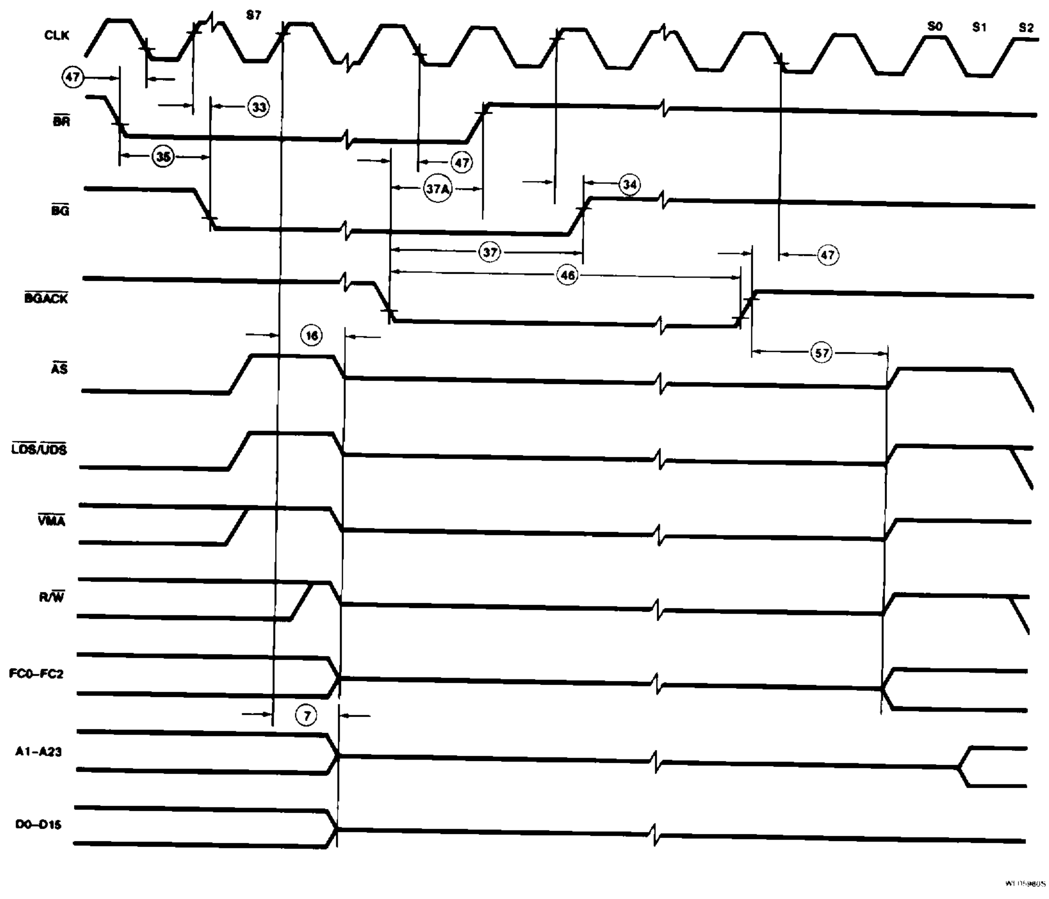


Figure 50. Bus Arbitration Timing Diagram — Active Bus Case

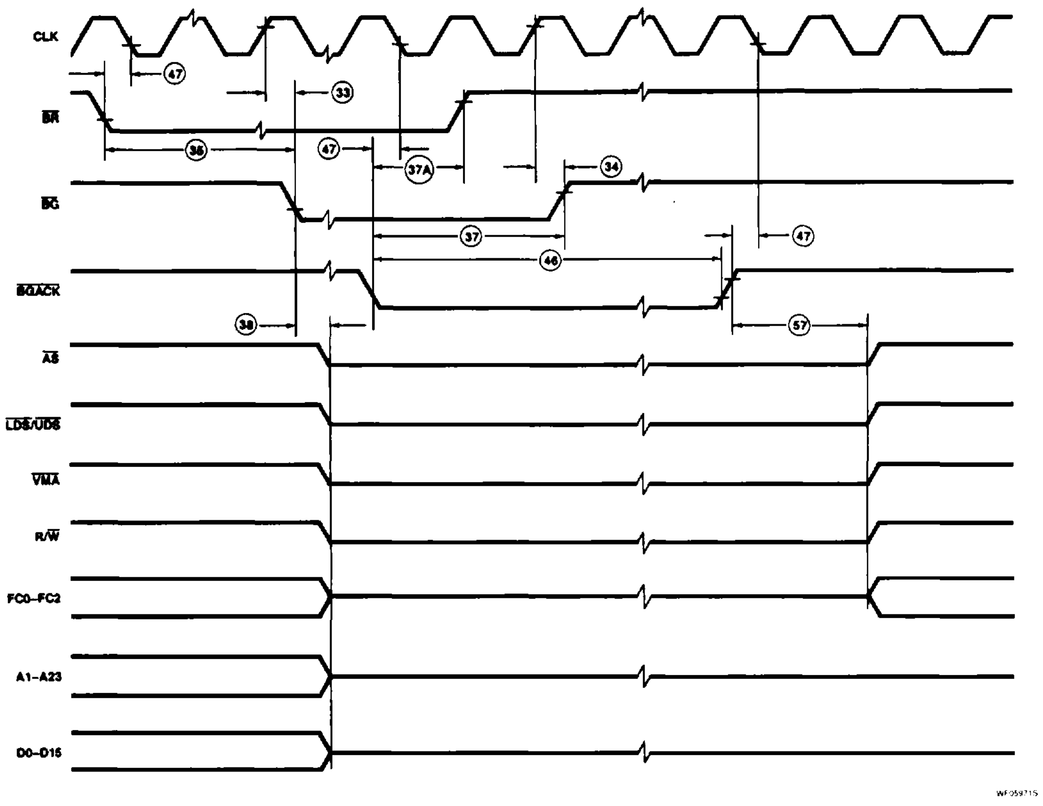


Figure 51. Bus Arbitration Timing Diagram — Idle Bus Case

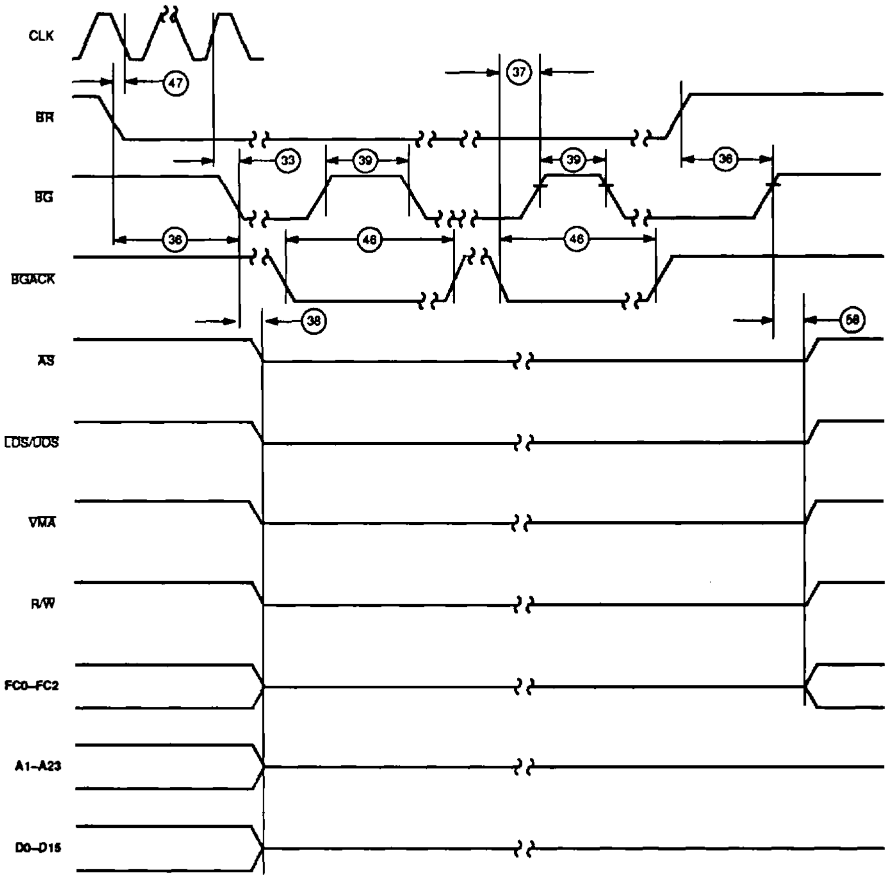


Figure 52. Bus Arbitration Timing Diagram — Multiple Bus Requests