T-49-17-14

16-BIT MICROPROCESSOR
WITH 8-BIT DATA BUS
TMP68008P-8 / TMP68008P-10

Package Type : P: Plastic DIP

1. INTRODUCTION

The TMP68008 is a member of the TLCS-68000 Family of advancedmicroprocessors. This device allows the design of costeffective systems using 8-bit data buses while providing the benefits of a 32-bit microprocessor architecture. The performance of the TMP68008 is greater than any 8-bit microprocessor and superior to several 16-bit microprocessors.

The resources available to the TMP68008 user consist of the following:

- 17 32-bit Data and Address Registers
- 56 Basic Instruction Types
- Extensive Exception Processing
- Memory Mapped I/O
- 14 Addressing Modes
- Complete Code Compatibility with the TMP68000

A system implementation based on an 8-bit data bus reduces system cost in comparison to 16-bit systems due to amore effective use of components and the fact that byte-wide memories and peripherals can be used much more effectively. In addition, the non-multiplexed address and data buses eliminate the need for external demultiplexers, thus further simplifying the system.

The TMP68008 has full code compatibility (source and object) with the TMP68000 which allows programs to be run oneither MPU, depending on performance requirements and cost objectives.

The TMP68008 is available in a 48-pin dual-in-line package (plastic) and a 52-pin quad plastic package. Among the four additional pins of the 52-pin package, two additional address lines are included beyond the 20 address lines of the 48-pin package. The address range of the TMP68008 is one or four megabytes with the 48 or 52-pin package, respectively.

The large non-segmented linear address space of the TMP68008 allows large modular programs to be developed and executed efficiently. A large linear address space allows program segment sizes to be determined by the application rather than forcing the designer to adopt an arbitrary segment size without regard to the application's individual requirements.

The programmer's model is identical to that of the TMP68000, as shown in Figure

1.1, with seventeen 32-bit registers, a 32-bit program counter, and a 16-bit status register. The first eight registers (D0~D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) operations. The second set of seven registers (A0~A6), the user stack pointer (A7), and the system stack pointer (A7') may be used as software stack pointers and base address registers. In addition, the registers may be used for some simple word and long word data operations. All of the 17 registers may be used as index registers.

While all of the address registers can be used to create stacks and queues, the A7 address register, by convention, is used as the system stack pointer. Supplementing this convention is another address register, A7; also referred to as the system stack pointer. This powerful concept allows the supervisor made and user mode of the TMP68008 to each have their own system stack pointer (consistently referred to as SP) without needing to move pointers for each context of use when the mode is switched.

The system stack pointer (SP) is either the supervisor stack pointer (A7' = SSP) or the user stack point (A7 = USP), depending on the state of the S bit in the status register. If the S bit is set, indicating that the processor is in the supervisor state, when the SSP is the active system stack pointer and the USP is not used. If the S bitis clear, indicating that the processor is in the user state, then the USP is the active system stack pointer and the SSP is protected from user modification.

The status register, shown in Figure 1.2, may be considered as two bytes: the user byte and the system byte. The user byte contains five bits defining the overflow (V), zero (Z), negative (N), carry (C), and extended (X) condition codes. The system byte contains five defined bits. Three bits are used to define the current interrupt priority; any interrupt level higher than the current mask level will be recognized. (Note that level 7 interrupts are non-maskable — that is, level 7 interrupts are always processed.) Tow additional bits indicate whether the processor is in the trace (T) mode and/or in the supervisor (S) state.

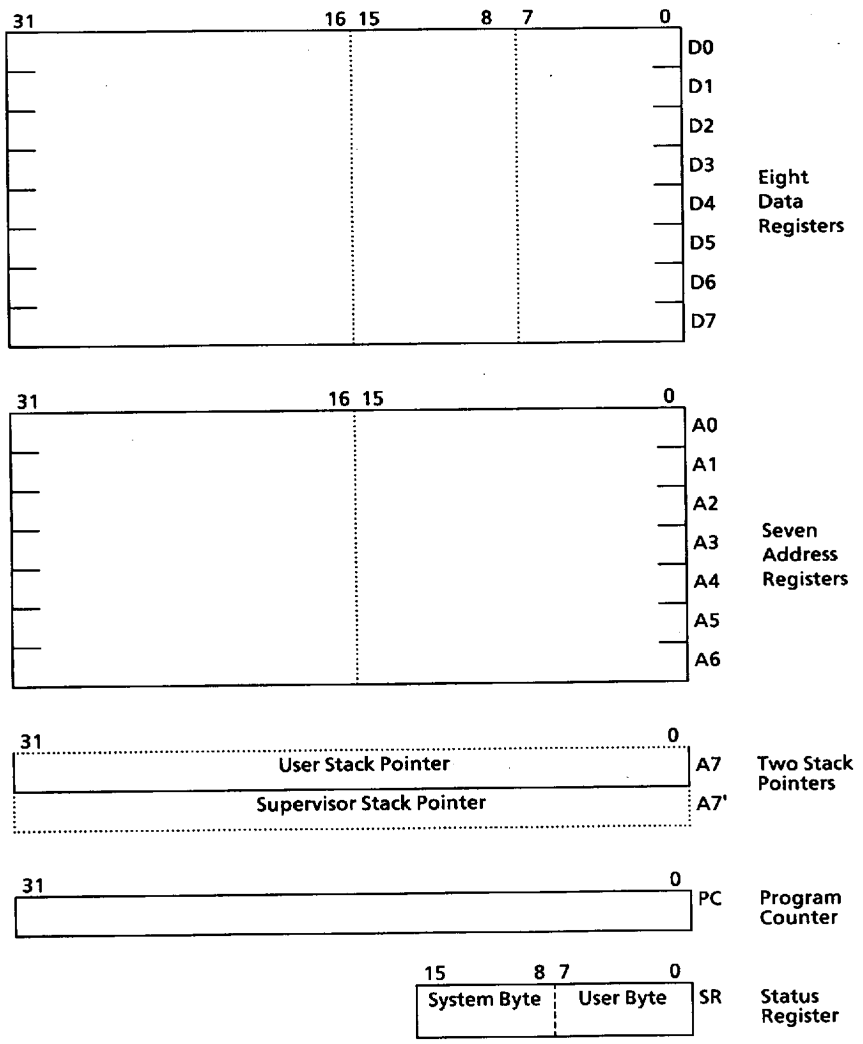


Figure 1.1 Programming Model

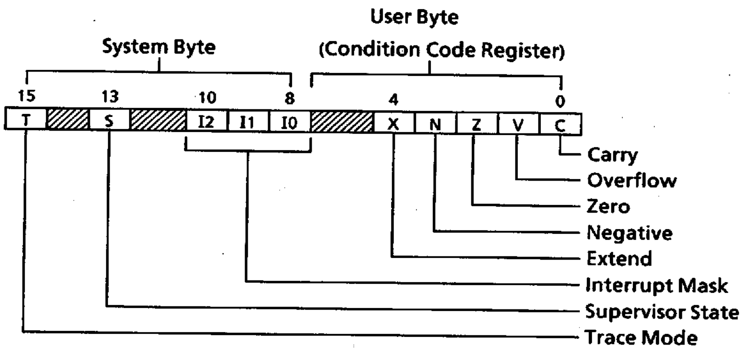


Figure 1.2 Status Register

1.1 DATA TYPES AND ADDRESSING MODES

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

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

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

Most instructions can use any of the 14 addressing modes which are listed in Table 1.1. These addressing modes consist of six basic types:

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

The register indirect addressing modes also have the capability to perform postincrementing, predecrementing, offsetting, and indexing. The program counter relative mode may be used in combination with indexing and offseting for writing relocatable programs.

Table 1.1 Addressing Modes

Addressing Modes	Syntax
Register Direct Addressing	
Data Register Direct	Dn
Address Register Direct	An
Absolute Data Addressing	
Absolute Short	Abs, W
Absolute Long	Abs, L
Program Counter Relative Addressing	
Relative with Offset	d16 (PC)
Relative with Index Offset	d8(PC, Xn)
Register Indirect Addressing	
Register Indirect	(An)
Postincrement Register Indirect	(An) +
Predecrement Register Indirect	- (An)
Register Indirect with Offset	d16 (An)
Indexed Register Indirect with Offset	d8 (An, Xn)
Immediate Data Addressing	
Immediate	#xxx
Quick Immediate	#1~#8
Implied Addressing	
Implied Register	SR / USP / SP / PC

Notes :

Dn	=	Data Register
An	=	Address Register
Xn	=	Address or Data Register used as Index Register
SR	=	Status Register
PC	=	Program Counter
SP	=	Stack Pointer
USP	=	User Stack Pointer
( )	=	Contents of
d8	=	8-Bit Offset (Displacement)
d16	=	16-Bit Offset (Displacement)
#xxx	=	Immediate Data

1.2 INSTRUCTION SET OVERVIEW

The TMP68008 is completely code compatible with the TMP68000. This means that programs developed for the TMP68000 will run on the TMP68008 and vice versa. This applies equally to either source code or object code.

The instruction set was designed to minimize the number of mnemonics remembered by the programmer. To further reduce the programmer's burden,the addressing modes are orthogonal.

The instruction set,shown in Table 1.2,forms a set of programming tools that include all processor functions to perform data movement,integer arithmetic,logical operations,shift and rotate operations,bit manipulation,BCD operations,and both program and system control. Some additional instructions are variations or subests of these and appear in Table 1.3.

Table 1.2 Instruction Set (1/2)

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
Bcc	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
DBcc	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

Table 1.2 Instruction Set (2/2)

Mnemonic	Description
MOVE	Move
MOVEM	Mode Address
MOVEP	Move Peripheral Data
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
Scc	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

Table 1.3 Variations of Instruction Types

Instruction Type	Variation	Description
ADD	ADD	Add
	ADDA	Add Address
	ADDQ	Add Quick
	ADDI	Add Immediate
	ADDX	Add with Extend
AND	AND	Logical And
	ANDI	And Immediate
	ANDI to CCR	And Immediate to Condition Codes
	ANDI to SR	And Immediate to Status Register
CMP	CMP	Compare
	CMPA	Compare Address
	CMPM	Compare Memory
	CMPI	Compare Immediate
EOR	EOR	Exclusive Or
	EORI	Exclusive Or Immediate
	EORI to CCR	Exclusive Or Immediate to Condition Codes
	EORI to SR	Exclusive Or Immediate to Status Register
MOVE	MOVE	Move
	MOVEA	Move Address
	MOVEM	Move Multiple Registers
	MOVEP	Move Peripheral Data
	MOVEQ	Move Quick
	MOVE from SR	Move from Status Register
	MOVE to SR	Move to Status Register
	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

2. DATA ORGANIZATION AND ADDRESSING CAPABILITIES

This section describes the registers and data organization of the TMP68008.

2.1 OPERAND SIZE

Operand sizes are defined as follows: a byte equals eight bits, a word equals 16 bits (two bytes), and a long word equals 32 bits (four bytes). The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Implicit instruction support some subset of all three sizes. When fetching instructions, the TMP68008 always fetches pairs of bytes (words) thus guaranteeing compatibility with the TMP68000.

2.2 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.

2.2.1 Data Registers

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

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

2.2.2 Address Registers

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

2.3 DATA ORGANIZATION MEMORY

The data types supported by the TMP68008 are: bit data, integer data of 8, 16, or 32 bits, and 32-bit addresses. Figure 2.1 shown the organization of these data types in memory.

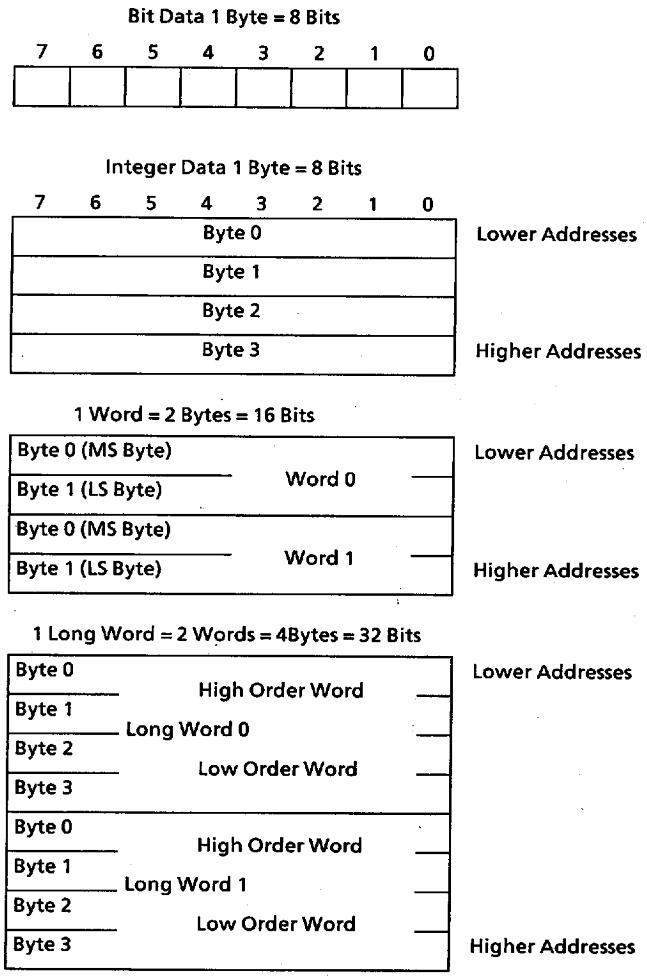


Figure 2.1 Memory Data Organization

2.4 ADDRESSING

Instructions for the TMP68008 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 the instruction.

Effective Address - use of the different effective address modes.

Implicit Refernce - the definition of certain instructions implies the use of specific registers.

2.5 INSTRUCTION FORMAT

Instructions are from one to five words (two to ten bytes) in length as shown in Figure 2.2. Instructions always start on a word boundary thus guaranteeing compatibility with the TMP68000. The length of the instruction and the operation to 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.

Even Byte (A0 = 0) 7 6 5 4 3 2 1 0	Odd Byte (A0 = 1) 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 2.2 Instruction Operation Word General Format

2.6 PROGRAM/DATA REFERENCES

The TMP68008 separates memory references into two classes: program references, and data references. Program references, as the name implies, are references to that sectionof memory containing the program being executed. Data references refer to that section of memory containing 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. The function codes are used to indicate the address space being accessed during a bus cycle.

2.7 REGISTER SPECIFICATION

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

2.8 EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 2.3 shows the general format of the single-effective-address instruction operation word.

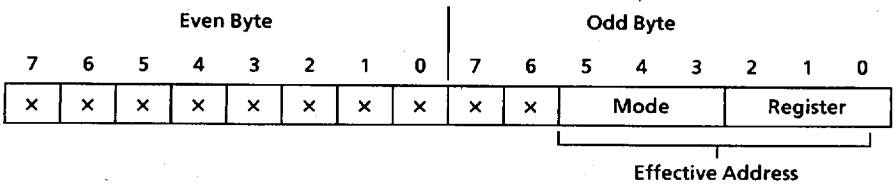


Figure 2.3 Single - Effective - Address Instruction Operation Word

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 registerfield contains the unmber 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 2.2. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

2.8.1 Register Direct Modes

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

2.8.1.1 Data Register Direct

The operand is in the data register specified by the effective address register field.

2.8.1.2 Address Register Direct

The operand is in the address register specified by the effective address register field.

2.8.2 Memory Address Modes

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

2.8.2.1 Address Register Indirect

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

2.8.2.2 Address Register Indirect with Postincrement

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

2.8.2.3 Address Register Indirect with Predecrement

The address of the operand will be in the address register specified by the register field. Before the address register is used for operand access, 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 calssified as a data reference.

2.8.2.4 Address Register Indirect with Displacement

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

2.8.2.5 Address Register Indirect with Index

This address mode requires one word of extension. The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The reference is classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

2.8.3 Special Address Modes

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

2.8.3.1 Absolute Short Address

This address 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.

2.8.3.2 Absolute Long Address

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

2.8.3.3 Program Counter with Displacement

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

2.8.3.4 Program Counter with Index

This address mode requires one word of extension. This 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.

2.8.3.5 Immediate Data

This address mode requires either one or two words of extension depending on the size of the operation.

Byte Operation -	opreand is low order byte of extension word
Word Operation -	opreand is extension word
Long Word Operation -	opreand is in two extension words, high order 16 bits are in the first extension word, low order 16 bits are in the second extension word.

2.8.3.6 Implicit Reference

Some instructions make implicit reference to the program counter (PC), the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), 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	EORI to SR	MOVE to CCR
ANDI to SR	ORI to CCR	MOVE to SR
EORI to CCR	ORI to SR	MOVE from SR

2.9 EFFECTIVE ADDRESS ENCODING SUMMARY

Table 2.1 is a summary of the effective addressing modes discussed in the previous paragraphs.

Table 2.1 Effective Address Encoding Summary

Addressing Mode	Mode	Register
Data Register Direct	000	Register Number
Address Register Direct	001	Register Number
Address Register Indirect	010	Register Number
Address Register Indirect with Postincrement	011	Register Number
Address Register Indirect with Predecrement	100	Register Number
Address Register Indirect with Predecrement	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

2.10 SYSTEM STACK

The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes. Address register seven (A7) is the system stack pointer (SP). The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S bit in the status register. If the S bit indicates supervisor state, SSP is the active system stack pointer and the USP is not used. 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.

3. INSTRUCTION SET SUMMARY

This section contains an overview of the form and structure of the TMP68008 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

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

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

3.1 DATA MOVEMENT OPERATIONS

The basic method of data acquisition (transfer and storage) is provided by the move (MOVE) instruction. The move instruction and the effective addressing modes allow both address and data manipulation. Data 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 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 3.1 is a summary of the data movement operations.

Table 3.1 Data Movement Operations

Instruction	Operand Size	Operation
EXG	32	Xx↔Xy
LEA	32	EA→An
LINK	–	An→-(SP) SP→An SP + displacement→SP
MOVE	8, 16, 32	s→d
MOVEM	16, 32	(EA)→An, Dn An, 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	– =indirect with predecrement
d=destination	+ =indirect with postincrement
[ ]=bit number	# =immediate data

3.2 INTEGER ARITHMETIC OPERATIONS

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

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

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

A test operand (TST) instruction that will set the condition codes as a result of a compare of the operand with zero is also available. Test and set (TAS) is a synchronization instruction useful in multiprocessor systems. Table 3.2 is a summary of the integer arithmetic operations.

Table 3.2 Integer Arithmetic Operations

Instruction	Operand Size	Operation
ADD	8, 16, 32 16, 32	Dn + (EA)→Dn (EA) + Dn→(EA) (EA) + #xxx→(EA) An + (EA)→An
ADDX	8, 16, 32 16, 32	Dx + Dy + X→Dx -(Ax) + -(Ay) + X→(Ax)
CLR	8, 16, 32	0→(EA)

CMP	8, 16, 32
	
	16, 32
		Dn-(EA)
		(EA)-#xxx
		(Ax) + -(Ay) +
		An-(EA)

DIVS	32 ÷ 16	Dn ÷ (EA)→Dn
DIVU	32 ÷ 16	Dn ÷ (EA)→Dn
EXT	8→16 16→32	$(Dn)_8 \rightarrow Dn_{16}$ $(Dn)_{16} \rightarrow Dn_{32}$
MULS	16 x 16→32	Dn x(EA)→Dn
MULU	16 x 16→32	Dn x(EA)→Dn
NEG	8, 16, 32	0-(EA)→(EA)
NEGX	8, 16, 32	0-(EA)-X→(EA)
SUB	8, 16, 32 16, 32	Dn-(EA)→Dn (EA)-Dn→(EA) (EA)-#xxx→(EA) An-(EA)→An
SUBX	8, 16, 32	Dx-Dy-X→Dx -(Ax)--(Ay)-X→(Ax)
TAS	8	(EA)-0, 1→EA[7]
TST	8, 16, 32	(EA)-0

Notes:
[ ] = bit number
#xxx = immediate data
−( ) = indirect with predecrement
( )+ = indirect with postincrement

3.3 LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are avavilable for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 3.3 is a summary of the logical operations.

Table 3.3 Logical Operations

Instruction	Operand Size	Operation
AND	8, 16, 32	$Dn \wedge (EA) \rightarrow Dn$ $(EA) \wedge Dn \rightarrow (EA)$ $(EA) \wedge \#xxx \rightarrow (EA)$
OR	8, 16, 32	$Dn \vee (EA) \rightarrow Dn$ $(EA) \vee Dn \rightarrow (EA)$ $(EA) \vee \#xxx \rightarrow (EA)$
EOR	8, 16, 32	$(EA) \oplus Dn \rightarrow (EA)$ $(EA) \oplus \#xxx \rightarrow (EA)$
NOT	8, 16, 32	$\sim (EA) \rightarrow EA$

Notes:

#xxx = immediate data	$\vee$ = logical OR
~ = invert	$\oplus$ = logical exclusive OR
$\wedge$ = logical AND	

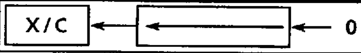
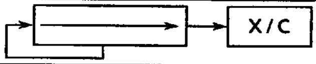
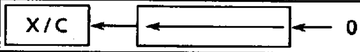
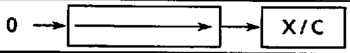
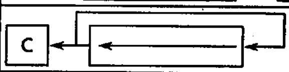
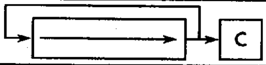
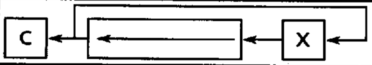
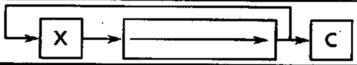
3.4 SHIFT AND ROTATE OPERATIONS

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 and provide single-bit shifts or rotates.

Table 3.4 is a summary of the shift and rotate operations.

Table 3.4 Shift and Rotate Operations

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

3.5 BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change(BCHG). Table 3.5 is a summary of the manipulation operations. (Z is bit 2 of the status register.)

Table 3.5 Bit Manipulation Operations

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

Note :
~ = invert

3.6 BINARY CODED DECIMAL OPERATIONS

Multiprecision arithmetic operations on binary coded decimal numbers are accomplished using the following instructions: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD).Table 3.6 is a summary of the binary coded decimal operations.

Table 3.6 Binary Coded Decimal Operations

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

3.7 PROGRAM CONTROL OPERATIONS

Program control operations are accomplished using a series of conditional and unconditional branch instructions, jump instructions, and return instructions. These instructions are summarized in Table 3.7.

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

CC	- carry clear	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	- alway true
HI	- high	VC	- no overflow
LE	- less or equal	VS	- overflow

Table 3.7 Program Control Operations

Instruction	Operation
Conditional	
Bcc	Branch Conditionally (14 conditions) 8-and 16-Bit Displacement
DBcc	Test Condition, Decrement, and Branch 16Bit Displacement
Scc	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

3.8 SYSTEM CONTROL OPERATIONS

System control oprerations are accomplished by using priviledge instructions, trap generating instructions, and instructions that use or modify the status register. These instructions are summarized in Table 3.8.

Table 3.8 System Control Operations

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

4. SIGNAL AND BUS OPERATION DESCRIPTION

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

4.1 SIGNAL DESCRIPTION

The TMP68008 is available in two package sizes (48-pin and 52-pin). The additional four pins of the 52-pin quad package allow for additional signals: A20, A21, $\overline{\text{BGACK}}$, and $\overline{\text{IPL2}}$.

Throughout this document, references to the address bus pins (A0-A19) and the interrupt priority level pins ($\overline{\text{IPL0}}/\overline{\text{IPL2}}$, $\overline{\text{IPL1}}$) refer to A0-A21 and $\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$, and $\overline{\text{IPL2}}$ for the 52-pin version of the TMP68008.

The pin assignment is show in Figure 4.1(a) for the 48-pin version and in Figure 4.1(b) for the 52-pin version.The input and output signals can be functionally organized into the groups shown in Figure 4.1(a) for the 48-pin version and in Figure 4.1(b) for the 52-pin version. The following paragraphs provide a brief description of the signals and a reference (if applicable) to other paragraphs that contain more information about the function being performed.

(a) 48-Pin Version

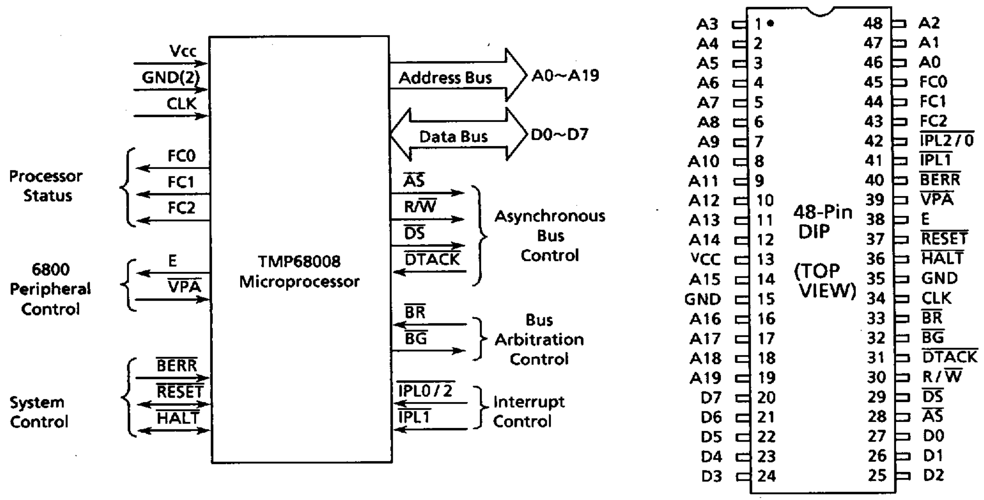
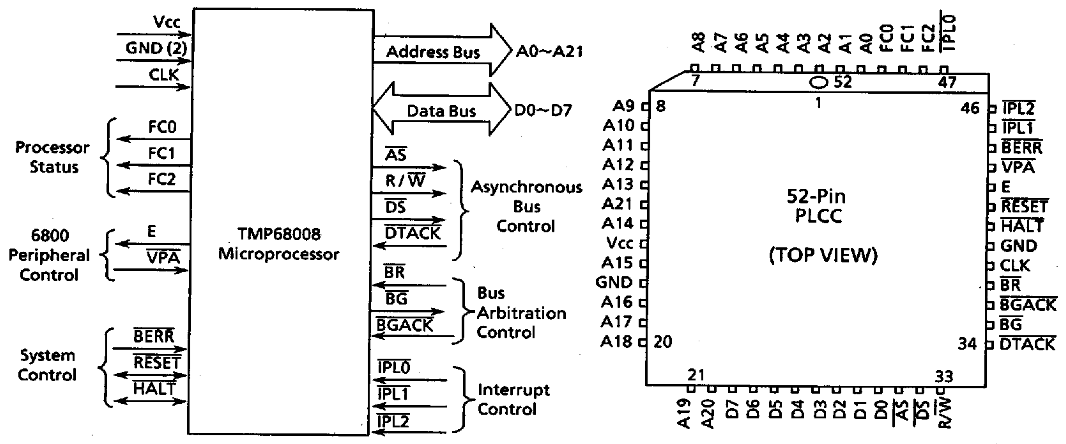


Figure 4.1 Input/Output Signal and Pin Assignments

(b) 52-Pin Version



Note : 52-pin PLCC is under development

Figure 4.1 Input/Output Signals and Pin Assignments

4.1.1 Address Bus (48-Pin: A0~A19 52-Pin: A0~A21)

This unidirectional three-state bus provides the address for bus operation during all cycles except interrupt acknowledge cycles. During interrupt acknowlodge cycles, address lines A1, A2, and A3 provide information about what level interrupt is being serviced while address lines A0and A4 through A19 (A21) are all driven high.

4.1.2 Data Bus (D0~D7)

This 8-bit, bidirectional, three-state bus is the general purpose data path. During an interrupt acknowledge cycle, the external device supplies the vector number on datalines D0~D7.

4.1.3 Asynchronous Bus Control

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

4.1.3.1 Address Strobe ($\overline{AS}$)

This three-state signal indicates that there is a valid address on the address bus.It is also used to "lock" the bus during the read-modify-write cycle used by the test and set (TAS) instruction.

4.1.3.2 Read/Write (R/$\overline{W}$)

This three-state signal defines the data bus transfer as a read or write cycle. The R/$\overline{W}$ signal also works in conjunction with the data strobe as explained in the following paragraph.

4.1.3.3 Data Strobe ($\overline{\text{DS}}$)

This three-state signal controls the flow of data on the data bus as shown in Table 4.1. When the R/$\overline{\text{W}}$ line is high, the processor will read from the data bus as indicated. When the R/$\overline{\text{W}}$ line is low, the processor will write to the data bus as shown.

Table 4.1 Data Strobe Control of Data Bus

$\overline{\text{DS}}$	R/$\overline{\text{W}}$	D0~D7
1	—	No Valid Data
0	1	Valid Data Bits 0~7 (Read Cycle)
0	0	Valid Data Bits 0~7 (Write Cycle)

4.1.3.4 Data Transfer Acknowledge ($\overline{\text{DTACK}}$)

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

4.1.4 Bus Arbitration Control

The 48-pin TMP68008 contains a simple two-wire arbitration circuit and the 52-pin TMP68008 contains the full three-wire TMP68000 bus arbitration control. Both version are designed to work with daisychained networks, priority encoded networks, or a combination of these techniques. This circuit is used in determining which device will be the bus master device.

4.1.4.1 Bus Request ($\overline{\text{BR}}$)

This input is wire ORed with all other devices that could be bus masters. This device indicates to the processor that some other device desires to become the bus master. Bus requests may be issued at any time in a cycle or even if no cycle is being performed.

4.1.4.2 Bus Grant ($\overline{\text{BG}}$)

This output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.

4.1.4.3 Bus Grant Acknowledge ($\overline{\text{BGACK}}$)

This input, available on the 52-pin version only, 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.

Notes :

1) There is a two-clock interval stradding the transition of $\overline{\text{AS}}$ from the inactive state to the active state during which $\overline{\text{BG}}$ cannot be issued.

2) If an existing TMP68000 system is retrofitted to use the TMP68008, 48-pin version (using $\overline{\text{BR}}$ and $\overline{\text{BG}}$ only), the existing $\overline{\text{BR}}$ and $\overline{\text{BGACK}}$ signals should be ANDed and the resultant signal connected to the TMP68008's $\overline{\text{BR}}$.

4.1.5 Interrupt Control (48-Pin: $\overline{\text{IPL0}}/\overline{\text{IPL2}}$, $\overline{\text{IPL1}}$
52-Pin: $\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$, $\overline{\text{IPL2}}$)

These input pins indicate the encoded priority level of the device requesting an interrupt. The TMP68000 and the52-pin TMP68008 MPUs use three pins to encode a range of 0~7but, the 48-pin TMP68008 only two pins are available. By connecting the $\overline{\text{IPL0}}/\overline{\text{IPL2}}$ pin to both the $\overline{\text{IPL0}}$ and $\overline{\text{IPL2}}$ inputs internally, the 48-pin encodes values of 0, 2, 5, and 7.Level zero is used to indicate that there are no interrupts pending and level seven is a non-maskable edge-triggered interrupt. Except for level seven, the requseting level must be greater than the interrupt mask level contained in the processor status register before the processor will acknowledge the request.

The level presented to these inputs is continually monitored to allow for the case of a requesting level that is less than or equal to the processor status register level to be followed by a request that is greater than the processor status register level. A satisfactory interrupt condition must exist for two successive clocks before triggering an internal interrupt request. An interrupt acknowledge sequence is indicated by the function codes.

4.1.6 System Control

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

4.1.6.1 Bus Error ($\overline{\text{BERR}}$)

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 until, or

4. various other application dependent errors.

The bus error signal interacts with the halt signal to determine if the current bus cycle sould be reexecuted or if exception processing sould be perfromed. Refer to "4.2.3 Bus Error and Halt Operation" for a detailed description of the interaction which is summarized below.

$\overline{\text{BERR}}$	$\overline{\text{HALT}}$	Resulting Operation
high	high	Normal operation
high	low	Single bus cycle operation
low	high	Bus error - exception processing
low	low	Bus error - re-run current cycle

4.1.6.2 Reset ($\overline{\text{RESET}}$)

This bidirectional signal line acts to reset (start a system initialization sequence) the processor in response to an external $\overline{\text{RESET}}$ signal. An internally generated reset (result of a reset instruction) causes all external devices to be reset and the internal state of the processor is not affected. A total system reset (prpcessor and external devices) is the result of external $\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ signals applied at the same time. Refer to "4.2.1 Reset Operation" for further information.

4.1.6.3 Halt ($\overline{\text{HALT}}$)

When this bidirectional line is driven by an external device, it will cause the processor to stop at the completion of the current bus cycle. When the processor has been halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state. Refer to "4.2.3 Bus Errorand Halt Operation" for additional information about the interaction between the halt and bus error signals.

When the processor has stopped executing instructions, such as in a double bus fault condition, the halt line is driven by the processor to indicate to external devices that the processor has stopped.

4.1.7 6800 Peripheral Control

These control signals are used to allow the interfacing of synchronous 6800 peripheral devices with the asynchronous TMP68008. These signals are explained in the following paragraphs.

The TMP68008 does not supply a valid memory address ($\overline{\text{VMA}}$)signal like that of the TMP68000. The $\overline{\text{VMA}}$ signal indicates to the 6800 peripheral devices that there is a valid address on the address bus and that the processor is synchronized to the enable clock. This signal can be produced by a TTL circuit (see a sample circuit in Figure 4.2). The $\overline{\text{VMA}}$ signal, in this circuit, only responds to a valid peripheral address (VPA) input which indicates that the peripheral is an 6800 Family device. Timing for this circuit is shown in Figure 6.2.

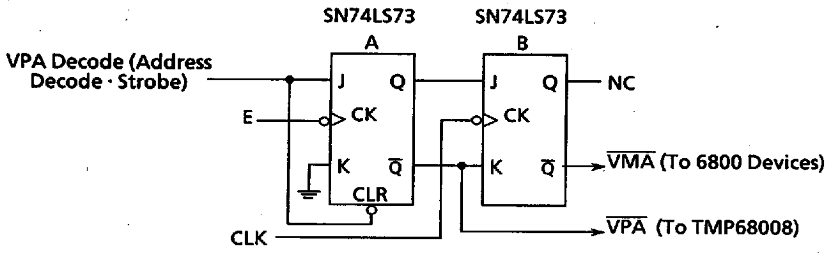

SN74LS73
A
SN74LS73
B
VPA Decode (Address Decode · Strobe)
J
Q
CK
K
$\overline{Q}$
CLR
E
CLK
NC
$\overline{VMA}$ (To 6800 Devices)
$\overline{VPA}$ (To TMP68008)

Figure 4.2 External $\overline{VMA}$ Generation

The VPA decode shown in Figure 4-2 is an active high decode indicating that address strobe ($\overline{AS}$) has been asserted and the address bus is addressing an 6800 peripheral.The $\overline{VPA}$ output of the circuit is used to indicate to the TMP68008 that the data transfer should be synchronized with the enable (E) signal.

4.1.7.1 Enable (E)

This signal is the standard enable signal common to all 6800 type peripheral devices. The period for this output is 10 TMP68008 clock periods (six clocks low, four clocks high).

4.1.7.2 Valid Peripheral Address ($\overline{VPA}$)

This input indicates that the device or region addressed is a 6800 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 interrupt. Refer to "6.0 INTERFACE WITH 6800 PERIPHERALS. "

4.1.8 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 4.2. The information indicated by the function code outputs is valid whenever address strobe ($\overline{AS}$) is active.

Table 4.2 Function Code Outputs

Function Code Output			Cycle Type
FC2	FC1	FC0	
L	L	L	(Undefined, Reserved)
L	L	H	User Data
L	H	L	User Program
L	H	H	(Undefined, Reserved)
H	L	L	(Undefined, Reserved)
H	L	H	Supervisor Data
H	H	L	Supervisor Program
H	H	H	Interrupt Acknowledge

4.1.9 Clock (CLK)

The clock input is a TTL-compatible signal that is internally buffered for development of the internal clocks needed by the processor. The clock input shall be a constant frequency.

4.1.10 Vcc and GND

Power is supplied to the processor using these two signals. Vcc is power and GND is the ground connection.

4.1.11 Signal Summary

Table 4.3 Signal Summary

Signal Name	Mnemonic		Input/Output	Active State	3-State	Hi-Z	
	48Pin	52Pin				on $\overline{HALT}$	on $\overline{BGACK}$
Address Bus	A0~A19	A0~A21	Output	High	Yes	Yes	Yes
Data Bus	D0~D7	D0~D7	Input/Output	High	Yes	Yes	Yes
Address Strobe	$\overline{AS}$	$\overline{AS}$	Output	Low	Yes	No	Yes
Read/Write	R / $\overline{W}$	R / $\overline{W}$	Output	High-Read Low-Write	Yes	No	Yes
Data Strobes	$\overline{DS}$	$\overline{DS}$	Output	Low	Yes	No	Yes
Data Transfer Acknowledge	$\overline{DTACK}$	$\overline{DTACK}$	Input/Output	Low	No	No	No
Bus Request	$\overline{BR}$	$\overline{BR}$	Input/Output	Low	No	No	No
Bus Grant	$\overline{BG}$	$\overline{BG}$	Output	Low	No	No	No
Bus Grant Acknowledge*	—	$\overline{BGACK}$	Input	Low	No	No	No
Interrupt Priority Level	$\overline{IPL0/2}$, $\overline{IPL1}$	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	Input	Low	No	No	No
Bus Error	$\overline{BERR}$	$\overline{BERR}$	Input	Low	No	No	No
Reset	$\overline{RESET}$	$\overline{RESET}$	Input	Low	Yes	No^1	No^1
Halt	$\overline{HALT}$	$\overline{HALT}$	Input/Output	Low	Yes	No^1	No^1
Enable	E	E	Output	High	No	No	No
Valid Peripheral Address	$\overline{VPA}$	$\overline{VPA}$	Input	Low	No	No	No
Function Code Output	FC0, FC1, FC2	FC0, FC1, FC2	Output	High	Yes	No	Yes
Clock	CLK	CLK	Input	High	No	No	No
Power Input	Vcc	Vcc	Input	—	—	—	—
Ground	GND	GND	Input	—	—	—	—

Note :
1. Open drain

4.2 BUS OPERATION

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

4.2.1 Data Transfer Operations

Transfer of data between devices involves the followingleads:

- Address bus A0~A19
- Data bus D0~D7
- Control signals

The address and data buses are separate non-multiplexed parallel buses. Data tarnsfer is accomplished with an asynchronous bus structure that uses handshakes to ensure the correct movement of data. 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 TMP68008 for interlocked multiprocessor communications.

Note : The terms **assertion** and **negation** will be used extensively. This is done to avoid confusion when dealing 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 voltage is low or high. The term negate or negation is used to indicate that a signal is inactive or false.

4.2.1.1 Read Cycle

During a read cycle, the processor receives data from the memory or a peripheral device. The processor reads bytes of data in all cases. If the instruction specifies a word (or double word) operation, the processor reads both bytes. When the instruction specifies byte operation, the processor uses A0 to determine which byte to read and then issues data strobe.

Aword read cycle flowchart is given in Figure 4.3. A byte read cycle flowchart is given in Figure 4.4. Read cycle timing is given in Figure 4.5. Figure 4.6 details words and byte read cycle operations.

4.2.1.2 Write Cycle

During a write cycle, the processor sends data to either the memory or a pripheral 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 A0 to determine which byte to write and then issues the data strobe. A word write cycle flowchart is given in Figure 4.7. A byte write cycle flowchart is given in Figure 4.8. Write cycle timing is given in Figure 4.5. Figure 4.9 details word and byte write cycle operation.

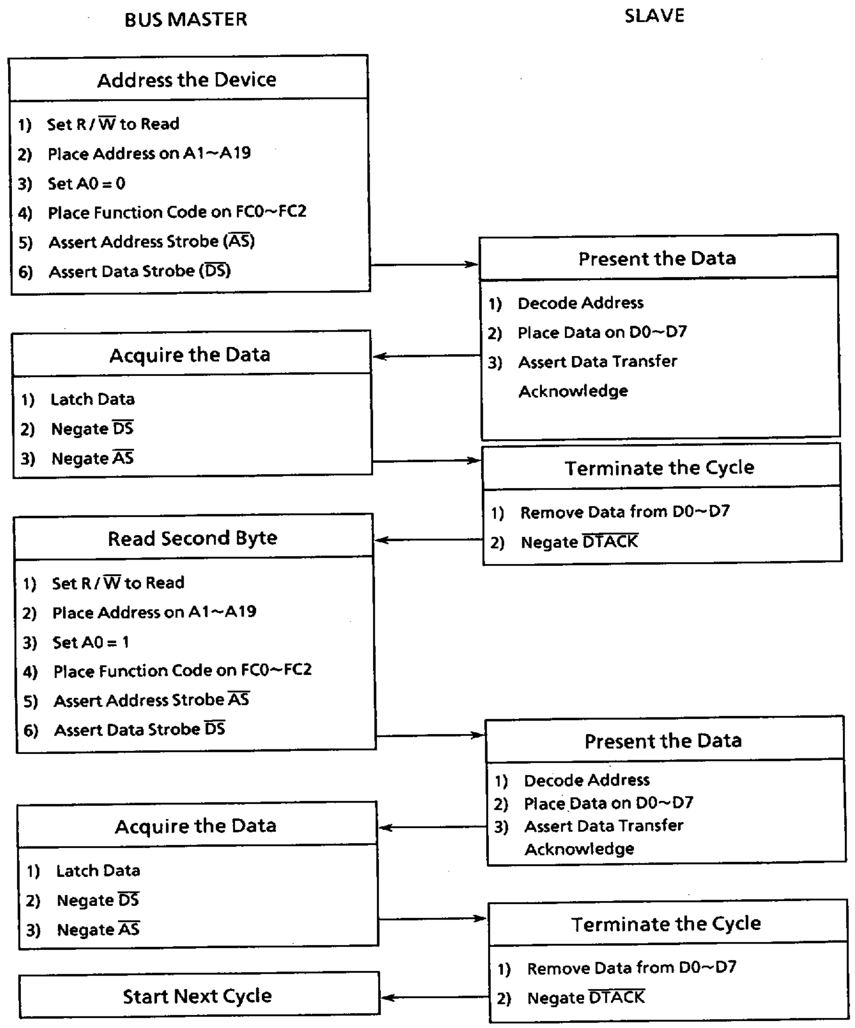

BUS MASTER
SLAVE
Address the Device
1) Set $R/\overline{W}$ to Read
2) Place Address on A1~A19
3) Set A0 = 0
4) Place Function Code on FC0~FC2
5) Assert Address Strobe ($\overline{AS}$)
6) Assert Data Strobe ($\overline{DS}$)
Present the Data
1) Decode Address
2) Place Data on D0~D7
3) Assert Data Transfer Acknowledge
Acquire the Data
1) Latch Data
2) Negate $\overline{DS}$
3) Negate $\overline{AS}$
Terminate the Cycle
1) Remove Data from D0~D7
2) Negate $\overline{DTACK}$
Read Second Byte
1) Set $R/\overline{W}$ to Read
2) Place Address on A1~A19
3) Set A0 = 1
4) Place Function Code on FC0~FC2
5) Assert Address Strobe $\overline{AS}$
6) Assert Data Strobe $\overline{DS}$
Present the Data
1) Decode Address
2) Place Data on D0~D7
3) Assert Data Transfer Acknowledge
Acquire the Data
1) Latch Data
2) Negate $\overline{DS}$
3) Negate $\overline{AS}$
Terminate the Cycle
1) Remove Data from D0~D7
2) Negate $\overline{DTACK}$
Start Next Cycle

Figure 4.3 Word Read Cycle Flowchart

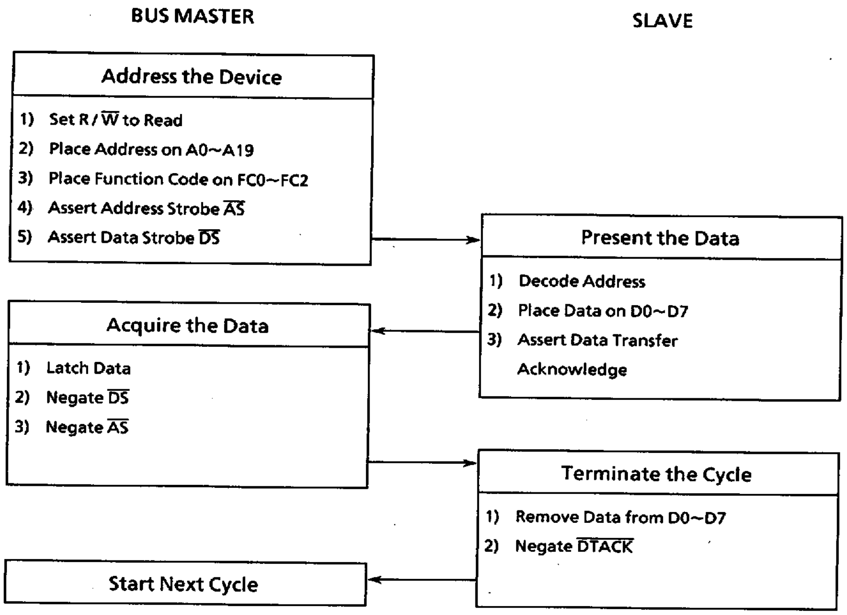

BUS MASTER
SLAVE
Address the Device
1) Set R / $\overline{W}$ to Read
2) Place Address on A0~A19
3) Place Function Code on FC0~FC2
4) Assert Address Strobe $\overline{AS}$
5) Assert Data Strobe $\overline{DS}$
Present the Data
1) Decode Address
2) Place Data on D0~D7
3) Assert Data Transfer Acknowledge
Acquire the Data
1) Latch Data
2) Negate $\overline{DS}$
3) Negate $\overline{AS}$
Terminate the Cycle
1) Remove Data from D0~D7
2) Negate $\overline{DTACK}$
Start Next Cycle

Figure 4.4 Byte Read Cycle Flowchart

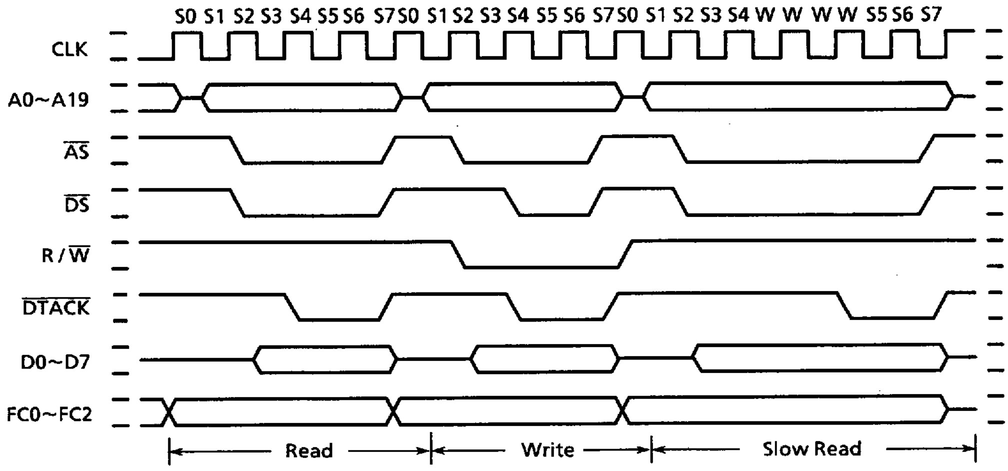

S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 W W W W S5 S6 S7
CLK
A0~A19
AS
DS
R/W
DTACK
D0~D7
FC0~FC2
Read
Write
Slow Read

Figure 4.5 Read and Write Cycle Timing Diagram

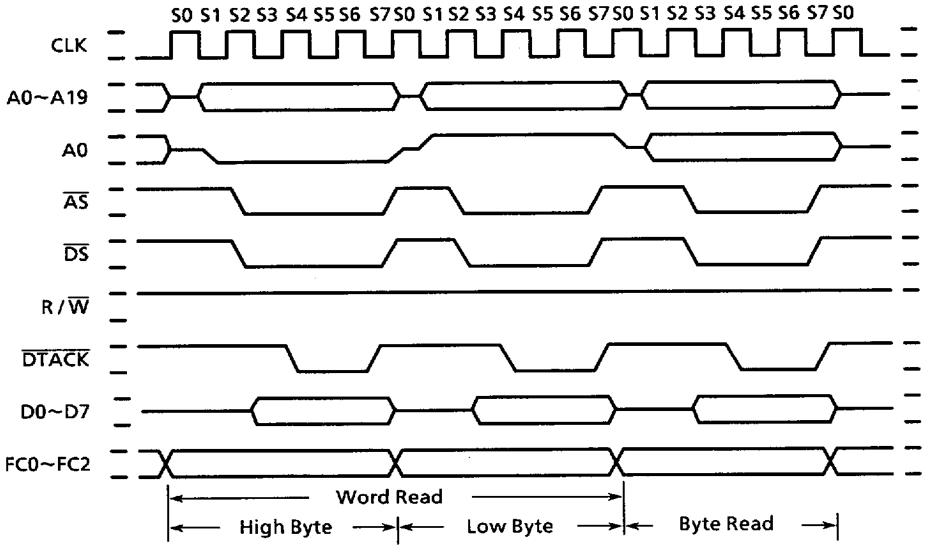

S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 S5 S6 S7 S0
CLK
A0~A19
A0
AS
DS
R/W
DTACK
D0~D7
FC0~FC2
Word Read
High Byte
Low Byte
Byte Read

Figure 4.6 Word and Byte Read Cycle Timing

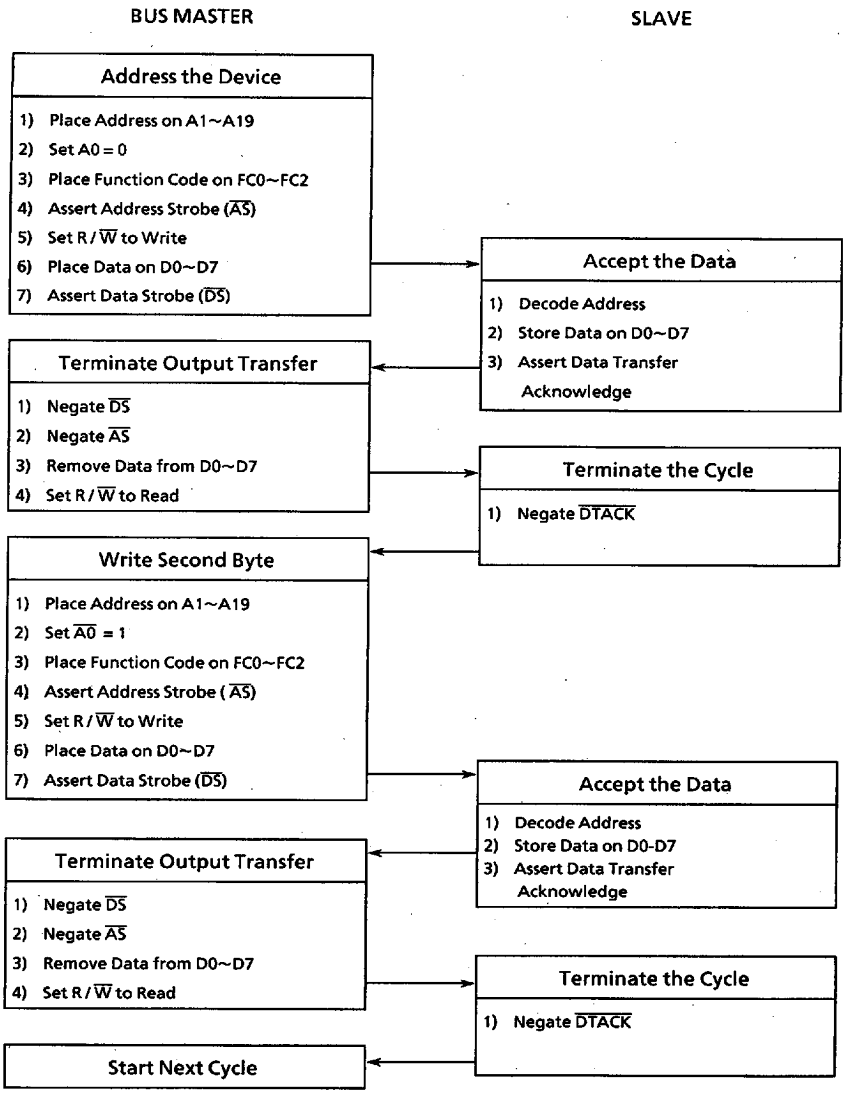

BUS MASTER
SLAVE
Address the Device
1) Place Address on A1~A19
2) Set A0 = 0
3) Place Function Code on FC0~FC2
4) Assert Address Strobe ($\overline{AS}$)
5) Set R/$\overline{W}$ to Write
6) Place Data on D0~D7
7) Assert Data Strobe ($\overline{DS}$)
Accept the Data
1) Decode Address
2) Store Data on D0~D7
3) Assert Data Transfer Acknowledge
Terminate Output Transfer
1) Negate $\overline{DS}$
2) Negate $\overline{AS}$
3) Remove Data from D0~D7
4) Set R/$\overline{W}$ to Read
Terminate the Cycle
1) Negate $\overline{DTACK}$
Write Second Byte
1) Place Address on A1~A19
2) Set $\overline{A0}$ = 1
3) Place Function Code on FC0~FC2
4) Assert Address Strobe ($\overline{AS}$)
5) Set R/$\overline{W}$ to Write
6) Place Data on D0~D7
7) Assert Data Strobe ($\overline{DS}$)
Accept the Data
1) Decode Address
2) Store Data on D0-D7
3) Assert Data Transfer Acknowledge
Terminate Output Transfer
1) Negate $\overline{DS}$
2) Negate $\overline{AS}$
3) Remove Data from D0~D7
4) Set R/$\overline{W}$ to Read
Terminate the Cycle
1) Negate $\overline{DTACK}$
Start Next Cycle

Figure 4.7 Word Write Cycle Flowchart

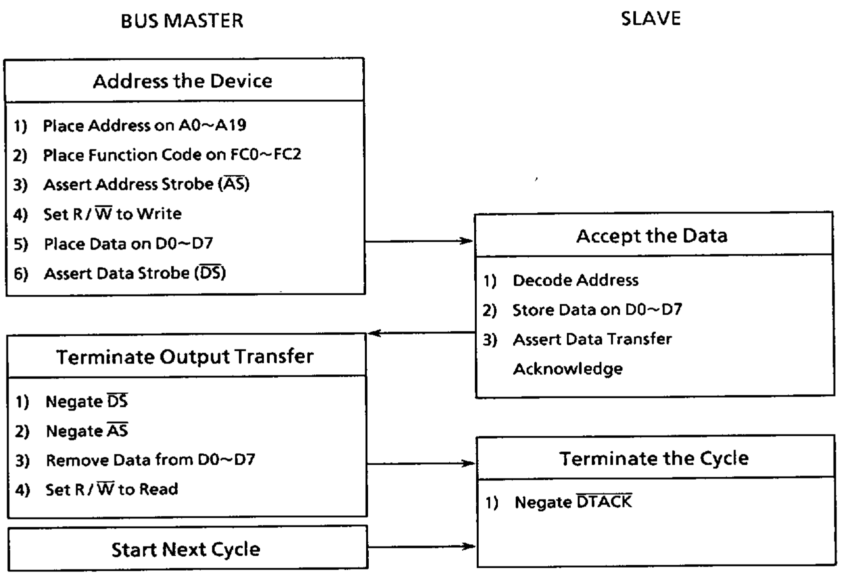

BUS MASTER
SLAVE
Address the Device
1) Place Address on A0~A19
2) Place Function Code on FC0~FC2
3) Assert Address Strobe ($\overline{AS}$)
4) Set R/$\overline{W}$ to Write
5) Place Data on D0~D7
6) Assert Data Strobe ($\overline{DS}$)
Accept the Data
1) Decode Address
2) Store Data on D0~D7
3) Assert Data Transfer Acknowledge
Terminate Output Transfer
1) Negate $\overline{DS}$
2) Negate $\overline{AS}$
3) Remove Data from D0~D7
4) Set R/$\overline{W}$ to Read
Terminate the Cycle
1) Negate $\overline{DTACK}$
Start Next Cycle

Figure 4.8 Byte Write Cycle Flowchart

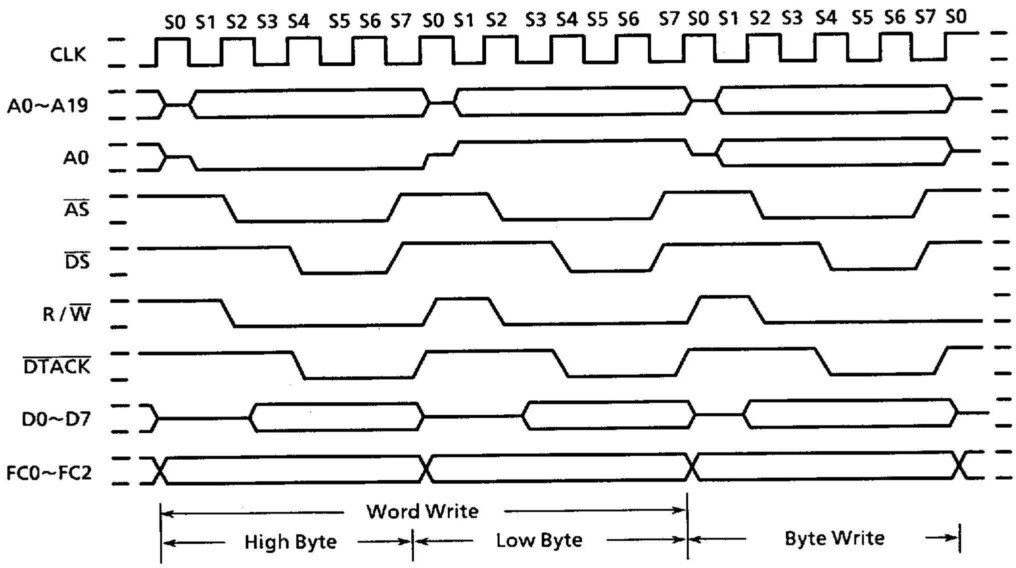


Figure 4.9 Word and Byte Write Cycle Timing

4.2.1.3 Read-Modify-Write Cycle

The read-modify-write cycle performs a byte read, modifies the data in the arithmetic logic unit, and writes the data back to the same address. in the TMP68008, 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 processorsin in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycle and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write cycle flowchart is given in Figure 4.10 and a timing diagram is given in Figure 4.11.

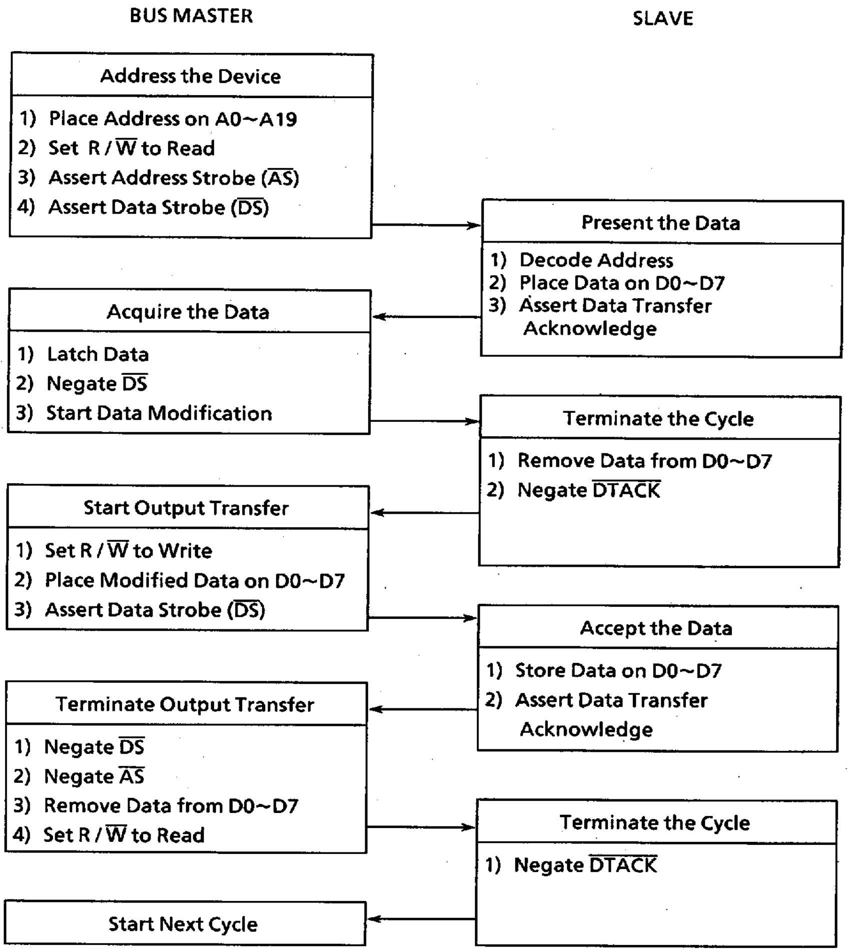

BUS MASTER
SLAVE
Address the Device
1) Place Address on A0~A19
2) Set R/$\overline{W}$ to Read
3) Assert Address Strobe ($\overline{AS}$)
4) Assert Data Strobe ($\overline{DS}$)
Present the Data
1) Decode Address
2) Place Data on D0~D7
3) Assert Data Transfer Acknowledge
Acquire the Data
1) Latch Data
2) Negate $\overline{DS}$
3) Start Data Modification
Terminate the Cycle
1) Remove Data from D0~D7
2) Negate $\overline{DTACK}$
Start Output Transfer
1) Set R/$\overline{W}$ to Write
2) Place Modified Data on D0~D7
3) Assert Data Strobe ($\overline{DS}$)
Accept the Data
1) Store Data on D0~D7
2) Assert Data Transfer Acknowledge
Terminate Output Transfer
1) Negate $\overline{DS}$
2) Negate $\overline{AS}$
3) Remove Data from D0~D7
4) Set R/$\overline{W}$ to Read
Terminate the Cycle
1) Negate $\overline{DTACK}$
Start Next Cycle

Figure 4.10 Read - Modify - Write Cycle Flowchart

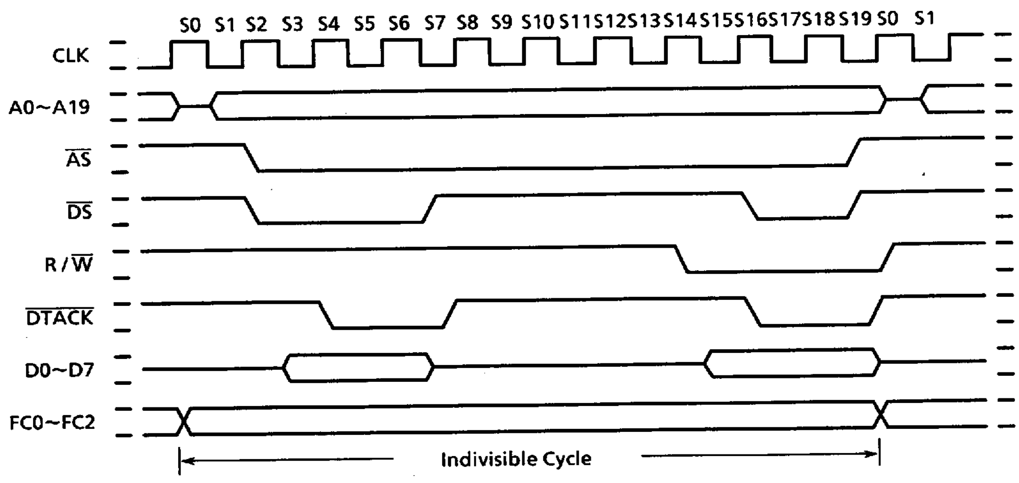

S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S0 S1
CLK
A0~A19
AS
DS
R/W
DTACK
D0~D7
FC0~FC2
Indivisible Cycle

Figure 4.11 Read -Modify-Write Cycle Timing

4.2.2 Bus Arbitration

Bus arbitration on the 52-pin version of the TMP68008 is identical to that on the TMP68000.

Bus arbitration on the 48-pin version of the TMP68008 has been modified from that on the TMP68000. It is controlled by the same finite state machine as on the TMP68000, but because the $\overline{\text{BGACK}}$ input signal is not bonded out to a pin and is, instead, permanently negated internally, the bus arbitration becomes a two-wire handshake circuit. Therefore, in reading the following paragraphs for a description of bus arbitration on the 48-pin version of the TMP68008, the $\overline{\text{BGACK}}$ signal should be considered permanently negated.

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. on the 52-pin version of the TMP68008 only, acknowledging that mastership has been assumed.

Flowcharts showing the detail involved in a request from a single device are illustrated in Figure 4.12 for the 48-pin version and Figure 4.13 for the 52-pin version. Timing diagrams for the sama operation are given in Figure 4.14 and Figure 4.15. This technique allows processing of bus requests during data transfer cycles.

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

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.

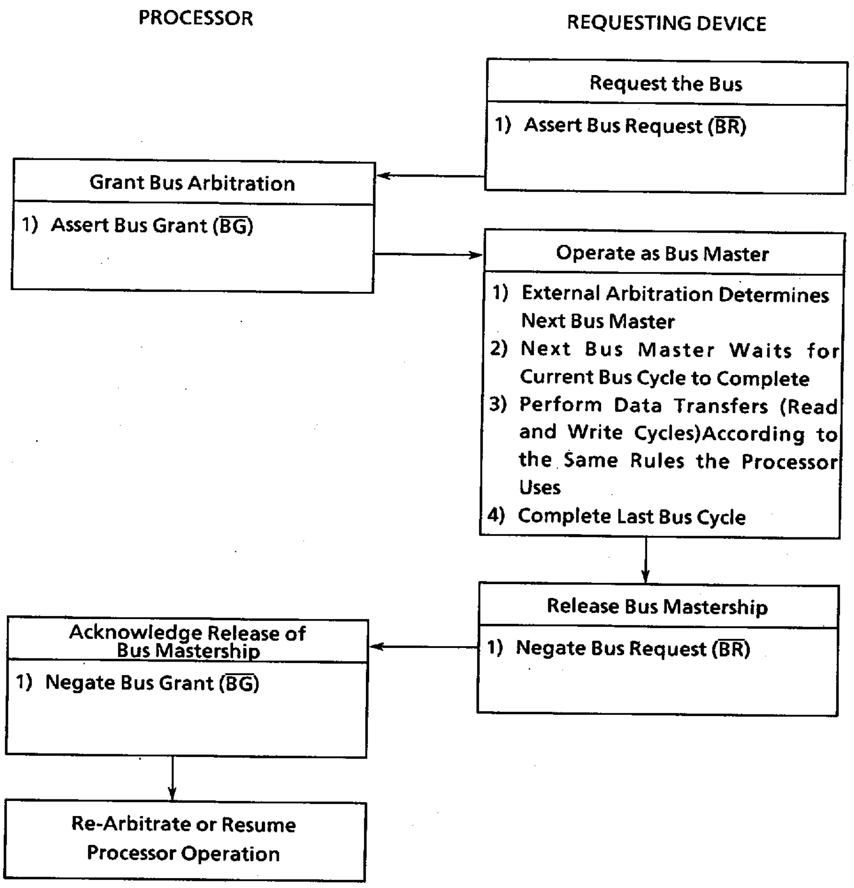

PROCESSOR
REQUESTING DEVICE
Request the Bus
1) Assert Bus Request (BR)
Grant Bus Arbitration
1) Assert Bus Grant (BG)
Operate as Bus Master
1) External Arbitration Determines Next Bus Master
2) Next Bus Master Waits for Current Bus Cycle to Complete
3) Perform Data Transfers (Read and Write Cycles)According to the Same Rules the Processor Uses
4) Complete Last Bus Cycle
Release Bus Mastership
1) Negate Bus Request (BR)
Acknowledge Release of Bus Mastership
1) Negate Bus Grant (BG)
Re-Arbitrate or Resume Processor Operation

Figure 4.12 Bus Arbitration Cycle Flowchart for the 48-Pin Version

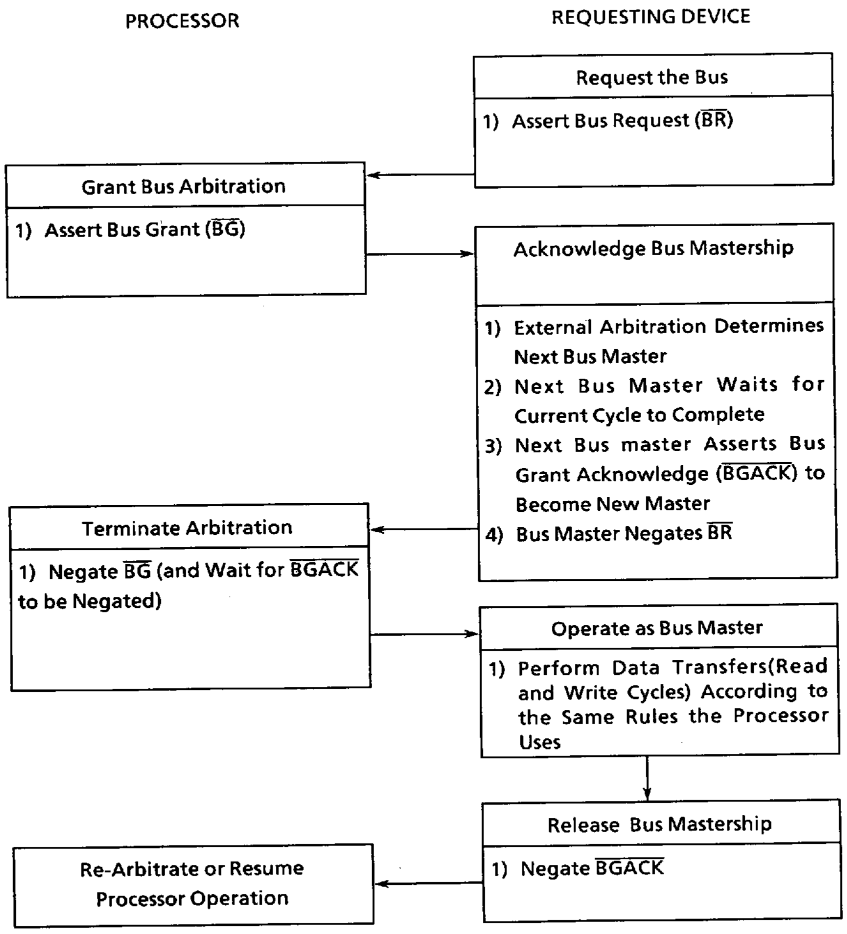

PROCESSOR
REQUESTING DEVICE
Request the Bus
1) Assert Bus Request ($\overline{BR}$)
Grant Bus Arbitration
1) Assert Bus Grant ($\overline{BG}$)
Acknowledge Bus Mastership
1) External Arbitration Determines Next Bus Master
2) Next Bus Master Waits for Current Cycle to Complete
3) Next Bus master Asserts 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 4.13 Bus Arbitration Cycle Floowchart for the 52-Pin Version

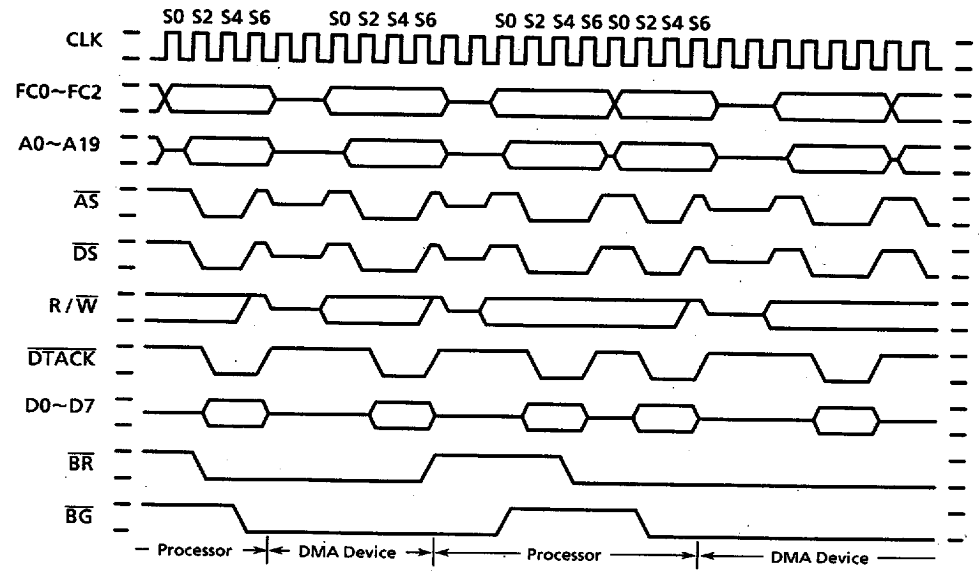


Figure 4.14 Bus Arbitration Timing for the 48-Pin Version

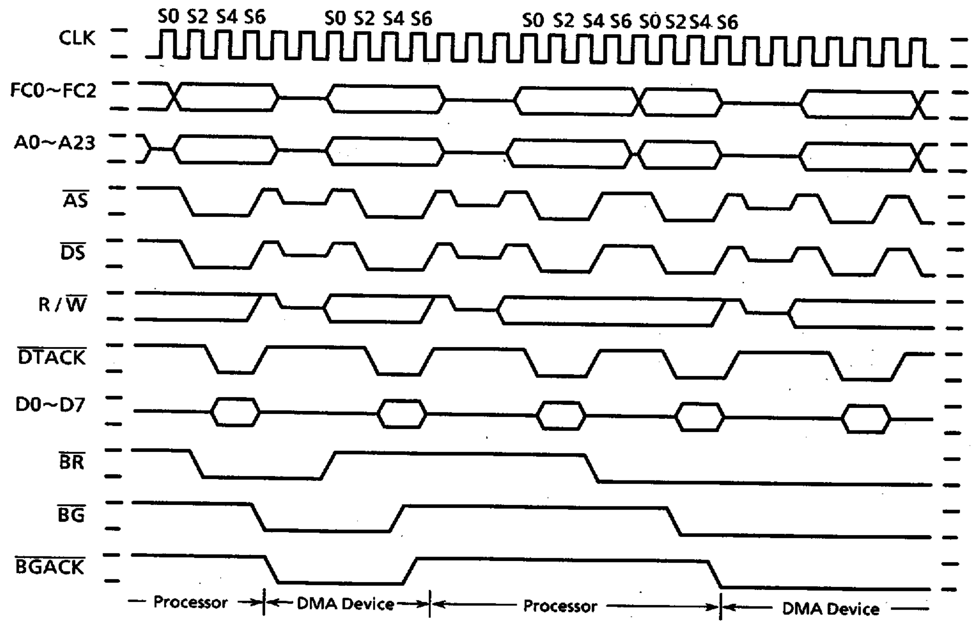


Figure 4.15 Bus Arbitration Timing for the 52-Pin Version

4.2.2.1 Requesting The Bus

External devices capable of becoming bus masters request the bus by asserting the bus request ($\overline{\text{BR}}$) signal. This is a wire-0Red 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.

On the 52-pin version, when no acknowledge is received before the bus request signal goes inactive, the processor will continue processing when it detects that the bus request is inactive. This allows ordinary processing to continue if the arbitration circuitry responded to noise inadvertently.

4.2.2.2 Reciving The Bus Grant

The processor asserts bus grant ($\overline{\text{BG}}$) 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 ($\overline{\text{AS}}$) signal. In this case, bus grant will be delayed until $\overline{\text{AS}}$ is asserted to indicate to external devices that a bus cycle is being executed.

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

4.2.2.3 Acknowledgement of Mastership (52-Pin Version of TMP68008 Only)

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 $\overline{\text{BGACK}}$. 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.

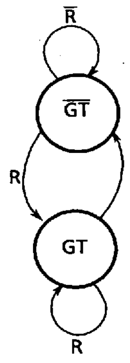
The bus requset from the granted device should be dropped after bus grant acknowledge is asserted. If a bus request is still pending, another bus grant will be asserted within a few clocks of the negation of the bus grant. Refer to "4.2.3 Bus Arbitration Control Unit. " Note that processor does perform any external bus cycles before it re-asserts bus grant.

4.2.3 Bus Arbitration Control

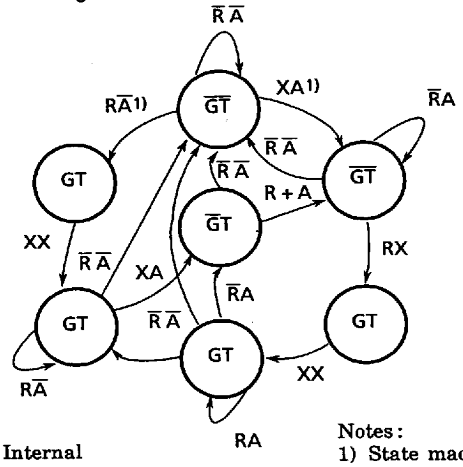
The bus arbitration control unit in the TMP68008 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 4.16 for both pin versions of the TMP68008. All asynchronous signals to the TMP68008 are synchronized before being used internally. This synchronization is accomplished in a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (#47) has been met (see Figure 4.17). The input signal is sampled on the falling edge of the clock and is valid internally after the next falling edge.

As shown in Figure 4.16, input signals labeled R and A are internally synchronoized on the bus request and bus grant acknowledge pins respectively. The bus grant output is labeled G and the internal three-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) State Diagram for the 48-Pin Version of TMP68008



(b) State Diagrm for 52-Pin Version of TMP68008



$\overline{R}\,\overline{A}$
$R\overline{A}$1)
XA1)
$\overline{R}$A
$\overline{GT}$
GT
$\overline{R}\,\overline{A}$
$\overline{R}\,\overline{A}$
R + A
$\overline{GT}$
$\overline{GT}$
XX
RX
$\overline{R}\,\overline{A}$
XA
$\overline{R}$A
GT
$\overline{R}\,\overline{A}$
GT
GT
XX
$R\overline{A}$
RA

R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = 3-State Control to Bus Control Logic2)
X = Don't Care

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

Figure 4.16 TMP68008 Bus Arbitration Unit State Diagram

Atiming diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 4.18. The bus arbitration sequence while the bus is inactive (i, e., executing internal operations such as a multiply instruction) is shown in Figure 4.19.

If a bus request is made at a time when the MPU has already begun as 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 unit the second rising edge following its internal assertion. This sequence is shown in Figure 4.20.

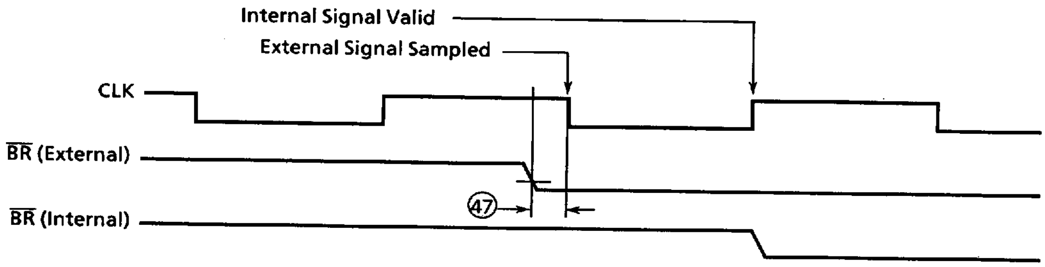


Figure 4.17 Timing Relationship of External Asynchronqus Inputs to Internal Signals

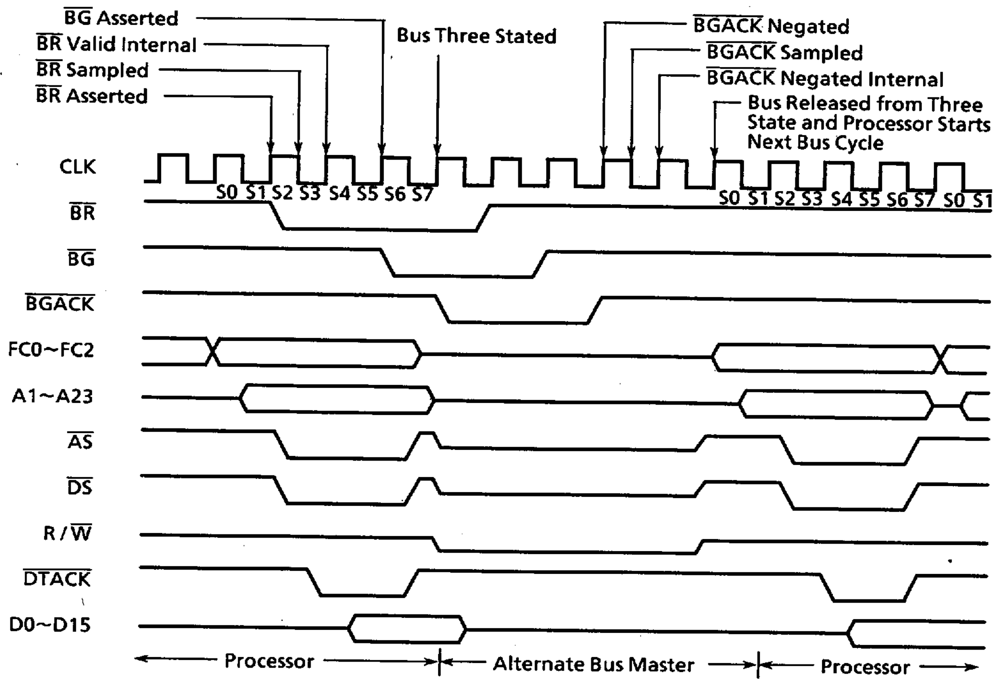


Figure 4.18 Bus Arbitration Timing Diagram-Processor Active

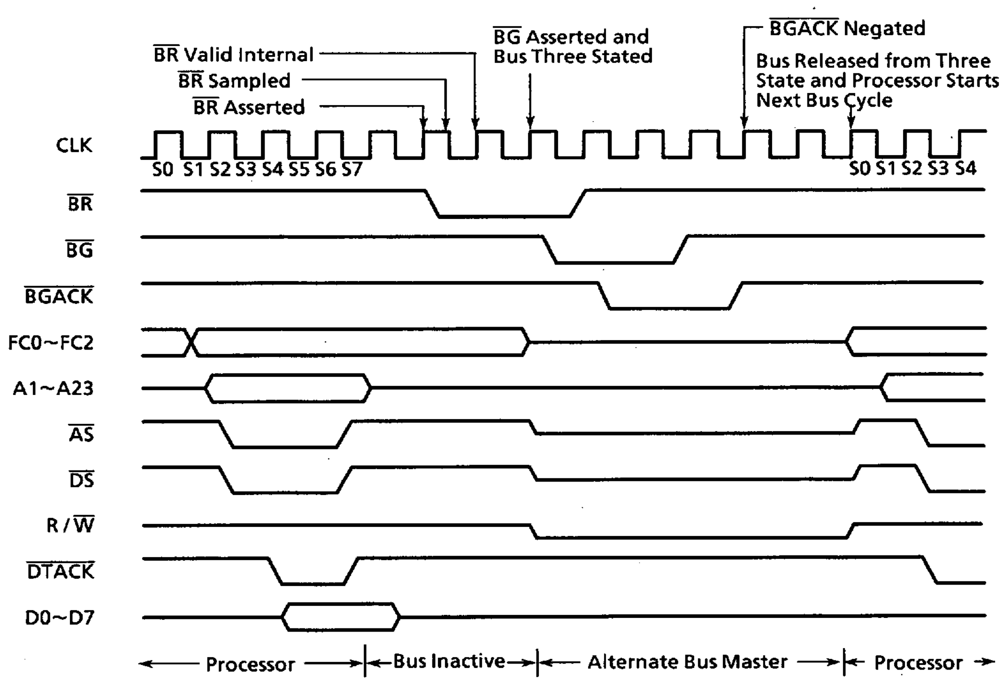


Figure 4.19 Bus Arbitration Timing Diagram-Bus Inactive

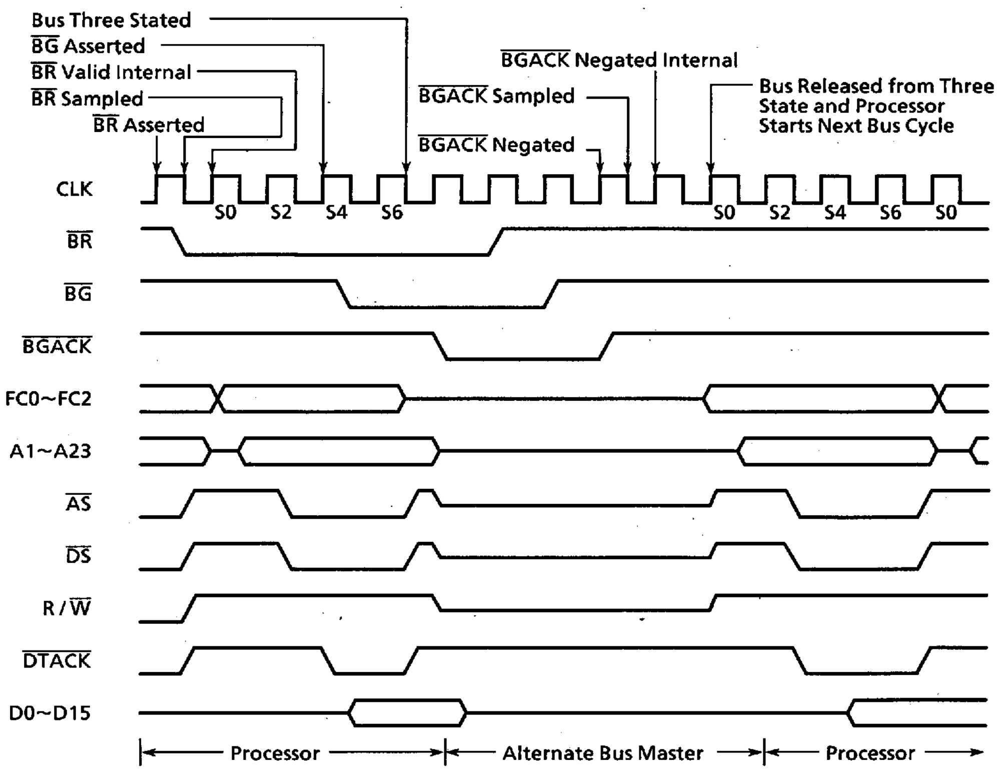


Figure 4.20 Bus Arbitration Timing Diagram-Special Cace

4.2.4 Bus Error and Halt Operation

In a bus architecture that requires a handshake from an external device, the possibility exists that the handshake might not occur. Since different systems will requirea 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 thebus cycle again.

4.2.4.1 Exception Sequence

When the bus error signal is asserted, the current bus cycle is terminated. $\overline{\text{AS}}$ will be negated 2.5 clock periods after $\overline{\text{BERR}}$ is recognized. See 4.4 ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION for more information. As long as $\overline{\text{BERR}}$ remains asserted, the data and address buses will be in the high-impedance state. When $\overline{\text{BERR}}$ is negated, the processor will begin stacking for exception processing. 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.
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 processor loads the new program counter from the bus error vector. A software bus error handler routine is then executed by the processor. Refer to "5.2 EXCEPTION PROCESSING" for additionalinformation.

4.2.4.2 Re-Running the Bus Cycle

When the processor receives a bus error signal during a bus cycle and the $\overline{\text{HALT}}$ pin is being driven by an external device, the processor enters the re-run sequence. Figure 4.21 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 $\overline{\text{AS}}$. If $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ are asserted during a read-modify-write bus cycle, a bus error operation results.

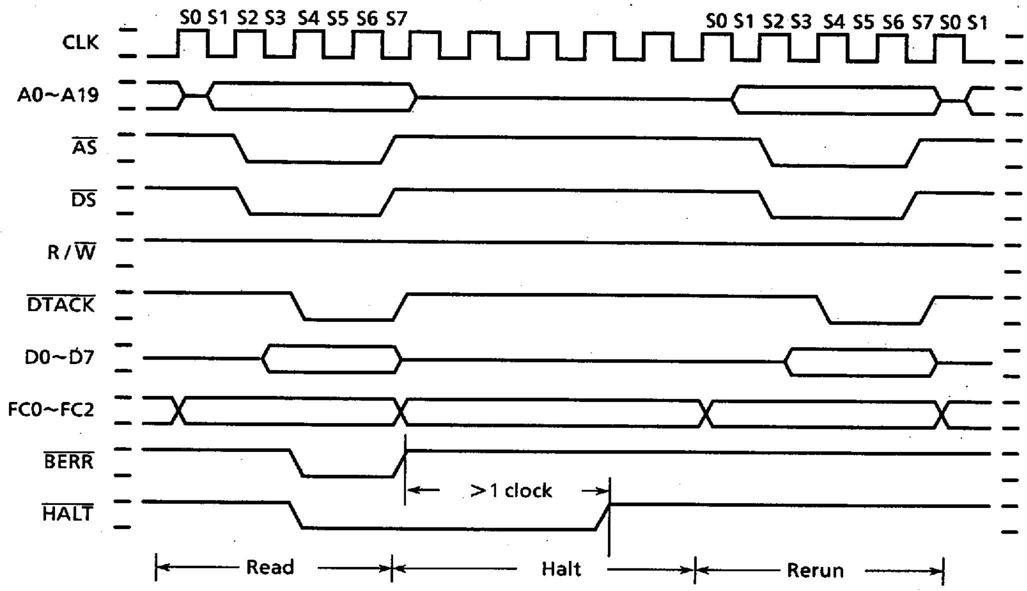


Figure 4.21 Re-Run Bus Cycle Timing Information

4.2.4.3 $\overline{\text{HALT}}$ Operation With No Bus Error

The halt input signal to the TMP68008 performs a halt/run/signal-step function in a similar fashion to the 6800 halt function.The halt and run modes are somewhat self explanatory in that when the halt signal is constantly active the processor "halts" (does noting) and when the halt signal is constantly inactive the processor "runs" (does something).$\overline{\text{HALT}}$ operation timing is shown in Figure 4.22.

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

Figure 4.23 details the timing required for correct single-step operations. Some care must be exercised to avoid harmful interactions between the bus error signal and the $\overline{\text{HALT}}$ 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(see 4.2.4 Reset Operation).

When the processor completes a bus cycle after recognizing that the halt signal is active,the address and data bus signals are put in the high-impedance state.

While the processor is honoring the halt request,bus arbitration perfroms as usual. That is,halting has no effect on bus arbitartion. It is the bus arbitration function that removes (i.e.,three-states) 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.

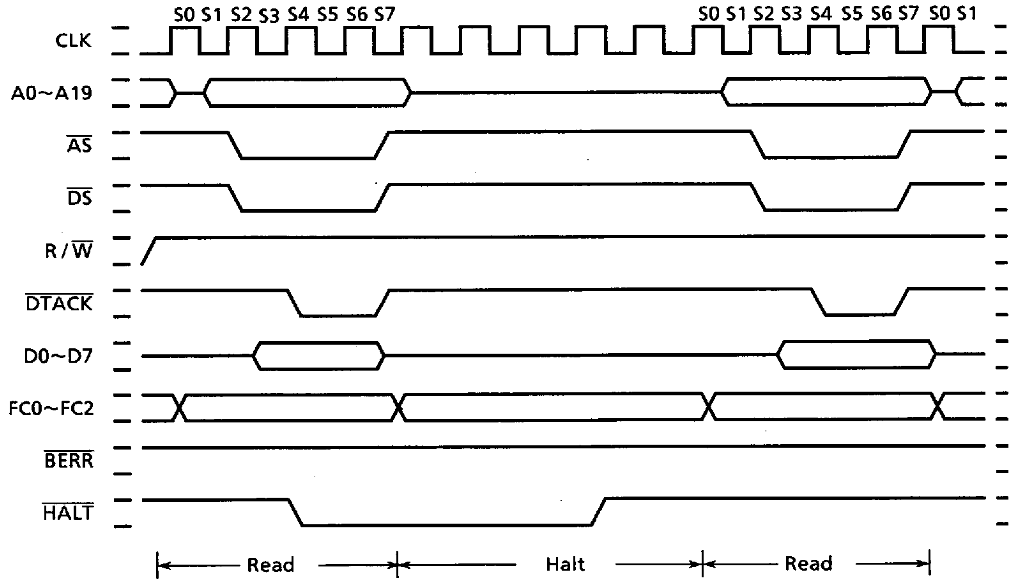


Figure 4.22 $\overline{\text{HALT}}$ Operation Timing Diagram

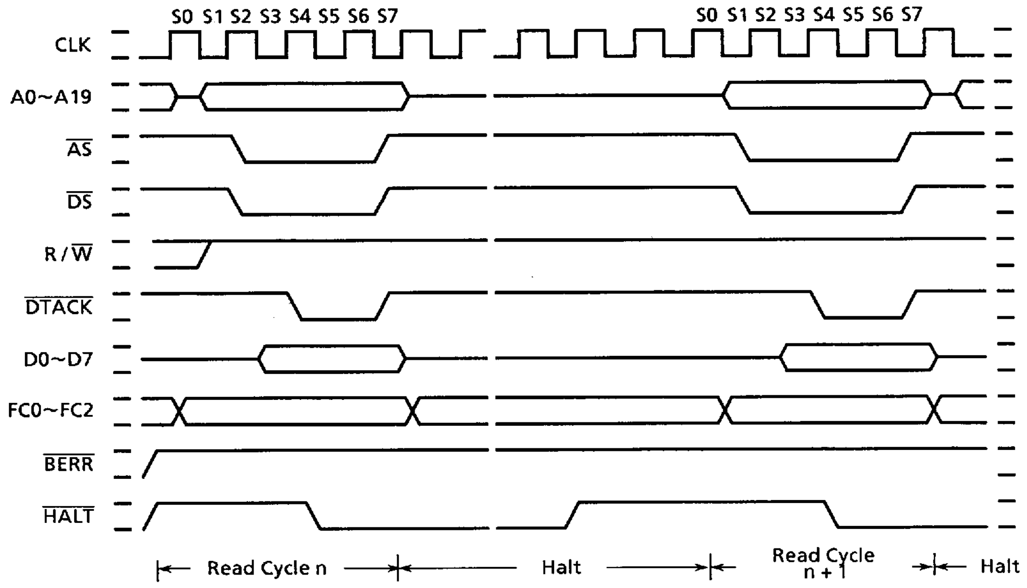


Figure 4.23 $\overline{\text{HALT}}$ Signal Single-Step Operation Timing Characteristics

4.2.4.4 Double Bus Faults

When a bus error exception occurs,the processor will attempt to stack several words containing information about the state of the machine. If a bus error exception occurs during the stacking operation,there have been two bus errors in a row. This is commonly referred to as a double bus fault. When a double bus fault occurs,the processor will halt. Once a bus error exception has occurred,any bus error exception occurring before the execution of the next instruction constitutes a double bus fault. Figure 4.24 is a diagram of the bus error timing.

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 bue 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.

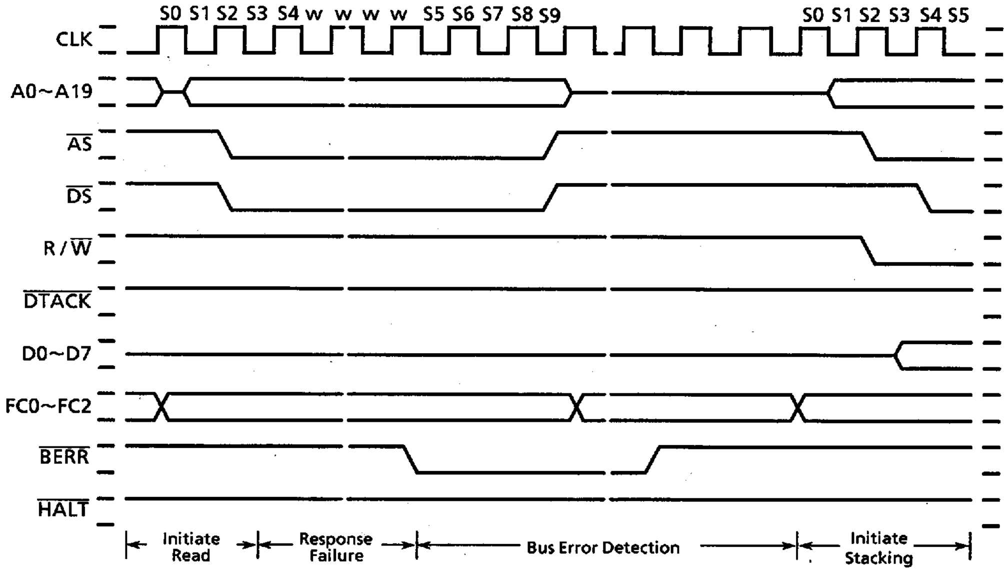


Figure 4.24 Bus Error Timing Diagram

4.2.5 Reset Operation

The $\overline{\text{RESET}}$ signal is a bidirectional signal that allows either the processor or the an external signal to reset the system. Figure 4.25 is a timing diagram for processor generated reset operation.

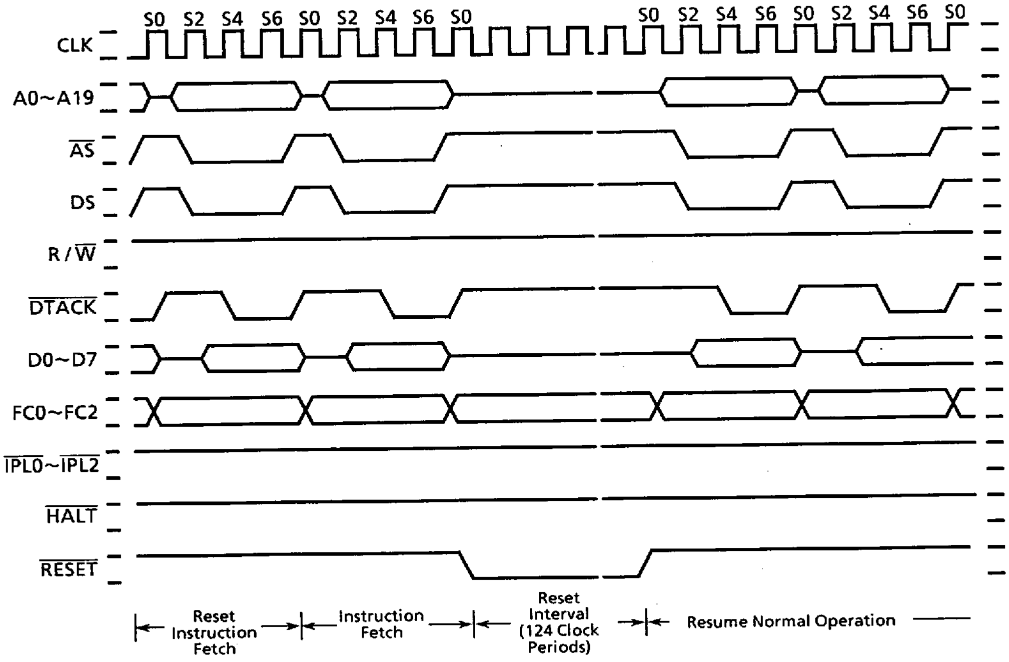


Figure 4.25 Reset Operation Timing Diagram

When the reset and halt lines are driven it is recognized as an entire system reset, including the processor. For an external reset, both the $\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ lines must be asserted to ensure total reset of the processor. Timing diagram for system reset is shown in Figure 4.26. 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 processor' 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 10 clock cycles will cause a processor reset, except when Vcc is initially applied to the processor. In this case, an external reset must be applied for at least 100 milliseconds allowing stabilization of the on-chip circuitry and system clock.

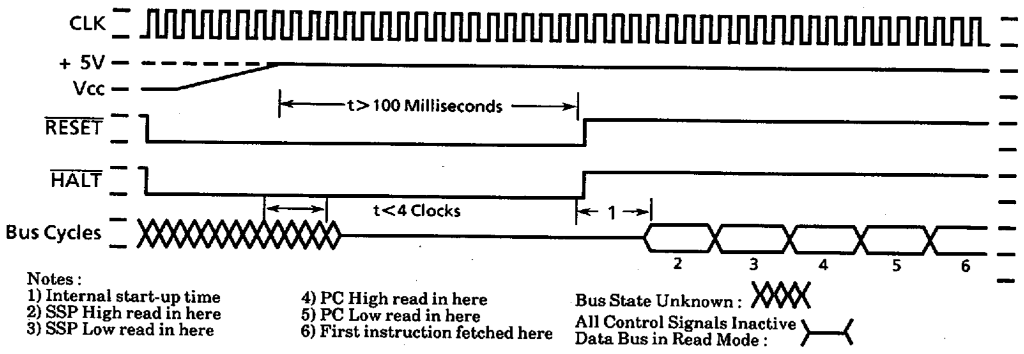


Figure 4.26 System Reset Timing Diagram

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

In order to properly control termination of a bus cycle for a re-run or a bus error condition, $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ should be asserted and negated on the rising edge of the TMP68008 clock. This will assrre that when two signals are asserted simultaneously, the required setup time (#47) for both of them will be met during the same bus state.

This, or some equivalent precaution, should be designed external to the TMP68008. Parameter #48 is intended to ensure this operation in a totally asynchronous system, and may be ignored if the above conditions are met.

The perferred bus cycle terminations may be summarized as follows (case numbers refer to Table 4.4):

Normal Termination	:	$\overline{\text{DTACK}}$ occurs first (case 1).
Halt Termination	:	$\overline{\text{HALT}}$ is asserted at same time, or precedes $\overline{\text{DTACK}}$ (no $\overline{\text{BERR}}$) cases 2 abd 3.
Bus Error Termination	:	$\overline{\text{BERR}}$ is asserted in lieu of, at same time, or preceding $\overline{\text{DTACK}}$ (Case 4); $\overline{\text{BERR}}$ negated at same time, or arter $\overline{\text{DTACK}}$.
Re-Run Termination	:	$\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ asserted at the same time, or before $\overline{\text{DTACK}}$ (cases 5 and 6); $\overline{\text{HALT}}$ must be held at least one cycle after $\overline{\text{BERR}}$.

Table 4.4 details the resulting bus cycle termination under various combinations of control signal sequences. The negation of these same control signals under several conditions is shown in Table 4.5 ($\overline{\text{DTACK}}$ is assumed to be negated normally in all cases; for correct results, both $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ should be negated when address strobe is negated).

Table 4.4 $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, 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 $\overline{\text{HALT}}$ removed.
4	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	X A NA	X S NA	Terminate and re-run.
5	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA A A	X S S	Terminate and re-run.
6	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA NA A	X A S	Terminate and re-run when $\overline{\text{HALT}}$ Removed.

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 an remains asserted in this state.

Table 4.5 $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ Negation Results

Conditions of Termination in Table 4.4	Control Signal	Negated on Rising Edge of State			Results – Next Cycle
		N		N + 2	
Bus Error	$\overline{\text{BERR}}$	●	or	●	Takes bus error trap.
	$\overline{\text{HALT}}$	●	or	●	
Re-run	$\overline{\text{BERR}}$	●	or	●	Illegal sequence; usually traps to vector number 0.
	$\overline{\text{HALT}}$	●			
Re-run	$\overline{\text{BERR}}$	●			Re-runs the bus cycle.
	$\overline{\text{HALT}}$			●	
Normal	$\overline{\text{BERR}}$	●			May lengthen next cycle.
	$\overline{\text{HALT}}$	●	or	●	
Normal	$\overline{\text{BERR}}$			●	If next cycle is started it will be terminated as a bus error.
	$\overline{\text{HALT}}$	●	or	none	

● : Signal is negated in this bus state.

EXAMPLE A : Asystem 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 5), or if valid, return $\overline{\text{DTACK}}$.

(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).

(c)return $\overline{\text{DTACK}}$ prior to data verification, as described in previous section. If data invalid, $\overline{\text{BERR}}$ is asserted (case 1) in next cycle. Error-handling software must know how to recover error cycle.

4.4 ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION

4.4.1 Asynchronous Operation

To achieve clock frequency independence at a system level, the TMP68008 can be used in an asynchronous manner. This entails using only the bus handshake lines ($\overline{\text{AS}}$, $\overline{\text{DS}}$, $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, $\overline{\text{HALT}}$, and $\overline{\text{VPA}}$) to control the data transfer. Using this method, AS signals the start of a bus cycle and the data strobes are used as a condition for valid data on a write cycle. The slave device (memory or peripheral) then responds by placing the requested data on the data bus for a read cycle or latching data on a write cycle and asserting the data transfer acknowledge signal ($\overline{\text{DTACK}}$) to terminate the bus cycle. If no slave responds or the access is invalid, external control logic asserts the $\overline{\text{BERR}}$, or $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ signal to abort or rerun the bus cycle.

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

4.4.2 Synchronous Operation

To allow for those systems which use the system clock as a signal to generate $\overline{\text{DTACK}}$ and other asynchronous inputs, the asynchronous input setup time is given as parameter #47. If this setup is met on an input, such as $\overline{\text{DTACK}}$, the processor is guaranteed to recognize that signal on the next falling edge of the system clock. However, the converse is not true-if the input signal dose not meet the setup time it is not guaranteed not to be recognized.In addition, if $\overline{\text{DTACK}}$ is recognized on a falling edge, valid data will be latched into the processor (on a read cycle) on the next falling edge provided that the data meets the setup time given as paramater #27. Given this, paramater #31 may be ignored. Note that if $\overline{\text{DATCK}}$ 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, $\overline{\text{VPA}}$ and $\overline{\text{BERR}}$ are sampled on every falling edge of the clock starting with S0. $\overline{\text{DTACK}}$is sampled on every falling edge of the clock starting with S4 and data is latched on the falling edge of S6 during a read. The bus cycle will then be terminated in S7 except when $\overline{\text{BERR}}$ is asserted in the absence of $\overline{\text{DTACK}}$, in which case it will terminate one clock cycle later in S9.

5. PROCESSING STATES

This section describes the actions of the TMP68008 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 is detailed.

The TMP68008 is always in one of three processing states: normal, exception, or halted. The normal processing stateis 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 proccessor enters when a STOP instruction is executed. In this state, no further memory 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 bya reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

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

5.1 PRIVILEGE STATES

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

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 perfroms the overhead tasks for the user state programs.

5.1.1 Supervisor State

The supervisor state is the higher state of privilege. For instruction execution, the supervisor state is determined by the S bit of the status register; if the S bit is asserted (high) or exception processing is invoked, 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.

5.1.2 User State

The user state is the lower state of privilege and is controlled by the S bit of the status register. If the S bit is negated (low), the processor is executing instructions in the user state. 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.

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

5.1.3 Privilege State Changes

Once the processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the 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.

5.1.4 Reference Classification

When the processor makes a reference, it classifies the kind of reference being made, using the encoding on the three function code output lines. This allows external translation of addresses, control of access, and differentiation of special processor states, such as interrupt acknowledge. Table 5.1 lists the classification of references.

Table 5.1 Reference Classification

Function Code Output			Reference Class
FC2	FC1	FC0	
L	L	L	(Unassigned)
L	L	H	User Data
L	H	L	User Program
L	H	H	(Unassigned)
H	L	L	(Unassigned)
H	L	H	Supervisor Data
H	H	L	Supervisor Program
H	H	H	Interrupt Acknowledge

5.2 EXCEPTION PROCESSING

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

5.2.1 Exception Vectors

Exception vectors are memory locations from which the processor fetchse the address of a routine which will handle that exception, All exception vectors are two words in length (Figure 5.1), except for the reset vector, which is four words. All exception vectors lie in the supervisor data space, except for the reset vector which is in the supervisor program space. A vector number is an 8-bit number which, when multiplied by four, gives the address of an exception vector. Vector numbers are generated internally or externally, depednding on the cause of the exception. In the case of vectord interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 5.2) to the processor on data bus lines D0 through D7. The processor translates the vector number into a full 32-bit address, as shown in Figure 5.3. The memory layout for exception vectors is given in Table 5.2.

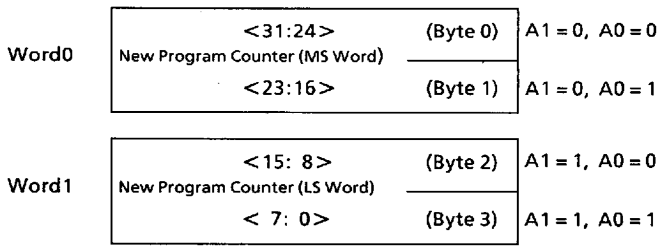


Figure 5.1 Format of Vector Table Entries

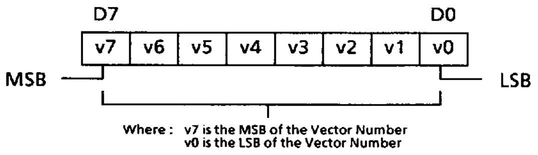

D7
D0
v7
v6
v5
v4
v3
v2
v1
v0
MSB
LSB
Where : v7 is the MSB of the Vector Number
v0 is the LSB of the Vector Number

Figure 5.2 Vector Number Format

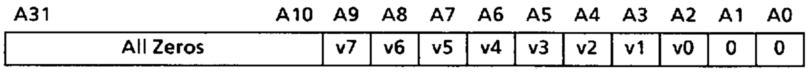


Figure 5.3 Vector Number Translated to an Address

As shown in Table 5.2, 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.

Table 5.2 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	040	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		–

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

5.2.2 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 inputs are used for access control and processor restart. The internally generated exceptions come from instructions, or from address errors or tracing. The trap (TRAP), trap on overflow (TRAPV), check register against 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.

5.2.3 Exception Processing Sequence

Exception processing occurs in four identifiable steps.In the first step, an internal copy is made of the status register. After the copy is made, the S bit is asserted, putting the processor into the supervisor privilege state.Also, the T bit is negated which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

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

5.2.4 Multiple Exceptions

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

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

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Threrefore, 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 5.3.

Table 5.3Exception Grouping and Priority

Group	Exection	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

5.3 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.

5.3.1 Reset

The reset input provides the highest exception level. The processing of the reset signal is designed for system initiation, and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state, 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.

5.3.2 Interrupts

Seven levels of interrupts are provided by the TLCS-68000 architecture. The TMP68008 supports three interrupt levels: two, five, and seven, level seven being the highest. Devices may be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. 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 requset is made to the processor by encoding the interrupt requset level on the interrupt request lines; a zero indicates no interrupt requset. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts are detected between instruction execution. 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. First a copy of the status register is saved, and the privilege state is set to supervisor, tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged.The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests 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 processimg, 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 5.4, a timing diagram is given in Figure 5.5, and the interrupt processing sequence is shown in Figure 5.6.

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 requset level is a seven and the processor priority is set to a lower level by an instruction.

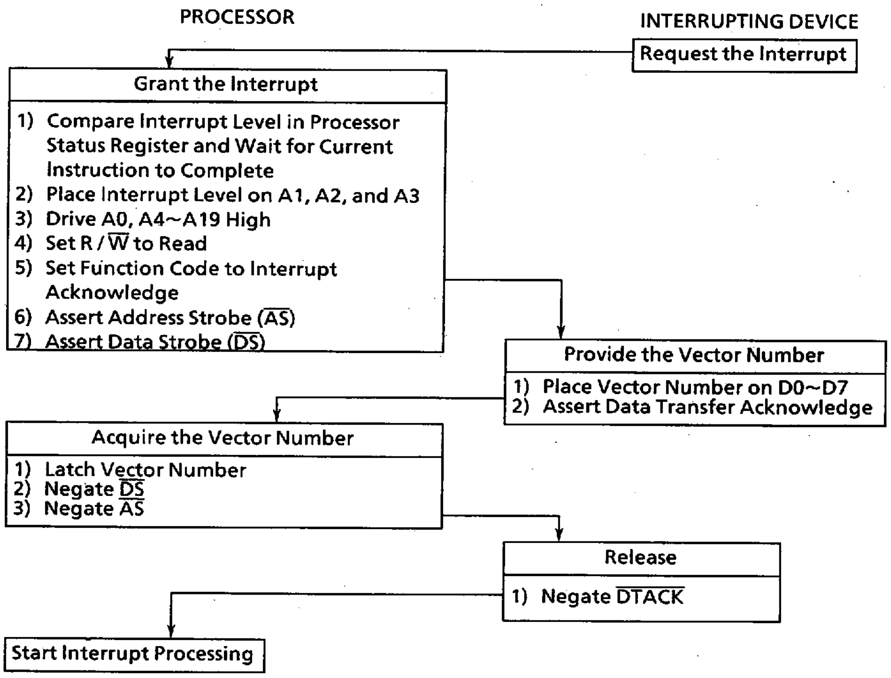

PROCESSOR
INTERRUPTING DEVICE
Request the Interrupt
Grant the Interrupt
1) Compare Interrupt Level in Processor Status Register and Wait for Current Instruction to Complete
2) Place Interrupt Level on A1, A2, and A3
3) Drive A0, A4~A19 High
4) Set R / $\overline{W}$ to Read
5) Set Function Code to Interrupt Acknowledge
6) Assert Address Strobe ($\overline{AS}$)
7) Assert Data Strobe ($\overline{DS}$)
Provide the Vector Number
1) Place Vector Number on D0~D7
2) Assert Data Transfer Acknowledge
Acquire the Vector Number
1) Latch Vector Number
2) Negate $\overline{DS}$
3) Negate $\overline{AS}$
Release
1) Negate $\overline{DTACK}$
Start Interrupt Processing

Figure 5.4 Vector Acquisition Flowchart

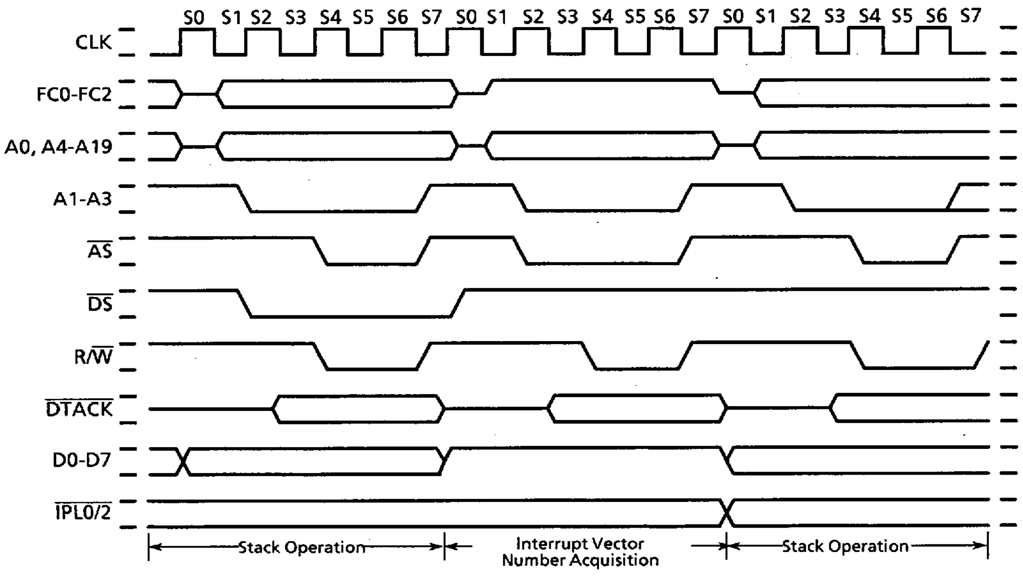


Figure 5.5 Interrupt Acknowledge Cycle

1. Acquire vector number via interrupt acknowledge.
2. Convert vector number to a full 32-bit address.
3. Stack the SR and PC by successive write cycles. Refer to Figure 4.7 for word write cycle operation.
4. Place vector table address on A0~A19. Refer to Figure 5.3 for address translation.
5. Read upper half of program counter (PC). Refer to Figure 4.3 for word read cycle operation
6. Increment vector table address by 2 and place it on A0~A19.
7. Read lower half of program counter (PC) .
8. Load new program counter (PC) .
9. Place contents of PC on A0~A19.
10. Read first instruction of service routine.

Figure 5.6 Interrupt Processing Sequence

5.3.3 Uninitialzed Interrupt

An interrupting device asserts $\overline{VPA}$ or provides an interrupt vector during an interrupt acknowledge cycle to the TMP68008. If the vector register of the peripheral has not been initialized, the responding TLCS-68000 Family peripheral will provide vector 15 ($OF), the uninitialized interrupt vector. This provides a uniform way to recover from a programming error.

5.3.4 Spurious Interrupt

If during the interrupt acknowledge cycle no device responds by asserting $\overline{\text{DTACK}}$ or $\overline{\text{VPA}}$, 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.

5.3.5 Instruction Traps

Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution. Or from use of instructions whose normal behavior is trapping. 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 divide (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

5.3.6 Illegal 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. Motorola reserves the right to define instructions whose opcodes may be any of the illegal instructions. Three bit patterns will always force an illegal instruction trap on all TLCS-68000 Family compatible microprocessor. They are;$4AFA, $4AFB, and$4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for Motrola system products. The third pattern, $4AFC, is reserved for customer use.

Word patterns whith 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.

5.3.7 Privilege Violations

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

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

5.3.8 Tracing

To aid in program development, the TMP68008 includes a facility to allow instruction-by-instruction tracing. In the trace state, after each instruction is executed an exception is forced, allowing a debugging program to monitor the execution of the program under test.

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

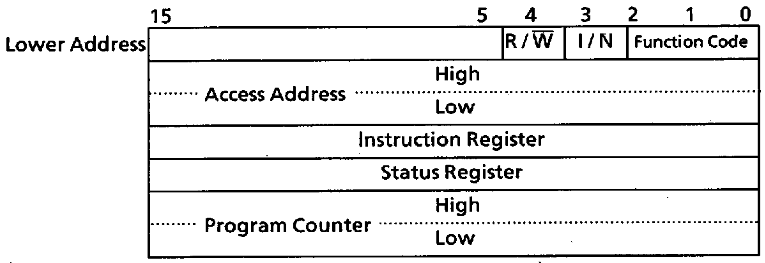
As an extreme illustration of the above rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt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

5.3.9 Bus Error

Bus error exceptions occur when 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. Regardless of whether the processor was doing instruction or exception processing, that processing is terminatd, and the processor immediately begins exception processing.

Exception processing for 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 (refer to Figure 5.7). 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, two to ten bytes 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 the processor was processing an instruction or not, 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 group 1 exception. Figure 5.7 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 the vector.It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, 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 all memory contents. only the $\overline{\text{RESET}}$ pin can restart a halted processor.



R / $\overline{W}$ (read / write) : write = 0, read = 1.
I / N (instruction / not) : instruction = 0, not = 1

Figure 5.7 Supervisor Stack Order (Group 0)

5.3.10 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. When the TMP68008 detects an address error it prevents assertion of $\overline{\text{DS}}$ but asserts $\overline{\text{AS}}$ to provide proper bus arbitration support. The effect is much like an internally generated bus error, in that the bus cycle is aborted, and the processor ceases whatever processing it is currently doing and begins exception processing.After exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector 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 5.8, an address error will execute a short bus cycle followed by exception processing.

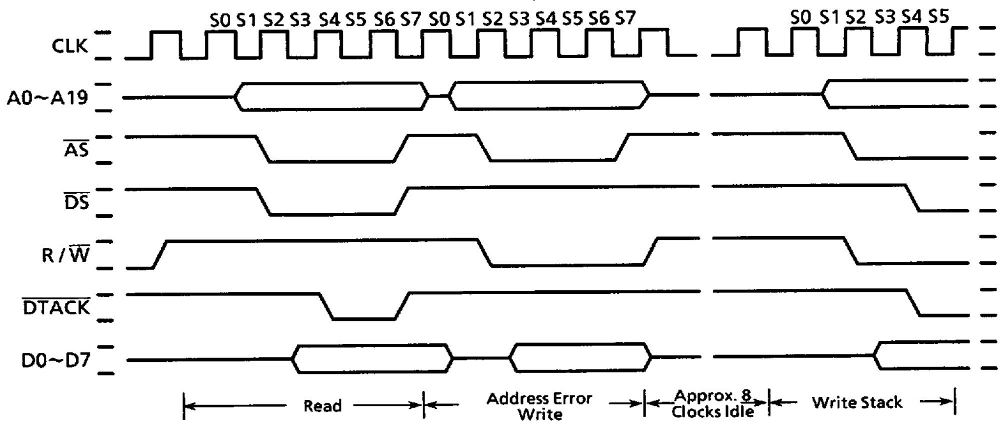


Figure 5.8 Address Error Timing

6. INTERFACE WITH 6800 PERIPHERALS

The extensive line of 6800 peripherals are compatible with the TMP68008. Some of these devices that are particularly useful are:

6821	Peripheral Interface Adapter
6840	Programmable Timer Module
6845	CRT Controller
6850	Asynchronous Communications Interface Adapter
6852	Synchronous Serial Data Adapter
6854	Advanced Data Link Controller
68488	General Purpose Interface Adapter

To interface the synchronous 6800 peripherals with the asynchronous TMP68008,the processor modifies its bus cycle to meet the 6800 cycle requarements whenever an 6800 device address is detected. This is possible since both processors use memory mapped I/O. Figure 6-1 is a flowchart of the interface operation between the processor and 6800 devices.

6.1 DATA TRANSFER OPERATION

Two signals on the processor provide the 6800 interface. They are:enable (E),and valid peripheral address ($\overline{\text{VPA}}$). In addition,a valid memory address ($\overline{\text{VMA}}$) signal must be provided (see 4.1.7 6800 Peripheral Control). Enable corresponds to the E signal in existing 6800 systems. The E clock frequency is one tenth of the incoming TMP68008 clock frequency. The timing of E allows 1 MHz peripherals to be used with an 8 MHz TMP68008.Enable has a 60/40 duty cycle;that is,it is low for six input clocks and high for input clocks.

6800 cycle timing is given in Section 8. At state zero 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 one,the address bus is released from the high-impedance state.

During state two,the address strobe ($\overline{\text{AS}}$) is asserted to indicate that there is a valid address on the address bus. If the bus cycle is a read cycle,the data strobe is also asserted in state two. If the bus cycle is a write cycle the read/write (R/$\overline{\text{W}}$) signal is switched to low (write) during state two. One half clock later,in state three,the write data is placed on the data bus,and in state four tha data strobe is issued to indicate valid data on the data bus. The processor now insert wait states until it recognizes the assertion of $\overline{\text{VPA}}$.

The $\overline{\text{VPA}}$ input signals the processor that the address on the bus is the address of an 6800 device (or an area reserved for 6800 devices) and that the bus should conform to the transfer characteritics of the 6800 bus. Valid peripheral address is derived by decoding the address bus, conditioned by address strobe. Chip select for the 6800 peripherals should be derives by decoding the address bus $\overline{\text{VMA}}$ (not $\overline{\text{AS}}$).

After recognition of $\overline{\text{VPA}}$,the processor assures that the enable (E) is low,by waiting if necessary. Valid memory address (provided by an external circuit similar to that of Figure 4.2) is then used as part of chip select equation of the peripheral. This ensures that the 6800 peripherals are selected and deselected at the correct time.The peripheral now runs its cycle during the high portion of the E signal. Figure 6.2 depicts the 6800 cycle timing using the VMA generation circuit shown in Figure 4.2. This cycle length is dependent strictly upon when $\overline{\text{VAP}}$ is asserted in relationship to the E clock.

During a read cycle,the processor latches the peripheral data in state six. For all cycles,the processor negates the address and data strobes one half clock cycle later in state seven,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 $\overline{\text{VPA}}$ within one clock after address strobe is negated.

$\overline{\text{DTACK}}$ should not be asserted while $\overline{\text{VPA}}$ is asserted. Notice that $\overline{\text{VMA}}$ is active low,contrasted with the active high 6800 VMA. Refer to Figure 4.2.

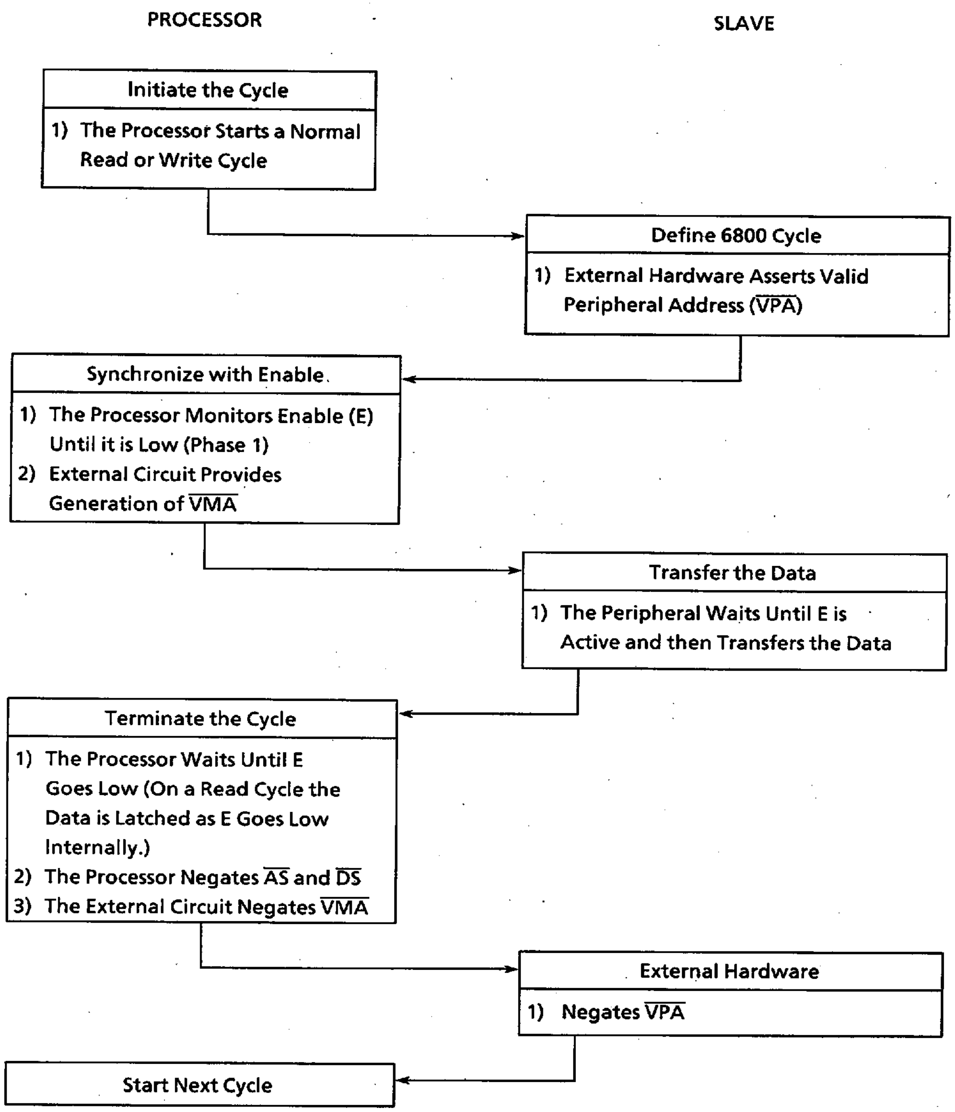

PROCESSOR
SLAVE
Initiate the Cycle
1) The Processor Starts a Normal Read or Write Cycle
Define 6800 Cycle
1) External Hardware Asserts Valid Peripheral Address ($\overline{VPA}$)
Synchronize with Enable
1) The Processor Monitors Enable (E) Until it is Low (Phase 1)
2) External Circuit Provides Generation of $\overline{VMA}$
Transfer the Data
1) The Peripheral Waits Until E is Active and then Transfers the Data
Terminate the Cycle
1) The Processor Waits Until E Goes Low (On a Read Cycle the Data is Latched as E Goes Low Internally.)
2) The Processor Negates $\overline{AS}$ and $\overline{DS}$
3) The External Circuit Negates $\overline{VMA}$
External Hardware
1) Negates $\overline{VPA}$
Start Next Cycle

Figure 6.1 6800 Cycle Flowchart

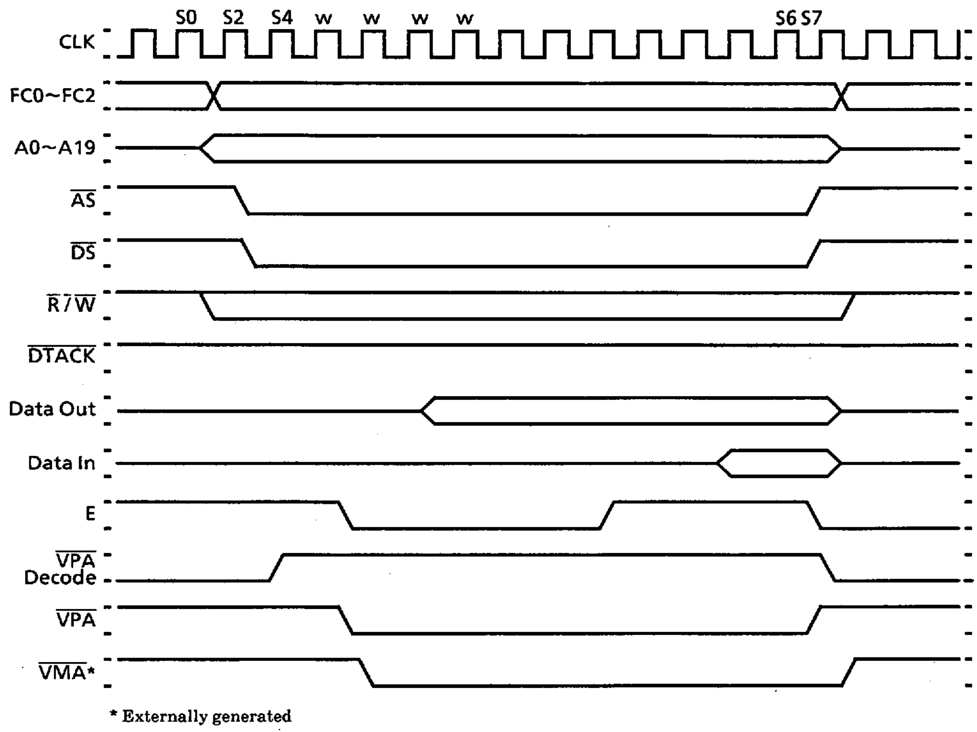


Figure 6.2 6800 Cycle Timing

6.2 AC ELECTRICAL SPECIFICATIONS

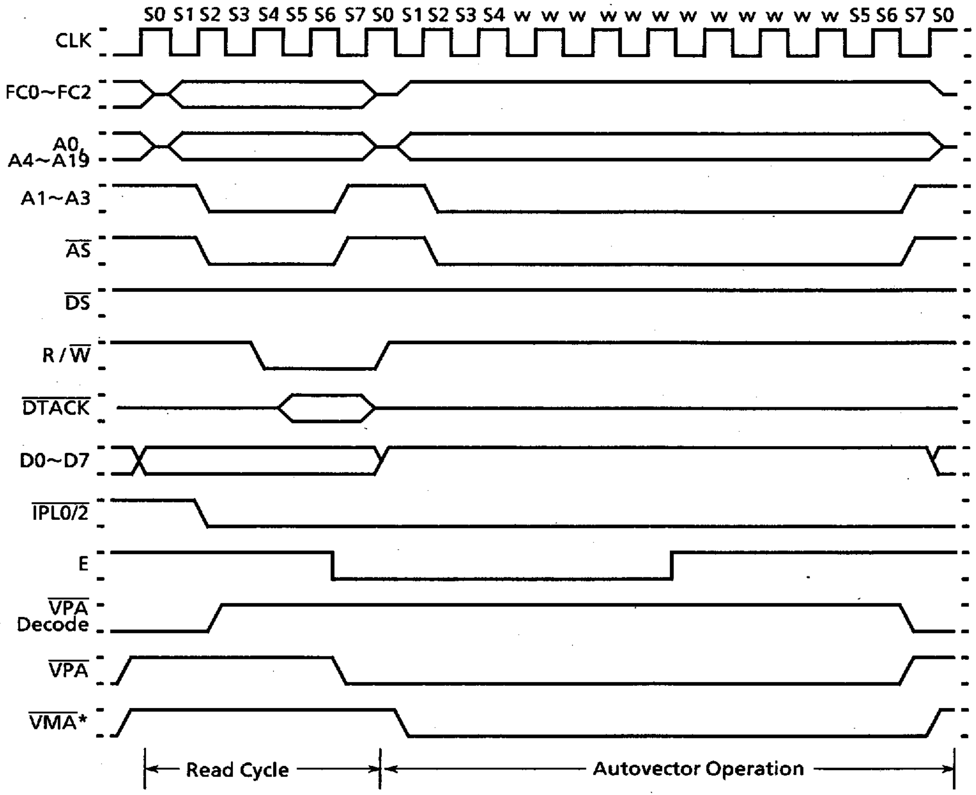
The electrical specifications for interfacing the TMP68008 to 6800 Family peripherals are located in Section 8.

6.3 INTERRUPT INTERFACE OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector,the $\overline{\text{VPA}}$ is asserted,the TMP68008 will complete a normal 6800 read cycle as shown in Figure 6.3. 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 25~31 (decimal).

Autovectoring operates in the same fashion (but is not restricted to) the 6800 interrupt sequence. The basic difference is that there are six normal interrupt vectors and one NMI type vector. As with both the 6800 and the TMP68008'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,the 6800 peripheral address decoding should prevent unintended accesses.



* : Externally generated

Figure 6.3 Autovector Operation Timing Diagram

7 INSTRUCTION SET AND EXECUTION TIMES

7.1 INSTRUCTION SET

The following paragraphs provide information about the in addressing categories and instruction set of the TMP68008.

7.1.1 Addressing Categories

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

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

These categories may be combined,so that additional,more restrictive,classifications may be defined. For example,the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses,and the latter refer to addressing modes which are both data and alterable. Table 7.1 shows the various categories to which each of the effective address modes belong. Table 7.2 is the instruction set summary.

Table 7.1 Effective Addressing Mode Categories

Effective Address Modes	Mode	Register	Data	Addressing Categories		
				Memory	Control	Alterable
Dn	000	Register Number	×	—	—	×
An	001	Register Number	—	—	—	×
(An)	010	Register Number	×	×	×	×
(An) +	011	Register Number	×	×	—	×
− (An)	100	Register Number	×	×	—	×
d16 (An)	101	Register Number	×	×	×	×
d8 (An, Xn)	110	Register Number	×	×	×	×
Abs.W	111	000	×	×	×	×
Abs.L	111	001	×	×	×	×
d16 (PC)	111	010	×	×	×	—
d8 (PC, Xn)	111	011	×	×	×	—
# xxx	111	100	×	×	—	—

Table 7.2 Instruction Set (Sheet 1 of 3)

Mnemonic	Operation	Condition Codes				
		X	N	Z	V	C
ABCD	$(\text{Destination})_{10} + (\text{Source})_{10} + x \rightarrow$ Destination	*	U	*	U	*
ADD	(Destination) + (Source) → Destination	*	*	*	*	*
ADDA	(Destination) + (Source) → Destination	—	—	—	—	—
ADDI	(Destination) + Immediate Data → Destination	*	*	*	*	*
ADDQ	(Destination) + Immediate Data → Destination	*	*	*	*	*
ADDX	(Destination) + (Source) + x → Destination	*	*	*	*	*
AND	(Destination) ∧ (Source) + x → Destination	—	*	*	0	0
ANDI	(Destination) ∧ Immediate Data → Destination	—	*	*	0	0
ASL, ASR	(Destination) Shifted by <count> → Destination	*	*	*	*	*
Bcc	If CC then PC + d → PC	—	—	—	—	—
BCHG	~ (<bit number>) OF Destination → Z ~ (<bit number>) OF Destination → <bit number> OF Destination	—	—	*	—	—
BCLR	~ (<bit number>) OF Destination → Z 0 → <bit number> OF Destination	—	—	*	—	—
BRA	PC + displacement → PC	—	—	—	—	—
BSET	~ (<bit number>) OF Destination → Z 1 → <bit number> OF Destination	—	—	*	—	—
BSR	PC → - (SP), PC + d → PC	—	—	—	—	—
BTST	~ (<bit number>) OF Destination → Z	—	—	*	—	—
CHK	If Dn < 0 or Dn > (<ea>) then TRAP	—	*	U	U	U
CLR	0 → Destination	—	0	1	0	0
CMP	(Destination) – (Source)	—	*	*	*	*
CMPA	(Destination) – (Source)	—	*	*	*	*
CMPI	(Destination) – Immediate Data	—	*	*	*	*
CMPM	(Destination) – (Source)	—	*	*	*	*
DBcc	If ~CC then Dn – 1 → Dn; if Dn ≠ – 1 then PC + d → PC	—	—	—	—	—
DIVS	(Destination) / (Source) → Destination	—	*	*	*	0
DIVU	(Destination) / (Source) → Destination	—	*	*	*	0
EOR	(Destination) ⊕ (Source) → Destination	—	*	*	0	0
EORI	(Destination) ⊕ Immediate Data: → Destination	—	*	*	0	0
EXG	Rx ↔ Ry	—	—	—	—	—
EXT	(Destination) Sign-extended → Destination	—	*	*	0	0

Table 7.2 Instruction Set (Sheet 2 of 3)

Mnemonic	Operation	Condition Codes				
		X	N	Z	V	C
JMP	Destination → PC	—	—	—	—	—
JSR	PC → – (SP) ; Destination → PC	—	—	—	—	—
LEA	Destination → An	—	—	—	—	—
LINK	An → – (SP); SP → An; SP + displacement → SP	—	—	—	—	—
LSL, LSR	(Destination) Shifted by <count> → Destination	*	*	*	0	*
MOVE	(Source) → Destination	—	*	*	0	0
MOVE to CCR	(Source) → CCR	*	*	*	*	*
MOVE to SR	(Source) → SR	*	*	*	*	*
MOVE from SR	SR → Destination	—	—	—	—	—
MOVE USP	USP → An; An → USP	—	—	—	—	—
MOVEA	(Source) → Destination	—	—	—	—	—
MOVEM	Registers → Destination (Source) → Registers	—	—	—	—	—
MOVEP	(Source) → Destination	—	—	—	—	—
MOVEQ	Immediate Data → Destination	—	*	*	0	0
MULS	(Destination) × (Source) → Destination	—	*	*	0	0
MULU	(Destination) × (Source) → Destination	—	*	*	0	0
NBCD	0 – $(\text{Destination})_{10}$ – x → Destination	*	U	*	U	*
NEG	0 – (Destination) → Destination	*	*	*	*	*
NEGX	0 – (Destination) – x → Destination	*	*	*	*	*
NOP	—	—	—	—	—	—
NOT	~ (Destination) → Destination	—	*	*	0	0
OR	(Destination) ∨ (Source) → Destination	—	*	*	0	0
ORI	(Destination) ∨ Immediate Data → Destination	—	*	*	0	0
PEA	Destination → – (SP)	—	—	—	—	—
RESET	—	—	—	—	—	—
ROL, ROR	(Destination) Rotated by <count> → Destination	—	*	*	0	*
ROXL, ROXR	(Destination) Rotated by <count> → Destination	*	*	*	0	*
RTE	(SP) + → SR; (SP) + → PC	*	*	*	*	*
RTR	(SP) + → CC; (SP) + → PC	*	*	*	*	*
RTS	(SP) + → PC	—	—	—	—	—
SBCD	$(\text{Destination})_{10}$ – $(\text{Source})_{10}$ – x → Destination	*	U	*	U	*

Table 7.2 Instruction Set (Sheet 3 of 3)

Mnemonic	Operation	Condition Codes				
		X	N	Z	V	C
Scc	If CC then 1's → Destination else 0's → Destination	—	—	—	—	—
STOP	Immediate Data → SR; STOP	*	*	*	*	*
SUB	(Destination) − (Source) → Destination	*	*	*	*	*
SUBA	(Destination) − (Source) → Destination	—	—	—	—	—
SUBI	(Destination) − Immediate Data → Destination	*	*	*	*	*
SUBQ	(Destination) − Immediate Data → Destination	*	*	*	*	*
SUBX	(Destination) − (Source) − x → Destination	*	*	*	*	*
SWAP	Register [31:16] ↔ Register [15:0]	—	*	*	0	0
TAS	(Destination) Tested → CC; 1 → [7] OF Destination	—	*	*	0	0
TRAP	PC → − (SSP) ; SR → − (SSP) ; (Vector) → PC	—	—	—	—	—
TRAPV	If V then TRAP	—	—	—	—	—
TST	(Destination) Tested → CC	—	*	*	0	0
UNLK	An → SP; (SP) + → An	—	—	—	—	—

→	the left operand is moved to the right operand	*	affected
↔	the two operands are exchanged	−	unaffected
+	the operands are added	0	cleared
−	the right operand is subtracted from the left operand	1	set
*	the operands are multiplied	U	undefined
/	the first operand is divided by the second operand		
∧	the operands are logically ANDed		
∨	the operands are logically ORed		
⊕	the operands are logically exclusively ORed		
<	relational test, true if left operand is less than right operand		
>	relational test, true if left operand is greater than right operand		
~	logical complement		
[ ]	bit number		

7.1.2 Instruction Prefetch

The TMP68008 uses a two-word tightly-coupled instruction prefetch mechanism to enhance preformance. 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 multiword instructions, as each additional word of the instruction is used internally, a fetch is mode to the instruction stream to repace it.

(3) The last fetch 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 superflouous fetch. In the case of an interrupt or trace exception, neither word is used.

(5) The program counter usually points to the last word fetched from the instruction stream.

7.2 INSTRUCTION EXECUTION TIMES

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

7.2.1 Operand Effective Address Calculation Times

Table 7-3 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 parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

Table 7.3 Effective Address Calculation Times

Addressing Mode		Byte	Word	Long
	Register			
Dn	Data Register Direct	0 (0 / 0)	0 (0 / 0)	0 (0 / 0)
An	Address Register Direct	0 (0 / 0)	0 (0 / 0)	0 (0 / 0)
	Memory			
(An)	Address Register Indirect	4 (1 / 0)	8 (2 / 0)	16 (4 / 0)
(An) +	Address Register Indirect with Postincrement	4 (1 / 0)	8 (2 / 0)	16 (4 / 0)
− (An)	Address Register Indirect with Predecrement	6 (1 / 0)	10 (2 / 0)	18 (4 / 0)
d16 (An)	Address Register Indirect with Displacement	12 (3 / 0)	16 (4 / 0)	24 (6 /0)
d8 (An, Xn)*	Address Register Indirect with Index	14 (3 / 0)	18 (4 / 0)	26 (6 / 0)
Ads.W	Absolute Short	12 (3 / 0)	16 (4 / 0)	24 (6 / 0)
Ads.L	Absolute Long	20 (5 / 0)	24 (6 / 0)	32 (8 / 0)
d16 (PC)	Program Counter with Displacement	12 (3 / 0)	16 (4 / 0)	24 (6 / 0)
d8 (PC, Xn)	Program Counter with Index	14 (3 / 0)	18 (4 / 0)	26 (6 / 0)
#xxx	Immediate	8 (2 / 0)	8 (2 / 0)	16 (4 / 0)

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

7.2.2 Move Instruction Execution Times

Table 7.4, 7.5, and 7.6 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 parenthsis as: (r/w).

Table 7.4 Move Byte Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	− (An)	d16(An)	d8 (An.Xn)*	Ads.W	Ads.L
Dn	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	20 (4/1)	22 (4/1)	20 (4/1)	28 (6/1)
An	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	20 (4/1)	22 (4/1)	20 (4/1)	28 (6/1)
(An)	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	24 (5/1)	26 (5/1)	24 (5/1)	32 (7/1)
(An) +	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	24 (5/1)	26 (5/1)	24 (5/1)	32 (7/1)
− (An)	14 (3/0)	14 (3/0)	18 (3/1)	18 (3/1)	18 (3/1)	26 (5/1)	28 (5/1)	26 (5/1)	34 (7/1)
d16(An)	20 (5/0)	20 (5/0)	24 (5/1)	24 (5/1)	24 (5/1)	32 (7/1)	34 (7/1)	32 (7/1)	40 (9/1)
d8 (An. Xn)*	22 (5/0)	22 (5/0)	26 (5/1)	26 (5/1)	26 (5/1)	34 (7/1)	36 (7/1)	34 (7/1)	42 (9/1)
Ads.W	20 (5/0)	20 (5/0)	24 (5/1)	24 (5/1)	24 (5/1)	32 (7/1)	34 (7/1)	32 (7/1)	40 (9/1)
Ads.L	28 (7/0)	28 (7/0)	32 (7/1)	32 (7/1)	32 (7/1)	40 (9/1)	42 (9/1)	40 (9/1)	48(11/1)
d16 (PC)	20 (5/0)	20 (5/0)	24 (5/1)	24 (5/1)	24 (5/1)	32 (7/1)	34 (7/1)	32 (7/1)	40 (9/1)
d8 (PC, Xn)*	22 (5/0)	22 (5/0)	26 (5/1)	26 (5/1)	26 (5/1)	34 (7/1)	36 (7/1)	34 (7/1)	42 (9/1)
#xxx	16 (4/0)	16 (4/0)	20 (4/1)	20 (4/1)	20 (4/1)	28 (6/1)	30 (6/1)	28 (6/1)	36 (8/1)

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

Table 7.5 Move Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	− (An)	d16 (An)	d8 (An.Xn)*	Ads.W	Ads.L
Dn	8 (2/0)	8 (2/0)	16 (2/2)	16 (2/2)	16 (2/2)	24 (4/2)	26 (4/2)	20 (4/2)	32 (6/2)
An	8 (2/0)	8 (2/0)	16 (2/2)	16 (2/2)	16 (2/2)	24 (4/2)	26 (4/2)	20 (4/2)	32 (6/2)
(An)	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	32 (6/2)	34 (6/2)	32 (6/2)	40 (8/2)
(An) +	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	32 (6/2)	34 (6/2)	32 (6/2)	40 (8/2)
− (An)	18 (4/0)	18 (4/0)	26 (4/2)	26 (4/2)	26 (4/2)	34 (6/2)	32 (6/2)	34 (6/2)	42 (8/2)
d16(An)	24 (6/0)	24 (6/0)	32 (6/2)	32 (6/2)	32 (6/2)	40 (8/2)	42 (8/2)	40 (8/2)	48(10/2)
d8 (An. Xn)*	26 (6/0)	26 (6/0)	34 (6/2)	34 (6/2)	34 (6/2)	42 (8/2)	44 (8/2)	42 (8/2)	50(10/2)
Ads.W	24 (6/0)	24 (6/0)	32 (6/2)	32 (6/2)	32 (6/2)	40 (8/2)	42 (8/2)	40 (8/2)	48(10/2)
Ads.L	32 (8/0)	32 (8/0)	40 (7/2)	40 (8/2)	40 (8/2)	48(10/2)	50(10/2)	48(10/2)	56(12/2)
d16 (PC)	24 (6/0)	24 (6/0)	32 (6/2)	32 (6/2)	32 (6/2)	40 (8/2)	42 (8/2)	40 (8/2)	48(10/2)
d8 (PC, Xn)*	22 (6/0)	26 (6/0)	34 (6/2)	34 (6/2)	34 (6/2)	42 (8/2)	44 (8/2)	42 (8/2)	50(10/2)
#xxx	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	32 (8/2)	34 (6/2)	32 (6/2)	40 (8/2)

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

Table 7.6 Move Long Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	− (An)	d16 (An)	d8(An. Xn)*	Ads.W	Ads.L
Dn	8 (2 / 0)	8 (2 / 0)	24 (2 / 4)	24 (2 / 4)	24 (2 / 4)	32 (4 / 4)	34 (4 / 4)	32 (4 / 4)	40 (6 / 4)
An	8 (2 / 0)	8 (2 / 0)	24 (2 / 4)	24 (2 / 4)	24 (2 / 4)	32 (4 / 4)	34 (4 / 4)	32 (4 / 4)	40 (6 / 4)
(An)	24 (6 / 0)	24 (6 / 0)	40 (6 / 4)	40 (6 / 4)	40 (6 / 4)	48 (8 / 4)	50 (8 / 4)	48 (8 / 4)	56(10 / 4)
(An) +	24 (6 / 0)	24 (6 / 0)	40 (6 / 4)	40 (6 / 4)	40 (6 / 4)	48 (8 / 4)	50 (8 / 4)	48 (8 / 4)	56(10 / 4)
− (An)	26 (6 / 0)	26 (6 / 0)	42 (6 / 4)	42 (6 / 4)	42 (6 / 4)	50 (8 / 4)	52 (8 / 4)	50 (8 / 4)	58(10 / 4)
d16(An)	32 (8 / 0)	32 (8 / 0)	48 (8 / 4)	48 (8 / 4)	48 (8 / 4)	56(10 / 4)	58(10 / 4)	56(10 / 4)	64(12 / 4)
d8 (An. Xn)*	34 (8 / 0)	34 (8 / 0)	50 (8 / 4)	50 (8 / 4)	50 (8 / 4)	58(10 / 4)	60(10 / 4)	58(10 / 4)	66(12 / 4)
Ads.W	32 (8 / 0)	32 (8 / 0)	48 (8 / 4)	48 (8 / 4)	48 (8 / 4)	56(10 / 4)	58(10 / 4)	56(10 / 4)	64(12 / 4)
Ads.L	40(10 / 0)	40(10 / 0)	56(10 / 4)	56(10 / 4)	56(10 / 4)	64(12 / 4)	66(12 / 4)	64(12 / 4)	72(14 / 4)
d16 (PC)	32 (8 / 0)	32 (8 / 0)	48 (8 / 4)	48 (8 / 4)	48 (8 / 4)	56(10 / 4)	58(10 / 4)	56(10 / 4)	64(12 / 4)
d8 (PC, Xn)*	34 (8 / 0)	34 (8 / 0)	50 (8 / 4)	50 (8 / 4)	50 (8 / 4)	58(10 / 4)	60(10 / 4)	58(10 / 4)	66(12 / 4)
#xxx	24 (6 / 0)	24 (6 / 0)	40 (6 / 4)	40 (6 / 4)	40 (6 / 4)	48 (8 / 4)	50 (8 / 4)	48 (8 / 4)	56(10 / 4)

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

7.2.3 Standard Instruction Execution Times

The number of clock periods shown in Table 7.7 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 parenthesis as: (r/w).The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated. In Table 7.7 the headings have the following meanings: An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand.

Table 7.7 Standard Instruction Execuution Times

Instruction	Size	op <ea>, An	op <ea>, Dn	op Dn, <M>
ADD	Byte	–	8 (2/0)+	12 (2/1)+
	Word	12 (2/0)+	8 (2/0)+	16 (2/2)+
	Long	10 (2/0)+**	10 (2/0)+**	24 (2/4)+
SND	Byte	–	8 (2/0)+	12 (2/1)+
	Word	–	8 (2/0)+	16 (2/2)+
	Long	–	10 (2/0)+**	24 (2/4)+
CMP	Byte	–	8 (2/0)+	–
	Word	10 (2/0)+	8 (2/0)+	–
	Long	10 (2/0)+	10 (2/0)+	–
DIVS	–	–	162 (2/0)+*	–
DIVU	–	–	144 (2/0)+*	–
EOR	Byte	–	8 (2/0)+***	12 (2/1)+
	Word	–	8 (2/0)+***	16 (2/2)+
	Long	–	12 (2/0)+***	24 (2/4)+
MULS	–	–	74 (2/0)+	–
MULU	–	–	74 (2/0)+*	–
ADDI	Byte	–	8 (2/0)+	12 (2/1)+
	Word	–	8 (2/0)+	16 (2/2)+
	Long	–	10 (2/0)+**	24 (2/4)+
ADDI	Byte	–	8 (2/0)+	12 (2/1)+
	Word	12 (2/0)+	8 (2/0)+	16 (2/2)+
	Long	10 (2/0)+**	10 (2/0)+**	24 (2/4)+

Notes :

+	Add effective address calculation time
*	Indicates maximum value
**	The base time of 10 clock periods is increased to 12 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 TMP68008 provides less than 10% difference between the best and worst case timings.
MULS, MULU	- The multiply algorithm requires 42+2n clocks where n is defined as: MULS: n = tag the <ea> with a zero as the MSB; 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. MULU: n = the number of ones in the <ea>

7.2.4 Immediate Instruction Execution Times

The number of clock periods shown in Table 7.8 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 7.8, the headings have the following meanings

= immediate operand

Dn = data register operand
An = address register operand
M = memory operand

Table 7.8 Immediate Instruction Clock Periods

Instruction	Size	op #, Dn	op #, An	op #, An
ADDI	Byte Word Long	16 (4/0) 16 (4/0) 28 (6/0)	— — —	20 (4/1)+ 24 (4/2)+ 40 (6/4)+
ADDQ	Byte Word Long	8 (2/0) 8 (2/0) 12 (2/0)	— 12 (2/0) 12 (2/0)	12 (2/1)+ 16 (2/2)+ 24 (2/4)+
ANDI	Byte Word Long	16 (4/0) 16 (4/0) 28 (6/0)	— — —	20 (4/1)+ 24 (4/2)+ 40 (6/4)+
CMPI	Byte Word Long	16 (4/0) 16 (4/0) 26 (6/0)	— — —	16 (4/0)+ 16 (4/0)+ 24 (6/0)+
EORI	Byte Word Long	16 (4/0) 16 (4/0) 28 (6/0)	— — —	20 (4/1)+ 24 (4/2)+ 40 (6/4)+
MOVEQ	Long	8 (2/0)	—	—
ORI	Byte Word Long	16 (4/0) 16 (4/0) 28 (6/0)	— — —	20 (4/1)+ 24 (4/2)+ 40 (6/4)+
SUBI	Byte Word Long	16 (4/0) 16 (4/0) 28 (6/0)	— — —	12 (2/1)+ 16 (2/2)+ 24 (2/4)+
SUBQ	Byte Word Long	8 (2/0) 8 (2/0) 12 (2/0)	— 12 (2/0) 12 (2/0)	20 (4/1)+ 24 (4/2)+ 40 (6/4)+

+ add effective address calculation time.

7.2.5 Single Operand Instruction Execution Times

Table 7.9 indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7.9 Single Operand Instruction Execution Times

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

+ add effective address calculation time.

7.2.6 Shift/Rotate Instruction Execution Times

Table 7.10 indicates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7.10 Shift/Rotate Instruction Clock Periods

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

+ add effective address calculation time
n is the shift count

7.2.7 Bit Manipulation Instruction Execution Times

Table 7.11 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7.11 Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	—	12 (2 / 1) +	—	20 (4 / 1) +
	Long	12 (2 / 0)*	—	20 (4 / 0)*	—
BCLR	Byte	—	12 (2 / 1) +	—	20 (4 / 1) +
	Long	14 (2 / 0)*	—	22 (4 / 0)*	—
BSET	Byte	—	12 (2 / 1) +	—	20 (4 / 1) +
	Long	12 (2 / 0)*	—	20 (4 / 0)*	—
BTST	Byte	—	8 (2 / 0) +	—	16 (4 / 0) +
	Long	10 (2 / 0)	—	18 (4 / 0)	—

+ add effective address calculation time
* indicates maximum value

7.2.8 Conditional Instruction Execution Times

Table 7.12 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as: (r/w).The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 7.12 Conditional Instruction Execution Times

Instruction	Displacement	Trap Branch Taken	Trap Branch Not Taken
Bcc	Byte	18 (4 / 0)	12 (2 / 0)
	Word	18 (4 / 0)	20 (4 / 0)
BRA	Byte	18 (4 / 0)	—
	Word	18 (4 / 0)	—
BSR	Byte	34 (4 / 4)	—
	Word	34 (4 / 4)	—
DBcc	CC True	—	20 (4 / 0)
	CC False	18 (4 / 0)	26 (6 / 0)
CHK	—	68 (8 / 6) + *	14 (2 / 0) +
TRAP	—	62 (8 / 6)	—
TRAPV	—	66 (10 / 6)	8 (2 / 0)

+ add effective address calculation time
* indicates maximum value

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

Table 7.13 indicates the number of clock periods required for the jump, jump-to-subroutine, load effective address, push effective address, and move multiple registers instructions.The number of bus read and write cycles is shown in parenthesis as: (r/w).

Table 7.13 JMP,JSR,LEA,PEA,and MOVEM Instruction Execution Times

Instruction	Size	(An)	(An) +	– (An)	d16(An)	d8(An, Xn)*	Ads. W	Ads. L	d16(PC)	d8(PC, Xn)*
JMP	–	16 (4/0)	–	–	18 (4/0)	22 (4/0)	18 (4/0)	24 (6/0)	18 (4/0)	22 (4/0)
JSR	–	32 (4/4)	–	–	34 (4/4)	38 (4/4)	34 (4/4)	40 (6/4)	34 (4/4)	38 (4/4)
LEA	–	8 (2/0)	–	–	16 (4/0)	20 (4/0)	16 (4/0)	24 (6/0)	16 (4/0)	20 (4/0)
PEA	–	24 (2/4)	–	–	32 (4/4)	36 (4/4)	32 (4/4)	40 (6/4)	32 (4/4)	36 (4/4)
MOVEM M→R	Word	24 + 8n (6 + 2n / 0)	24 + 8n (6 + 2n / 0)	– –	32 + 8n (8 + 2n / 0)	34 + 8n (8 + 2n / 0)	32 + 8n (10 + n / 0)	40 + 8n (10 + 2n / 0)	32 + 8n (8 + 2n / 0)	34 + 8n (8 + 2n / 0)
	Long	24 + 16n (6 + 4n / 0)	24 + 16n (6 + 4n / 0)	– –	32 + 16n (8 + 4n / 0)	32 + 16n (8 + 4n / 0)	32 + 16n (8 + 4n / 0)	40 + 16n (8 + 4n / 0)	32 + 16n (8 + 4n / 0)	34 + 16n (8 + 4n / 0)
MOVEM R→M	Word	16 + 8n (4 / 2n)	– –	16 + 8n (4 / 2n)	24 + 8n (6 / 2n)	26 + 8n (6 / 2n)	24 + 8n (6 / 2n)	32 + 8n (8 / 2n)		
	Long	16 + 16n (4 / 4n)	– –	16 + 16n (4 / 4n)	24 + 16n (6 / 4n)	26 + 16n	24 + 16n (8 / 4n)	32 + 16n (6 / 4n)		

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

7.2.10 Multi-Precision Instruction Execution Times

Table 7.14 indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The number of read and write cycles is shown in parenthesis as: (r/w). In Table 7.14, the headings have the following meanings.

Dn = data register operand
M = memory operand

Table 7.14 Multi-Precision Instruction Execution Times

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte	8 (2/0)	22 (4/1)
	Word	8 (2/0)	50 (6/2)
	Long	12 (2/0)	58 (10/4)
CMPM	Byte	–	16 (4/0)
	Word	–	24 (6/0)
	Long	–	40 (10/0)
SUBX	Byte	8 (2/0)	22 (4/1)
	Word	8 (2/0)	50 (6/2)
	Long	12 (2/0)	58 (10/4)
ABCD	Byte	10 (0/2)	20 (4/1)
SBCD	Byte	10 (0/2)	20 (4/1)

7.2.11 Miscellaneous Instruction Execution Times

Table 7.15 and 7.16 indicate the number of clock periods for the following miscellaneous instructions.The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

Table 7.15 Miscellaneous Instruction Execution Times

Instruction	Register	Memory
ANDI to CCR	32 (6/0)	–
ANDI to SR	32 (6/0)	–
EORI to CCR	32 (6/0)	–
EORI to SR	32 (6/0)	–
EXG	10 (2/0)	–
EXT	8 (2/0)	–
LINK	32 (4/4)	–
MOVE to CCR	18 (4/0)	18 (4/0) +
MOVE to SR	18 (4/0)	18 (4/0) +
MOVE from SR	10 (2/0)	16 (2/2) +
MOVE to USP	8 (2/0)	–
MOVE from USP	8 (2/0)	–
NOP	8 (2/0)	–
ORI to CCR	32 (6/0)	–
ORI to SR	32 (6/0)	–
RESET	136 (2/0)	–
RTE	40 (10/0)	–
RTR	40 (10/0)	–
RTS	32 (8/0)	–
STOP	4 (0/0)	–
SWAP	8 (2/0)	–
UNLK	24 (6/0)	–

+ add effective address calculation time

Table 7.16 Move Peripheral Instruction Execution Times

Instruction	Size	Register→Memory	Memory→Register
MOVEP	Word	24 (4/2)	24 (6/0)
	Long	32 (4/4)	32 (8/0)

+ add effective address calculation time

7.2.12 Exception Processing Execution Times

Table 7.17 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 instruction of the handler routine. The number of bus read and write cycles is shown in parenthesis as: (r/w).

Table 7.17 Exception Processing Execution Times

Exception	Periods
Address Error	94 (8/14)
Bus Error	94 (8/14)
CHK Instruction	68 (8/6) +
Interrupt	72 (9/16)*
Illegal Instruction	62 (8/6)
Privileged Instruction	62 (8/6)
Trace	62 (8/6)
TRAP Instruction	62 (8/6)
TRAPV Instruction	66 (10/6) +
Divide by Zero	66 (8/6) +
$\overline{\text{RESET}}$**	64 (12/0)

+ add effective address calculation time

* The interrupt acknowledge bus cycle is assumed to take four external clock periods

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

8. ELECTRICAL SPECIFICATIONS

This section contains the electrical specifications and associated timing information for the TMP68008.

8.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	Vcc	−0.3~+7.0	V
Input Voltage	Vin	−0.3~+7.0	V
Operating Temperature Range	Ta	0~+70	℃
Storage Temperature	Tstg	−55~+150	℃

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 voltages higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced unused inputs are tied to an appropriate logic voltage level (e. g., either ground or Vcc)

8.2 DC ELECTRICAL CHARACTERISITICS

(Vcc = 5.0 V ±5%, GND = 0 V, Ta = 0 ~ 70℃)

Characteristic		Symbol	Min	Max	Unit
Input High Voltage		VIH	2.0	Vcc	V
Input Low Voltage		VIL	GND-0.3	0.8	V
Input Leakage Current (5.25V)	$\overline{\text{BERR}}$, $\overline{\text{BR}}$, $\overline{\text{DTACK}}$, CLK, $\overline{\text{IPL0/2}}$, $\overline{\text{VPA}}$	Iin	–	2.5	μA
	$\overline{\text{HALT}}$, $\overline{\text{RESET}}$		–	20	
Hi-Z (Off State) Input Current (2.4V/0.4V)	$\overline{\text{AS}}$, A0~A19, D0~D7, FC0~FC2, $\overline{\text{DS}}$, R/$\overline{\text{W}}$	ITSI	–	20	μA
Output High Voltage (IOH = -400 μA)	E*	VOH	Vcc-0.75	–	V
	E, $\overline{\text{AS}}$, A0~A19, $\overline{\text{BG}}$, D0~D7, FC0~FC2, $\overline{\text{DS}}$, R/$\overline{\text{W}}$		2.4	–	
Output Low Voltage (IOL = 1.6mA)	$\overline{\text{HALT}}$	VOL	–	0.5	V
(IOL = 3.2mA)	A0~A19, $\overline{\text{BG}}$, FC0~FC2		–	0.5	
(IOL = 5.0mA)	$\overline{\text{RESET}}$		–	0.5	
(IOL = 5.3mA)	E, $\overline{\text{AS}}$, D0~D7, $\overline{\text{DS}}$, R/$\overline{\text{W}}$		–	0.5	
Power Dissipation,	*Ta = 0℃	PD	–	1.5	W
Capacitance (Vin = 0V, Ta = 25℃ : Frequency = 1MHz)**		Cin	–	20.0	pF
Load Capacitance	$\overline{\text{HALT}}$	CL	–	70	pF
	All Others		–	130	

* : With external pullup resistor of 1.1kΩ.
** : Capacitance is periodically sampled rather than 100% tested.
*** : During normal operation instantaneous Vcc current requirements may be as high as 1.5A.

8.3 AC ELECTRICAL SPECIFICATION-CLOCK TIMMING (See Figure 8.1)

Characteristic	Symbol	8MHz		10MHz		Unit
		Min	Max	Min	Max	
Frequency of Operation	f	2.0	8.0	2.0	10.0	MHz
Cycle Time	tcyc	125	500	100	500	ns
Clock Pulse Width	tCL tCH	55 55	250 250	45 45	250 250	ns
Rise and Fall Times	tCr tCf	— —	10 10	— —	10 10	ns

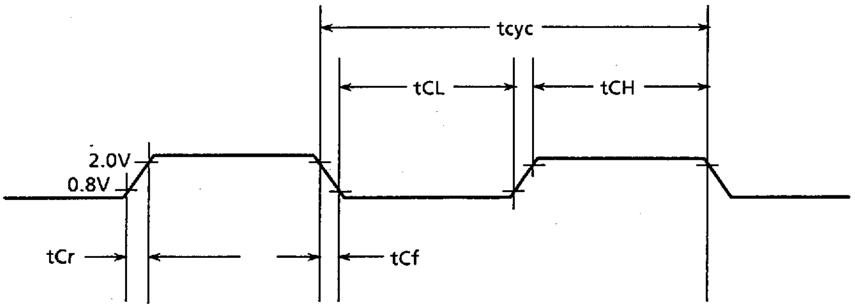


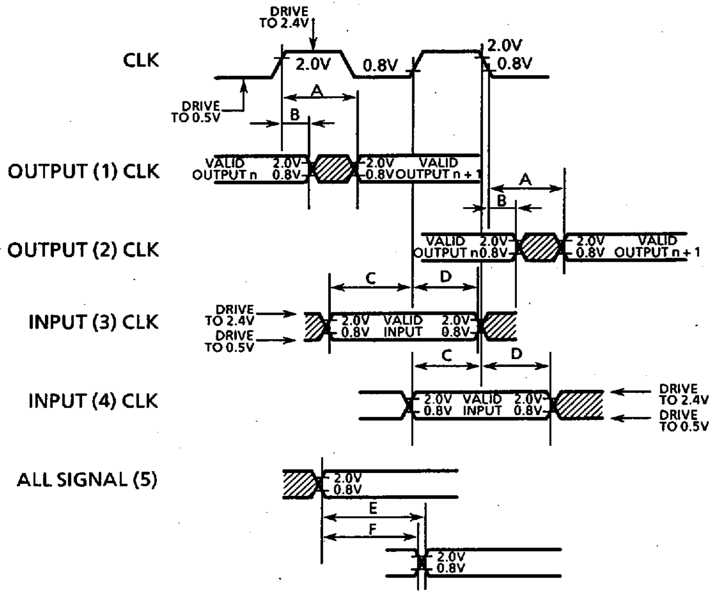
Figure 8.1 Input Clock Waveform

8.4 AC ELECTRICAL SPECIFICATION DEFINITIONS

The AC specifications presented consist of output delays, input setup and hold times, and signal skew times. All signals are specified relative to an appropriate edge of the clolk and possibly to one or more other signals.

The measurement of the AC specifications is defined by the waveforms shown in Figure 8.2. In order to test the parameters guaranteed by TOSHIBA, inputs must be driven to the voltage levels specified in this figure. Outputs are specified with minimum and / or maximum limits, as appropriate, and are measured as shown in Figure 8.2. Inputs are specified with minimum setup and hold times, and are measured as shown. Finaly, the measurement for signal-to-signal specifications are also shown.

Note : The testing levels used to verify conformance to the AC specifications does not affect the guaranteed DC operation of the device as specified in the DC electrical character-istics.



Notes :

1 This output timing is applicable to all parameters specified relative to the rising edge of the clock.
2 This output timing is applicable to all parameters specified relative to the falling edge of the clock.
3 This input timing is applicable to all parameters specified relative to the rising edge of the clock.
4 This input timing is applicable to all parameters specified relative to the falling edge of the clock.
5 This timing is applicable to all parameters specified relative to the assertion / negation of another signal.

Legend :

A	Maximum output delay specification.
B	Minimum output hold time.
C	Minimum input setup time specification.
D	Minimum input hold time specification.
E	Signal valid to signal valid specification (maximum or minimum) .
F	Signal valid to signal invalid specification (maximum to minimum) .

Figure 8.2 Drive Levels and Test Points for AC Specifications

8.5 AC ELECTRICAL SPECIFICATIONS - READ AND WRITE CYCLES (1 / 3)

(Vcc = 5.0V ± 5%, GND = 0V, Ta = 0 ~ 70°C; see Figures 8.3 and 8.4)

Num.	Characteristic	Symbol	8MHz		10MHz		Unit
			Min	Max	Min	Max	
1	Clock Period	tcyc	125	500	100	500	ns
2	Clock Width Low	tCL	55	250	45	250	ns
3	Clock Width High	tCH	55	250	45	250	ns
4	Clock Fall Time	tCf	—	10	—	10	ns
5	Clock Rise Time	tCr	—	10	—	10	ns
6	Clock Low to Address Valid	tCLAV	—	62	—	50	ns
6A	Clock High to FC Valid	tCHFCV	—	62	—	50	ns
7	Clock High to Address, Data Bus High Impedance (Maximum)	tCHADZ	—	80	—	70	ns
8	Clock High to Address, FC Invalid (Minimum)	tCHAFI	0	—	0	—	ns
9 [1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low	tCHSL	3	60	3	55	ns
11 [2]	Address Valid to $\overline{AS}$, $\overline{DS}$ Low (Read) / $\overline{AS}$ Low (Write)	tAVSL	30	—	20	—	ns
11A [2]	FC Valid to $\overline{AS}$, $\overline{DS}$ Low (Read) / $\overline{AS}$ Low (Write)	tFCVSL	90	—	70	—	ns
12 [1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	tCLSH	—	62	—	50	ns
13 [2]	$\overline{AS}$, $\overline{DS}$ High to Address / FC Invalid	tSHARI	40	—	30	—	ns
14 [2]	$\overline{AS}$, $\overline{DS}$ Width Low (Read) / $\overline{AS}$ Low (Write)	tSL	270	—	195	—	ns
14A [2]	$\overline{DS}$ Width Low (write)	tDSL	140	—	95	—	ns
15 [2]	$\overline{AD}$, $\overline{DS}$ Width High	tSH	150	—	105	—	ns
16	Clock High to Control Bus High Impedance	tCHCZ	—	80	—	70	ns
17 [2]	$\overline{AS}$, $\overline{DS}$ High to R / $\overline{W}$ High (Read)	tSHRH	40	—	30	—	ns
18 [1]	Clock High to R / $\overline{W}$ High	tCHRH	0	55	0	45	ns
20 [1]	Clock High to R / $\overline{W}$ Low	tCHRL	0	55	0	45	ns
20A [2,6]	$\overline{AS}$ Low to R / $\overline{W}$ Valid(Write)	tASRV	—	10	—	10	ns
21 [2]	Address Valid to R / $\overline{W}$ Low (Write)	tAVRL	20	—	0	—	ns
21A [2]	FC Valid to R / $\overline{W}$ Low (Write)	tFCVRL	60	—	50	—	ns
22 [2]	R / $\overline{W}$ Low to $\overline{DS}$ Low (Write)	tRLSL	80	—	50	—	ns
23	Clock Low to Data Out Valid (Write)	tCLDO	—	62	—	50	ns

8.5 AC ELECTRICAL SPECIFICATIONS - READ AND WRITE CYCLES(2 / 3)

Num.	Characteristic	Symbol	8MHz		10MHz		Unit
			Min	Max	Min	Max	
25[2]	$\overline{AS}$, $\overline{DS}$ High to Data Out Invalid (Write)	tSHDOI	50	–	30	–	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	tDOSL	40	–	30	–	ns
27[5]	Data In to Clock Low (Setup Time on Read)	tDICL	10	–	10	–	ns
28[2]	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High	tSHDAH	0	245	0	190	ns
29	$\overline{AS}$, $\overline{DS}$ High to Data In Invalid (Hold Time on Read)	tSHDII	0	–	0	–	ns
29A	$\overline{AS}$, $\overline{DS}$ Negated to Data In High Impedance	tSHDZ	–	187	–	150	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	tSHBEH	0	–	0	–	ns
31[2,5]	$\overline{DTACK}$ Low to Data Valid (Asynchronous Setup Time on Read)	tDALDI	–	90	–	65	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	tRHr, f	0	200	0	200	ns
33	Clock High to $\overline{BG}$ Low	tCHGL	–	62	–	50	ns
34	CLock High to $\overline{BG}$ High	tCHGH	–	62	–	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	tBRLGL	1.5	3.5	1.5	3.5	Clk. Per.
36[7]	$\overline{BR}$ High to $\overline{BG}$ High	tBRHGH	1.5	3.5	1.5	3.5	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High (52-Pin Version Only)	tGALGH	1.5	3.5	1.5	3.5	Clk. Per.
37A[8]	$\overline{BGACK}$ Low to $\overline{BR}$ High (52-Pin Version Only)	tGALBRH	20	1.5 Clocks	20	1.5 Clocks	ns
38	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High)	tGLZ	–	80	–	70	ns
39	$\overline{BG}$ Width High	tGH	1.5	–	1.5	–	Clk. Per.
41	Clock Low to E Transition	tCLET	–	50	–	45	ns
42	E Output Rise and Fall Time	tEr, f	–	15	–	15	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	tSHVPH	0	120	0	90	ns
45	E Low to Control, Address Bus Invalid (Address Hold Time)	tELCAI	30	–	10	–	ns
46	$\overline{BGACK}$ Width Low (52-Pin Version Only)	tGAL	1.5	–	1.5	–	Clk. Per.
47[5]	Asynchronous Input Setup Time	tASI	10	–	10	–	ns

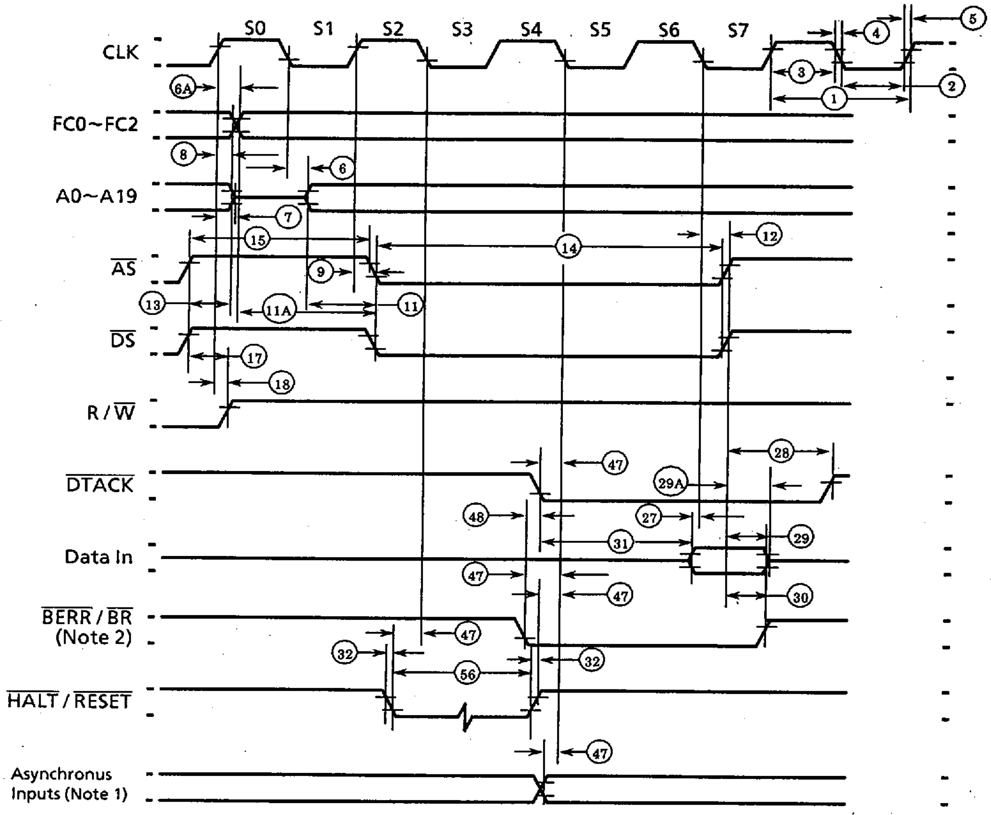
8.5 AC ELECTRICAL SPECIFICATIONS - READ AND WRITE CYCLES (3 / 3)

Num.	Characteristic	Symbol	8MHz		10MHz		Unit
			Min	Max	Min	Max	
48[2,3]	$\overline{BERR}$ Low to $\overline{DTACK}$ Low	tBELDAL	20	–	20	–	ns
49[9]	$\overline{AS}$, $\overline{DS}$ High to E Low	tSHEL	-70	70	-55	55	ns
50	E Width HIGH	tEH	450	–	350	–	ns
51	E Width Low	tEL	700	–	550	–	ns
53	Clock High to Data Out Invalid	tCLDOI	0	–	0	–	ns
54	E Low to Data Out Invalid	tELDOI	30	–	20	–	ns
55	R / $\overline{W}$ to Data Bus Impedance Driven	tRLDBD	30	–	20	–	ns
56[4]	$\overline{HALT}$ / $\overline{RESET}$ Pulse Width	tHRPW	10	–	10	–	Clk. Per.
57	$\overline{BGACK}$ High to Control Bus Driven (52-Pin Version Only)	tGABD	1.5	–	1.5	–	Clk. Per.
57A	$\overline{BGACK}$ Negated to FC, $\overline{VMA}$ Driven	tGAFD	1	–	1	–	Clks
58[7]	$\overline{BG}$ High to Control Bus Driven	tGHBD	1.5	–	1.5	–	Clk. Per.
58A[7]	$\overline{BR}$ Negated to FC, $\overline{VMA}$ Driven	tRHFD	1	–	1	–	Clks

Notes :

1. For a loading capacitance of less than or equal to 50 picofarads, subtract 5 nanoseconds from the values given in these columns.
2. Actual value depends on clock period.
3. If #47 is satisfied for both $\overline{DTACK}$ and $\overline{BERR}$, #48 may by 0 nanoseconds.
4. For power up the MPU must be held in $\overline{RESET}$ state for 100 milliseconds to allow stabilization of on-chip circuitry. After the system is powered up, #56 refers to the minimum pulse width required to reset the system.
5. If the asynchronous setup time (#47) requirements are satisfied, the $\overline{DTACK}$ low-to-data setup time (#31) requirement can ignored. The data must only satisfy the data-in to clock-low setup time (#27) for the following cycle.
6. When $\overline{AS}$ and R / $\overline{W}$ are equally loaded ($\pm$20%), subtract 10 nanoseconds from the values in these columns.
7. The processor will negate $\overline{BG}$ and driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.
8. The minimum value must be met to guarantee operation. If the maximum value is exceeded, $\overline{BG}$ may by reasserted.
9. The falling edge of S6 triggers both the negation of the strobes ($\overline{AS}$ and $\overline{DS}$) and the falling edge of E. Either of these events can occur first, depending upon the loading on each signal. Specification #49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and the falling edge of the E clock.

These waveforms should only 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.

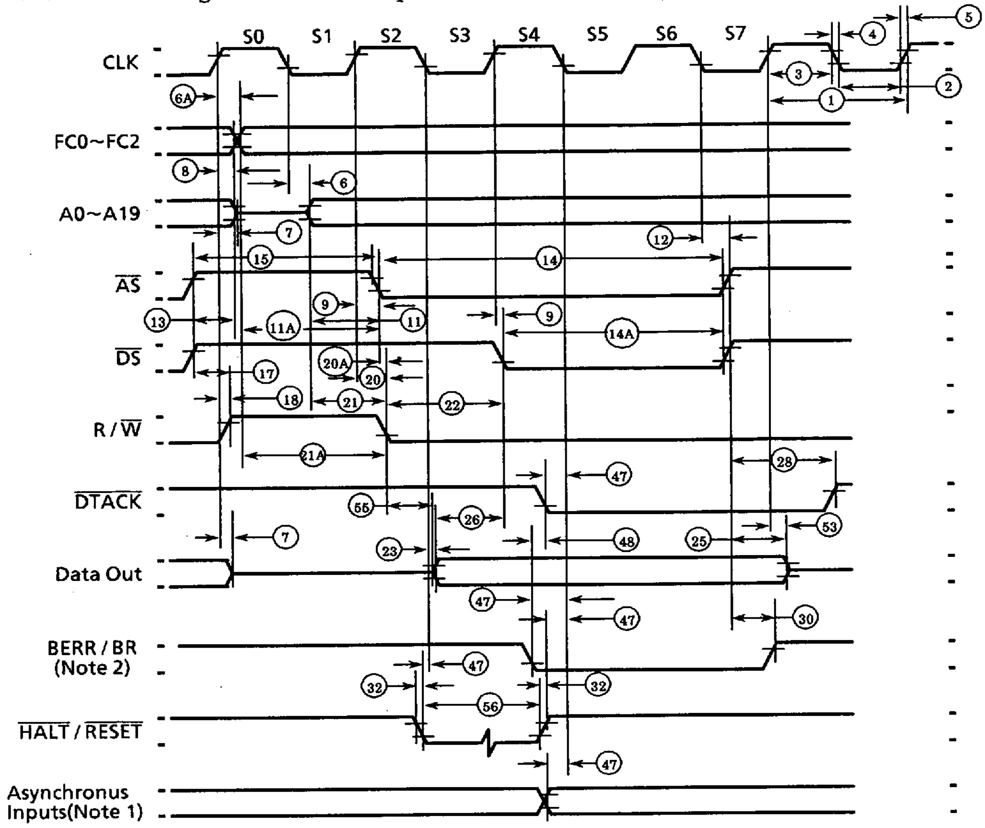


Notes:

1. Setup time for the asynchronus inputs $\overline{\text{IPL0/2}}$, $\overline{\text{IPL1}}$, and $\overline{\text{VPA}}$ guarantees their recognition at the falling edge of the clock.
2. $\overline{\text{BR}}$ need fall at this time only in order to insure being recognized at the end of this bus cycle.
3. Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage of 2.0 volts, unless otherwise noted.

Figure 8.3 Read Cycle Timing Diagram

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



Notes :

1. Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage of 2.0 volts, unless otherwise noted.
2. Because of loading variations, R / $\overline{W}$ may be valid after $\overline{AS}$ even though are initiated by the rising edge of S2 (Specification 20A)

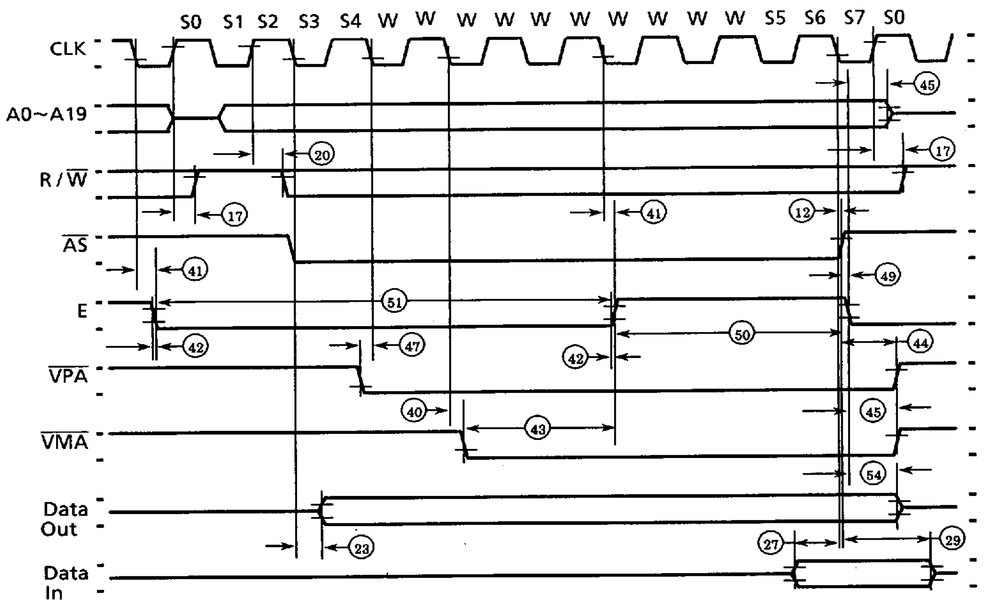
Figure 8.4 Write Cycle Timing Diagram

8.6 AC ELECTRICAL SPECIFICATIONS - TMP68008 TO 6800 PERIPHERAL

(Vcc = 5.0V ±5%; GND = 0V, Ta = 0 ~ 70℃; see Figures 8.5 and 8.6)

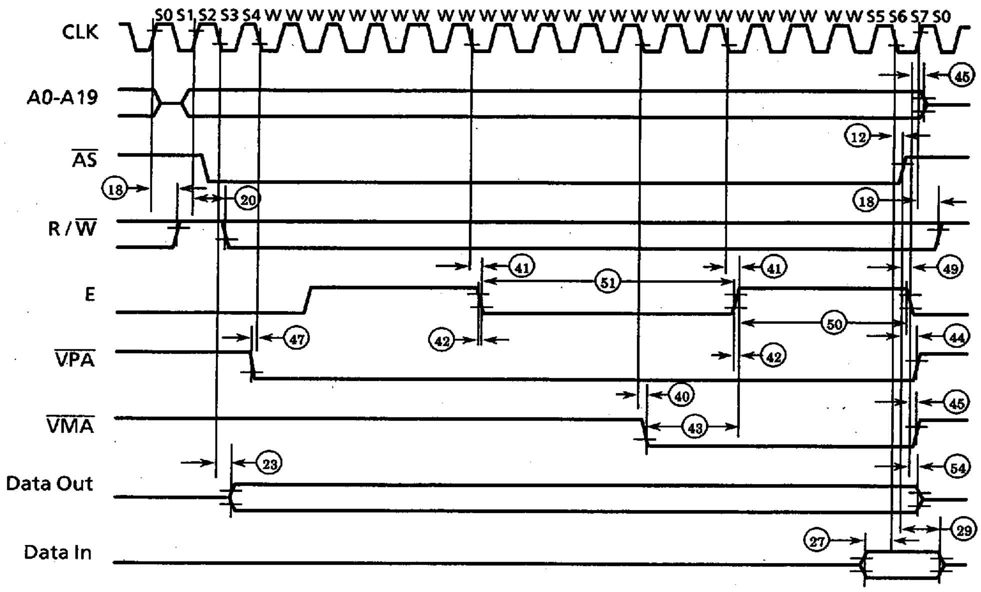
Num.	Characteristic	Symbol	8MHz		10MHz		Unit
			Min	Max	Min	Max	
12 [1]	Clock Low to $\overline{AS}$, $\overline{DS}$ Nagated	tCLSH	–	62	–	50	ns
18 [1]	Clock High to R/$\overline{W}$ High (Read)	tCHRH	0	55	0	45	ns
20 [1]	Clock High to R/$\overline{W}$ Low (Write)	tCHRL	0	55	0	45	ns
23	Clock Low to Data Out Valid (Write)	tCLDO	–	62	–	50	ns
27	Data In to Clock Low (Setup Time on Read)	tDICL	10	–	10	–	ns
29	$\overline{AS}$, $\overline{DS}$ Negated to Data-In Invalid (Hold Time on Read)	tSHDII	0	–	0	–	ns
41	Clock Low to E Transition	tCLET	–	55	–	45	ns
42	E Output Rise and Fall Time	$tE_{r,f}$	–	15	–	15	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	tSHVPH	0	120	0	90	ns
45	E Low to Control, Address Bus Invalid (Address Hold Time)	tELCAI	30	–	10	–	ns
47	Asynchronous Input Setup Time	tASI	10	–	10	–	ns
49 [2]	$\overline{AS}$, $\overline{DS}$ High to E Low	tSHEL	-70	70	-65	55	ns
50	E Width High	tEH	450	–	350	–	ns
51	E Width Low	tEL	700	–	550	–	ns
54	E Low to Data Out Invalid	tELDOI	30	–	20	–	ns

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



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

Figure 8.5 TMP68008 to 6800 Peripheral Timing Diagram -Best Case


CLK
S0 S1 S2 S3 S4 W W W W W W W W W W W W W W W W W W W W W W W W W W W W W W W S5 S6 S7 S0
A0-A19
AS
R/W
E
VPA
VMA
Data Out
Data In
12
18
20
23
27
29
40
41
42
43
44
45
47
49
50
51
54

Note : This timing diagram is included for those who wish to desigh their own circuit to generate VMA. lt shows the worst case possibly attainable.

Figure 8.6 TMP68008 to 6800 Peripheral Timing Diagram -Worst Case

8.7 AC ELECTRICAL SPECIFICATIONS - BUS ARBITRATION

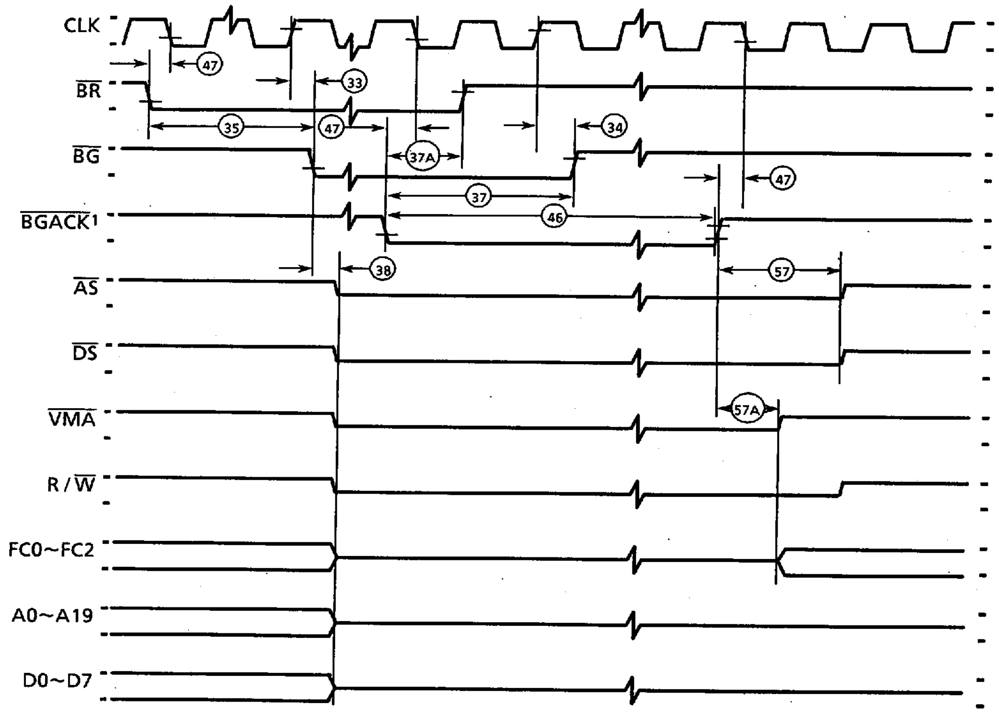
(Vcc = 5.0V ± 5%; GND = 0V, Ta = 0 ~ 70°C; see Figures 8.7, 8.8, and 8.9)

Num.	Characteristic	Symbol	8MHz		10MHz		Unit
			Min	Max	Min	Max	
7	Clock High to Address, Data Bus High Impedance	tCHADZ	—	80	—	70	ns
16	Clock High to Control Bus High Impedance	tCHCZ	—	80	—	70	ns
33	Clock High to $\overline{BG}$ Low	tCHGL	—	62	—	50	ns
34	Clock High to $\overline{BG}$ High	tCHGH	—	62	—	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	tBRLGL	1.5	3.5	1.5	3.5	Clk. Per
36[1]	$\overline{BR}$ High to $\overline{BG}$ High	tBKHGH	1.5	3.5	1.5	3.5	Clk. Per
37	$\overline{BGACK}$ Low to $\overline{BG}$ High (52-Pin Version Only)	tGALGH	1.5	3.5	1.5	3.5	Clk. Per
37A[2]	$\overline{BGACK}$ Low to BR High (52-Pin Version Only)	tGALBRH	20	1.5 Clocks	20	1.5 Clocks	ns
38	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High)	tGLZ	—	80	—	70	ns
39	$\overline{BG}$ Width High	tGH	1.5	—	1.5	—	Clk. Per
46	$\overline{BGACK}$ Width Low (52-Pin Version Only)	tGAL	1.5	—	1.5	—	Clk. Per
47	Asynchronous Input Setup Time	tASI	10	—	10	—	ns
57	$\overline{BGACK}$ High to Control Bus Driven (52-Pin Version Only)	tGABD	1.5	—	1.5	—	Clk. Per
57A	$\overline{BGACK}$ Negated to FC, $\overline{VMA}$ Driven (52-Pin Version Only)	tGAFD	1	—	1	—	Clk. Per
58[1]	$\overline{BG}$ High to Control Bus Driven	tGHBD	1.5	—	1.5	—	Clk. Per
58A[1]	$\overline{BR}$ Negated to FC, $\overline{VMA}$ Driven	tRHFD	1	—	1	—	Clk. Per

Notes :

1. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.
2. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, BG may be reasserted.

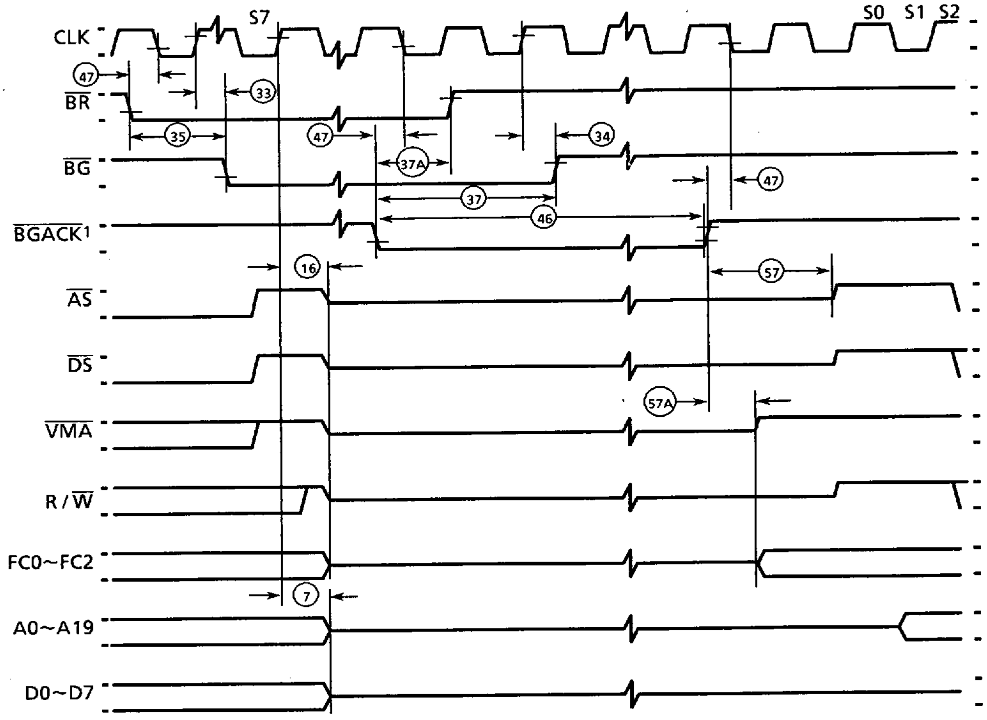
These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



Note:
1. 52-Pin Version of TMP68008 Only.

Figure 8.7 Bus Arbitration Timing - Idle Bus Case

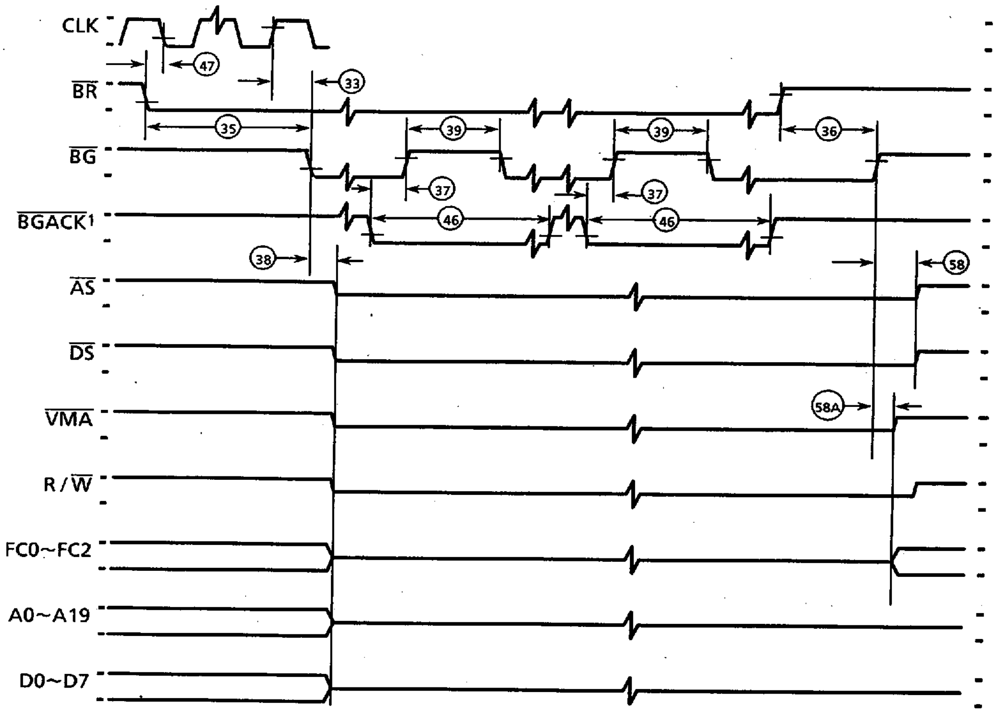
These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



Note :
1. 52-Pin Version of TMP68008 Only.

Figure 8.8 Bus Arbitration Timing - Active Bus Case

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



Note :
1. 52-Pin Version of TMP68008 Only.

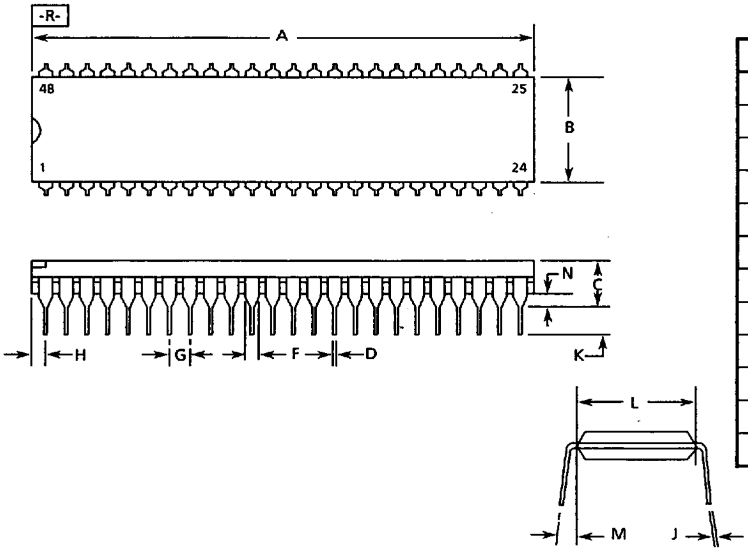
Figure 8.9 Bus Arbitration Timing - Multiple Bus Requests (52-Pin Version Only)

9. MECHANICAL DATA

This section contains package dimensions for the TMP68008.

9.1 PACKAGE DIMENSIONS

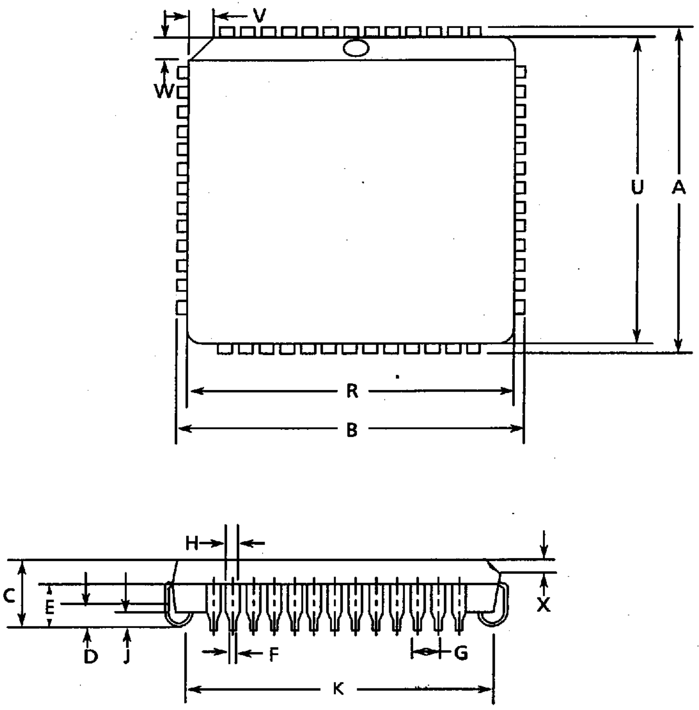
PLASTIC PACKAGE



Unit : mm

DIM	MIN	MAX
A	61.34	62.10
B	13.72	14.22
C	3.94	5.08
D	0.36	0.55
F	1.02	1.52
G	2.54BSC	
H	1.79BSC	
J	0.20	0.38
K	2.92	3.42
L	15.24BSC	
M	0°	15°
N	0.51	1.01

PLCC PACKAGE



Unit : mm

DIM	MIN	MAX
A	19.94	20.19
B	19.94	20.19
C	4.19	4.57
D	0.64	1.01
E	2.16	2.79
F	0.33	0.53
G	1.27BSC	
H	0.66	0.81
J	0.33	0.63
K	17.52	18.54
R	19.05	19.20
U	19.05	19.20
V	1.07	1.21
W	1.07	1.21
X	1.07	1.42
Y	0.00	0.50

Note :
52-Pin PLCC is under development.