16- BIT MICROPROCESSOR

TMP68HC000P-10 / TMP68HC000P-12 / TMP68HC000P-16
TMP68HC000N-10 / TMP68HC00N-12 / TMP68HC000N-16
TMP68HC000Y-10 / TMP68HC000Y-12 / TMP68HC000Y-16
TMP68HC000F-10 / TMP68HC000F-12 / TMP68HC000F-16
TMP68HC000T-10* / TMP68HC000T-12* / TMP68HC000T*-16

Package type :

P : plastic DIP
N : Shrank plastic DIP
Y : pin grid array (without stand-off) : TMP68HC000 only
F : plastic QFP : TMP68HC000 only
T : plastic leaded chi carrier

(* Under development)

# 1. INTRODUCTION

TMP68HC000 are compatible with the Motorola MC68HC000.

- Low power Dissipation (TMP68HC000)

As show in the user programming model (Figure 1.1) , the TMP68HC000 offers 16/32-bit registers and a 32-bit program counter. The first eight registers (D0~D7) are used as data registers for byte (8-bit) , word (16-bit) , and long word (32-bit) operations. The second set of seven registers (A0~A6) and the user stack pointer (USP) may be used as software stack pointers and base address registers. In addition, the registers may be used for word and long word operations. All of the 16 registers may be used as index registers.

In supervisor mode, the upper byte of the status register and the supervisor stack pointer (SSP) are also available to the programmer. These registers are shown in Figure 1.2.

The status register (Figure 1.3) contains the interrupt mask (eight levels available) as well as the condition codes: extend (X) , negative (N) , zero (Z) , overflow (V) , and carry (C) . Additional status bits indicate that the processor is in a trace (T) mode and in a supervisor (S) or user state.

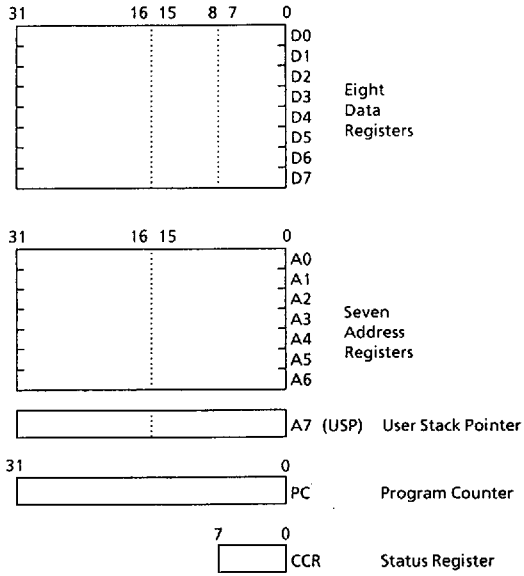



Figure 1.1 User Programming Model

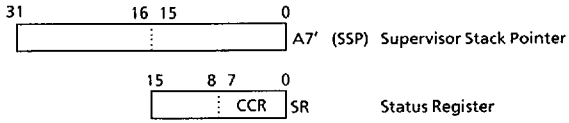



Figure 1.2 Supervisor Programming Model Supplement

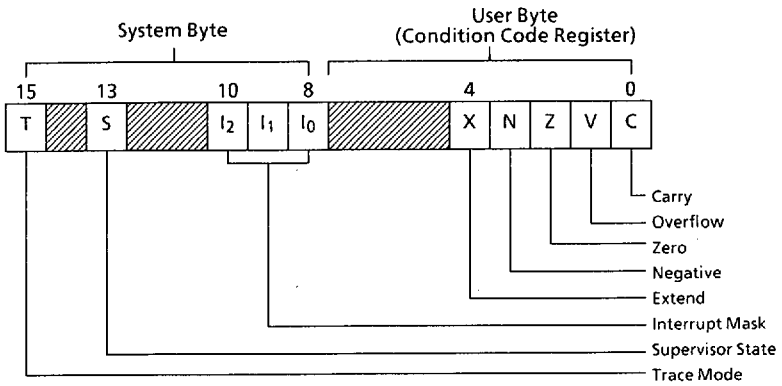



Figure 1.3 Status Register

## 1.1 DATA TYPES AND ADDRESSING MODES

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

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

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

The 14 address modes, shown in Table 1.1, include six basic types:

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

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

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 / SSP / 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
( ) = Effective Address
d8 = 8-Bit Offset (Displacement)
d16 = 16-Bit Offset (Displacement)
#xxx = Immediate Data

## 1.2 INSTRUCTION SET OVERVIEW

The TMP68HC000 instruction set is shown in Table 1.2. Some additional instructions are variations, or subsets, of these and they appear in Table 1.3. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned, multiply and divide, "quick"arithmetic operations, BCD arithmetic, and expanded operations (through traps).

Table 1.2 Instruction Set Summary (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 Summary (2/2)

| Mnemonic | Description |
|---|---|
| MOVE | Move |
| MOVEM | Move Multiple Registers |
| 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 |
| 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 |
| | MOVEQ | Move Quick |
| | MOVE from SR | Move from Status Register |
| | MOVE to SR | Move to Status Register |
| | MOVE to CCR | Move to Condition Codes |
| | MOVE USP | Move User Stack Pointer |
| NEG | NEG | Negate |
| | NEGX | Negate with Extend |
| OR | OR | Logical OR |
| | ORI | OR Immediate |
| | ORI to CCR | OR Immediate to Condition Codes |
| | ORI to SR | OR Immediate to Status Register |
| SUB | SUB | Subtract |
| | SUBA | Subtract Address |
| | SUBI | Subtract Immediate |
| | SUBQ | Subtract Quick |
| | SUBX | Subtract with Extend |

# 2. DATA ORGANIZATION AND ADDRESSING CAPABILITIES

This section contains a description of the registers and the data organization of the TMP68HC000.

## 2.1 OPERAND SIZE

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

## 2.2 DATA ORGANIZATION IN REGISTERS

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

### 2.2.1 Data Registers

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

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

### 2.2.2 Address Registers

Each address register and the stack pointer is 32 bits wide and holds a full 32-bit address. Address registers do not support the sized operands. Therefore, when an address register is used as a source operand, either the low order word or the entire long word operand is used depending 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 IN MEMORY

Bytes are individually addressable with the high order byte having an even address the same as the word, as shown in Figure 2.1. The low order byte has an odd address that is one count higher than the word address. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long word datum is located at address n (n even) , then the second word of that datum is located at address n + 2.

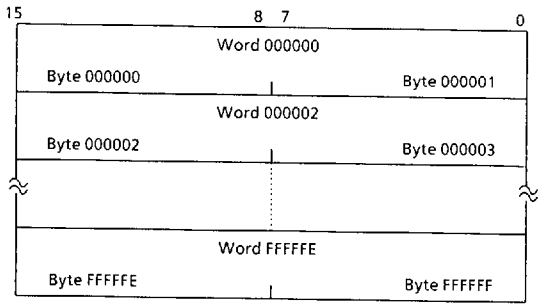



Figure 2.1 Word Organization in Memory

The data types supported by the TMP68HC000 are: bit data, integer data of 8, 16, or 32 bits, 32-bit addresses and binary coded decimal data. Each of these data types is put in memory, as shown in Figure 2.2. The numbers indicate the order in which the data accessed from the processor.

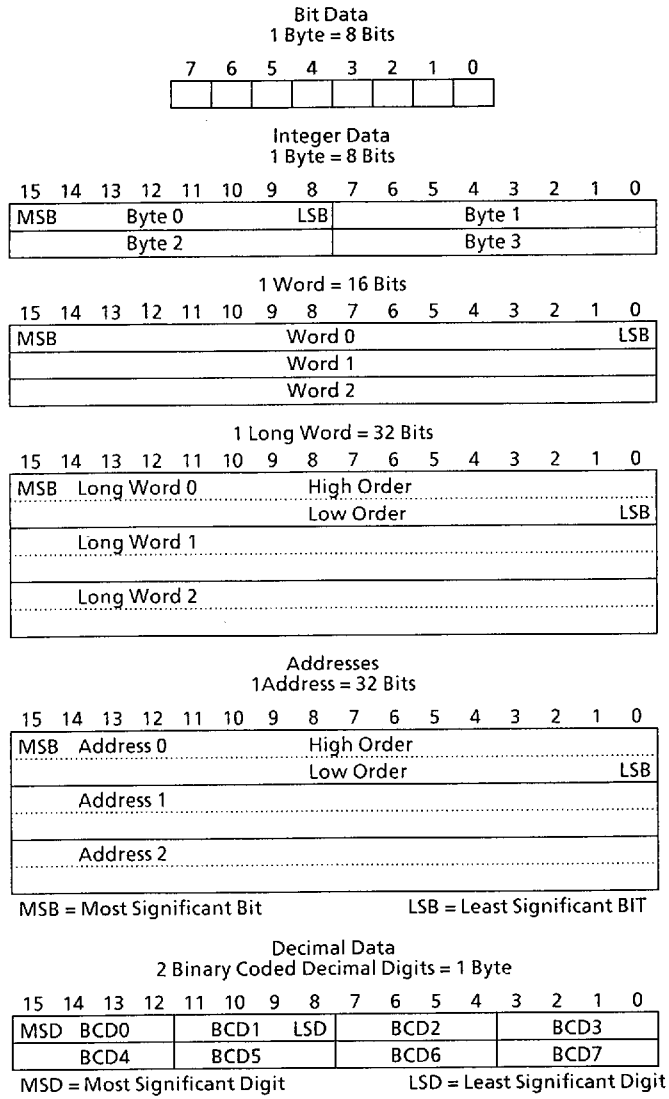



Figure 2.2 Memory Data Organization

## 2.4 ADDRESSING

Instructions for the TMP68HC000 contain two kinds of information: the type of function to be performed and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

| | | |
|---|---|---|
| Register Specification | – | the number of the register is given in the register field of their instruction. |
| Effective Address | – | use of the different effective addressing modes. |
| Implicit Reference | – | the definition of certain instructions implies the use of specific registers. |

## 2.5 INSTRUCTION FORMAT

Instructions are from one to five words in length as shown in Figure 2.3. The length of the instruction and the operation to be performed is specified by the first word of the instruction which is called the operation word. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective address mode specified in the operation word.

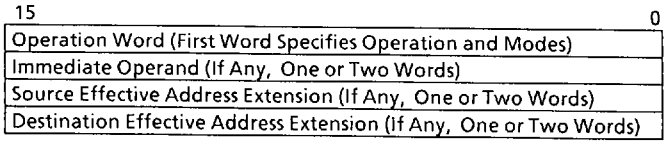



Figure 2.3 Instruction Operation Word General Format

## 2.6 PROGRAM/DATA REFERENCES

The TMP68HC000 separates memory references into two classes: program references and data references. Program references, as the name implies, are references to that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Operand reads are from the data space except in the case of the program counter relative addressing mode. All operand writes are to the data space.

## 2.7 REGISTER SPECIFICATION

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

## 2.8 EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 2.4 shows the general format of the single-effective-address instruction operation word. The effective address is composed of two 3-bit fields: the mode field and the register field. The Value in the mode field selects the different address modes. The register field contains the number of a register.

The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in the following word or words and is considered part of the instruction, as shown in Figure 2.3. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

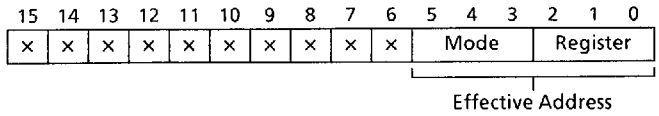



Figure 2.4 Single Effective Address Instruction Operation Word

### 2.8.1 Register Direct Modes

These effective addressing modes specify that the operand is in one of 16 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 classified as a data reference with the exception of the jump and jump-to-subroutine instructions.

#### 2.8.2.2 Address Register Indirect with Postincrement

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

2.8.2.3 Address Register Indirect with Predecrement

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

2.8.2.4 Address Register Indirect with Displacement

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

2.8.2.5 Address Register Indirect with Index

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

2.8.3.2 Absolute Long Address

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

2.8.3.3 Program Counter with Displacement

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

2.8.3.4 Program Counter with Index

This addressing mode requires one word of extension. The address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The 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 addressing mode requires either one or two words of extension depending on the size of the operation.

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

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

## 2.9 EFFECTIVE ADDRESS ENCODING SUMMARY

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

Table 2.1 Effective Address Encoding Summary

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

## 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 cannot be referenced as an address register. If the S bit indicates user state, the USP is the active system stack pointer, and the SSP cannot be referenced. Each system stack fills from high memory to low memory.

# 3. INSTRUCTION SET SUMMARY

This section contains an overview of the form and structure of the TMP68HC000 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 the general move instruction there are several special data movement instructions: move multiple registers (MOVEM) , move peripheral data (MOVEP) , exchange registers (EXG) , load effective address (LEA) , push effective address (PEA) , link stack (LINK) , unlink stack (UNLK) , and move quick (MOVEQ) . Table 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)<br>SP→An<br>SP + displacement →SP |
| MOVE | 8,16,32 | s →d |
| MOVEM | 16,32 | (EA)→An,Dn<br>An,Dn →(EA) |
| MOVEP | 16,32 | (EA)→Dn<br>Dn→(EA) |
| MOVEQ | 8 | #xxx→Dn |
| PEA | 32 | EA→ – (SP) |
| SWAP | 32 | Dn[31:16]↔Dn[15:0] |
| UNLK | – | An→SP<br>(SP) + →An |

Notes: s = source
d = destination
[ ] = bit number
–( ) = indirect with predecrement
( )+ = indirect with postincrement
#xxx = 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 word divisor to produce a word quotient with a word remainder.

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

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

Table 3.2 Integer Arithmetic Operations

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

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

## 3.3 LOGICAL OPERATIONS

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

Table 3.3 Logical Operations

| Instruction | Operand Size | Operation |
|---|---|---|
| AND | 8, 16, 32 | Dn∧(EA)→Dn<br>(EA)∧Dn→(EA)<br>(EA)∧#xxx→(EA) |
| OR | 8, 16, 32 | Dn∨(EA)→Dn<br>(EA)∨Dn→(EA)<br>(EA)∨#xxx→(EA) |
| EOR | 8, 16, 32 | (EA)⊕Dy→(EA)<br>(EA)⊕#xxx→(EA) |
| NOT | 8, 16, 32 | ~(EA)→(EA) |

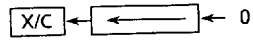
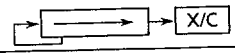
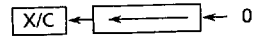
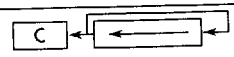
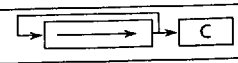
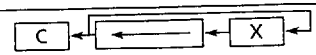
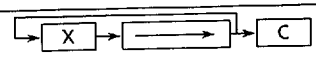
Notes: ~ = invert
#xxx = immediate data
∧ = logical AND
∨ = logical OR
⊕ = logical exclusive OR

## 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 only and allow only single-bit shifts or rotates.

Table3.4 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 instruction: bit test (BTST) , bit test and set (BSET) , bit test and clear (BCLR) , and bit test and change (BCHG) . Table 3.5 is a summary of the bit manipulation operations. (Z is bit 2 of the status register.)

Table 3.5 Bit Manipulation Operations

| Instruction | Operand Size | Operation |
|---|---|---|
| BTST | 8, 32 | ~bit of (EA) →Z |
| BSET | 8, 32 | ~bit of (EA) →Z<br>1 →bit of EA |
| BCLR | 8, 32 | ~bit of (EA) →Z<br>0 →bit of EA |
| BCHG | 8, 32 | ~bit of (EA) →Z<br>~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 | $Dx_{10} + Dy_{10} + X \rightarrow Dx$<br>$-(Ax)_{10} + -(Ay)_{10} + X \rightarrow (Ax)$ |
| SBCD | 8 | $Dx_{10} - Dy_{10} - X \rightarrow Dx$<br>$-(Ax)_{10} - -(Ay)_{10} - X \rightarrow (Ax)$ |
| NBCD | 8 | $0 - (EA)_{10} - X \rightarrow (EA)$ |

Note: -( ) = indirect with predecrement

## 3.7 PROGRAM CONTROL OPERATIONS

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

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

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

Table 3.7 Program Control Operations

| Instruction | Operation |
|---|---|
| Conditional Bcc | Branch Conditionally (14 Conditions) 8-and 16-Bit Displacement |
| DBcc | Test Condition, Decrement, and Branch 16-Bit 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 |
| Reterns RTR | Return and Restore Condition Codes |
| RTS | Return from Subroutine |

## 3.8 SYSTEM CONTROL OPERATIONS

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

Table 3.8 System Control Operations

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

# 4. SIGNAL AND BUS OPERATION DESCRIPTION

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

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

## 4.1 SIGNAL DESCRIPTION

The input and output signals can be functionally organized into the groups and the pin assignments is shown in Figure 4.1. The following paragraphs provide a brief description of the signals and a reference (if applicable) to other paragraphs that contain more detail about the function being performed.

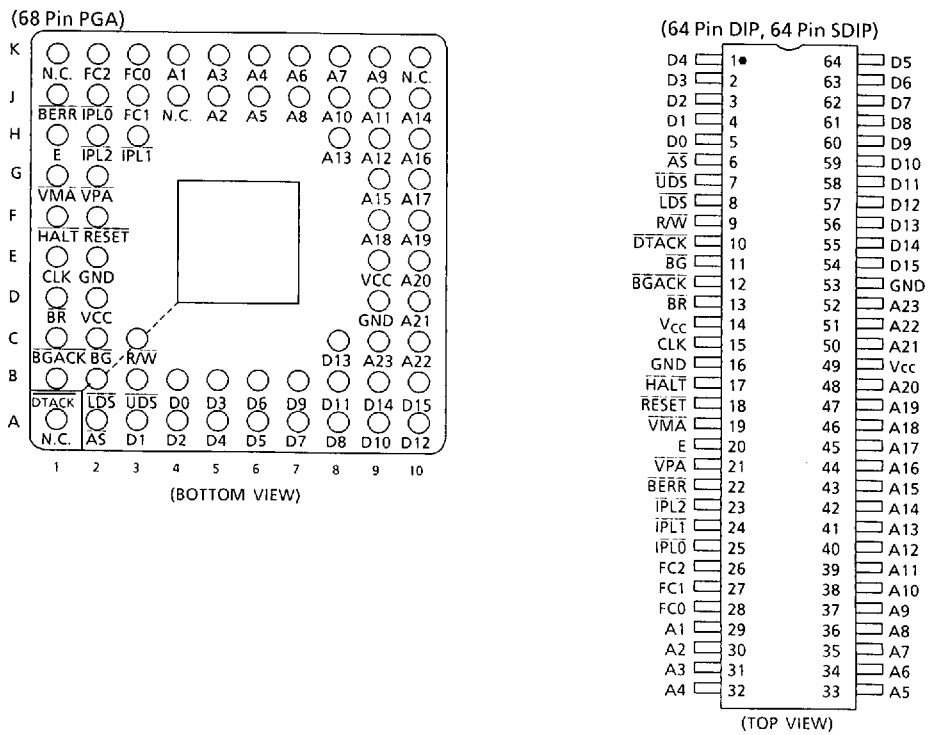
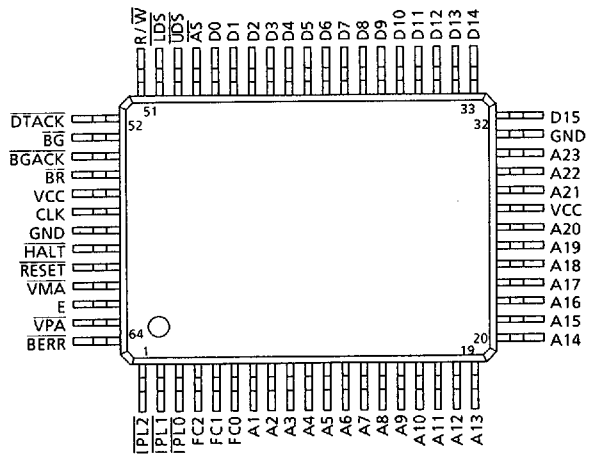


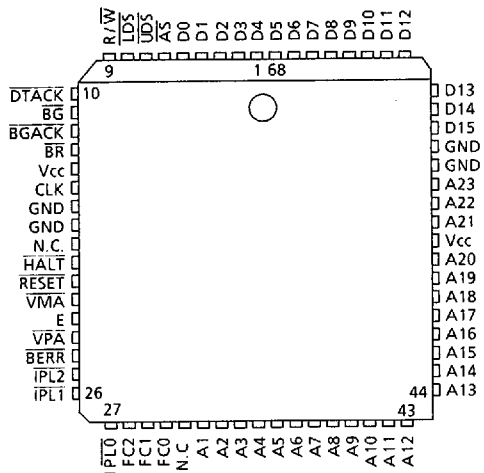

Figure 4.1 Input and Output Signals Pin Assingments (1/2)

(64 pin QFP)




(TOP VIEW)

(68 pin PLCC) * Under Development




(TOP VIEW)

図4.1 Input and Output Signals Pin Assignments (2/2)

4.1.1 Address Bus (A1~A23)

This 23-bit, unidirectional, three-state bus is capable of addressing 8 megawords of data. It provides the address for bus operation during all cycles except interrupt cycles. During interrupt cycles, address lines A1, A2, and A3 provide information about what level interrupt is being serviced while address lines A4 ~ A23 are all set to a logic high.

4.1.2 Data Bus (D0~D15)

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

4.1.3 Asynchronous Bus Control

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

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

This signal indicates that there is a valid address on the address bus.

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

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

4.1.3.3 Upper and Lower Data Storobe ($\overline{UDS}$, $\overline{LDS}$)

These signals control the flow of data on the data bus, as shown in Table 4.1. When the R/$\overline{W}$ line is high, the processor will read from the data bus as indicated. When the R/$\overline{W}$ line is low, the processor will write to the data bus as shown.

Table 4.1 Data Strobe Control of Data Bus

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

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

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

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

4.1.4 Bus Arbitration Control

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

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

This input is wire ORed with all other devices that could be bus masters. This input indicates to the processor that some other device desires to become the bus master.

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

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

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

This input indicates that some other device has become the bus master.
This signal should not be asserted until the following four conditions are met:

1. a bus grant has been received
2. address strobe is inactive which indicates that the microprocessor is not using the bus

3. data transfer acknowledge is inactive which indicates that neither memory nor peripherals are using the bus
4. bus grant acknowledge is inactive which indicates that no other device is still claiming bus mastership

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

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

4.1.6 System Control

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

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

This input informs the processor that these is a problem with the cycle currently being executed. Problems may be a result of:

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

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

Refer to "4.2.4 Bus Error and Halt Operation" for additional information about the interaction of the bus error and halt signals.

4.1.6.2 Reset ($\overline{RESET}$)

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

4.1.6.3 Halt ($\overline{HALT}$)

When this bidirectional line is driven by an external device, it will cause the processor to stop at the completion of the current bus cycle. When the procesor has been

halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state (refer to Table 4.3) . Refer to "4.2.4 Bus Error and Halt Operation" for additional information about the interaction between the $\overline{HALT}$ and bus error signals.

When the processor has stopped executing instructions, such as in a double bus fault condition (refer to "4.2.4.4 Double Bus Faults") , the $\overline{HALT}$ 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 TMP68HC000. These signals are explained in the following paragraphs.

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 ten TMP68HC000 clock periods (six clocks low, four clocks high) . Enable is generated by an internal ring counter which may come up in any state (i.e., at power on, it is impossible to guarantee phase relationship of E to CLK) . E is a free-running clock and runs regardless of the state of the bus on the MPU.

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

This input indicates that the device or region addressed is an 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 an interrupt. Refer to "SECTION 6 INTERFACE WITH 6800 PERIPHERALS".

4.1.7.3 Valid Memory Address ($\overline{VMA}$)

This output is used to indicate to 6800 peripheral devices that these is a valid address on the address bus and the processor is synchronized to enable. This signal only responds to a valid peripheral address ($\overline{VPA}$) input which indicates that the peripheral is an 6800 Family device.

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

| FC2 | FC1 | FC0 | Cycle Type |
|---|---|---|---|
| Low | Low | Low | (Undefined, Reserved) |
| Low | Low | High | User Data |
| Low | High | Low | User Program |
| Low | High | High | (Undefined, Reserved) |
| High | Low | Low | (Undefined, Reserved) |
| High | Low | High | Supervisor Data |
| High | High | Low | Supervisor Program |
| High | High | High | Interrupt Acknowledge |

4.1.9 Clock (CLK)

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

### 4.1.10 Signal Summary

Table 4.3 is a summary of all the signals discussed in the previous paragraphs.

Table 4.3 Signal Summary

| Signal Name | Mnemonic | Input / Output | Active State | 3 State | Hi-Z | |
|---|---|---|---|---|---|---|
| | | | | | On $\overline{HALT}$ | On $\overline{BGACK}$ |
| Address Bus | A1~A23 | Output | High | Yes | Yes | Yes |
| Data Bus | D0~D15 | Input / Output | High | Yes | Yes | Yes |
| Address Strove | $\overline{AS}$ | Output | Low | Yes | No | Yes |
| Read / Write | R/$\overline{W}$ | Output | Read-High Write-Low | Yes | No | Yes |
| Upper and Lower Data strobes | $\overline{UDS}$, $\overline{LDS}$ | Output | Low | Yes | No | Yes |
| Data Transfer Acknowledge | $\overline{DTACK}$ | Input | Low | No | No | No |
| Bus Request | $\overline{BR}$ | Input | Low | No | No | No |
| Bus Grant | $\overline{BG}$ | Output | Low | No | No | No |
| Bus Grant Acknowledge | $\overline{BGACK}$ | Input | Low | No | No | No |
| Interrupt Priority Level | $\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$ | Input | Low | No | No | No |
| Bus Error | $\overline{BERR}$ | Input | Low | No | No | No |
| Reset | $\overline{RESET}$ | Input / Output | Low | Yes | No[1] | No[1] |
| Halt | $\overline{HALT}$ | Input / Output | Low | Yes | No[1] | No[1] |
| Enable | E | Output | High | No | No | No |
| Valid Memory Address | $\overline{VMA}$ | Output | Low | Yes | No | Yes |
| Valid Peripheral Address | $\overline{VPA}$ | Input | Low | No | No | No |
| Function Code Output | FC0, FC1, FC2 | Output | High | Yes | No | Yes |
| Clock | CLK | Input | High | No | No | No |
| Power Input | $V_{CC}$ | Input | — | — | — | — |
| Ground | 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 following leads:

1. address bus A1~A23
2. data bus D0~D15
3. control signals

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

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the TMP68HC000 for interlocked multiprocessor communications.

#### 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 upper and lower both simultaneously by asserting both upper and lower data strobes. When the instruction specifies byte operation, the processor uses an internal A0 bit to determine which byte to read and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. When the data is received, the processor correctly positions it internally.

A word read cycle flowchart is given in Figure 4.2. A byte read cycle flowchart is given in Figure 4.3. Read cycle timing is given in Figure 4.4. Figure 4.5 details word and byte read cycle operations.

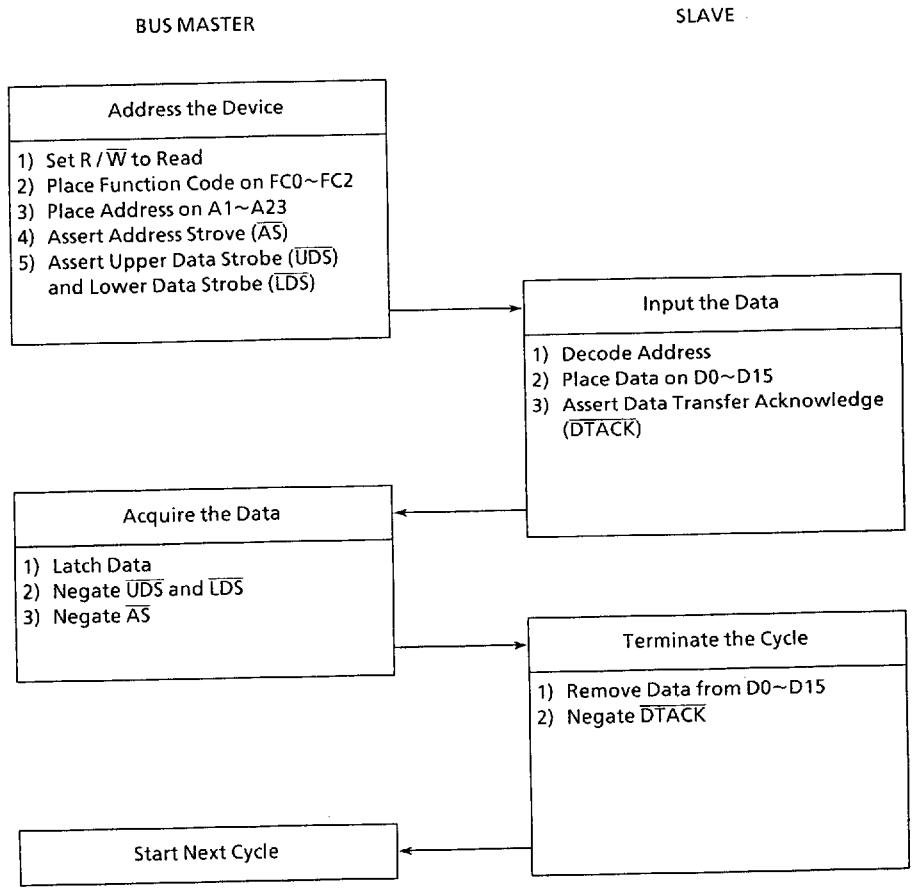



Figure 4.2 Word Read Cycle Flowchart

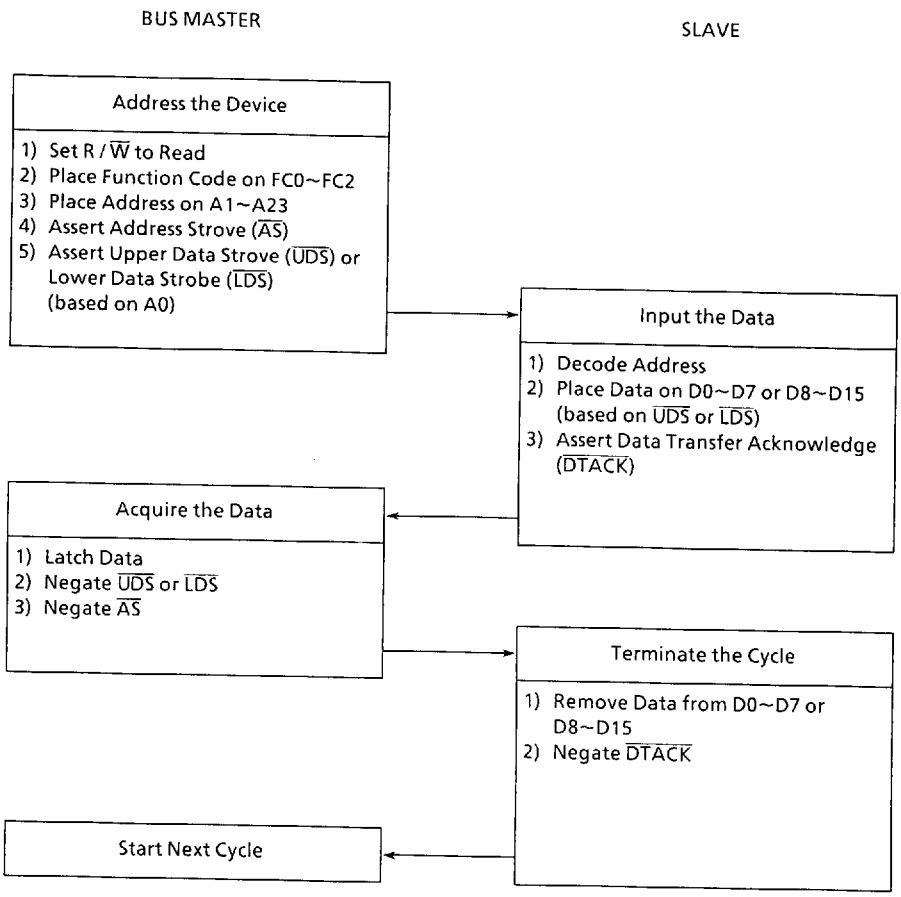



Figure 4.3 Byte Read Cycle Flowchart

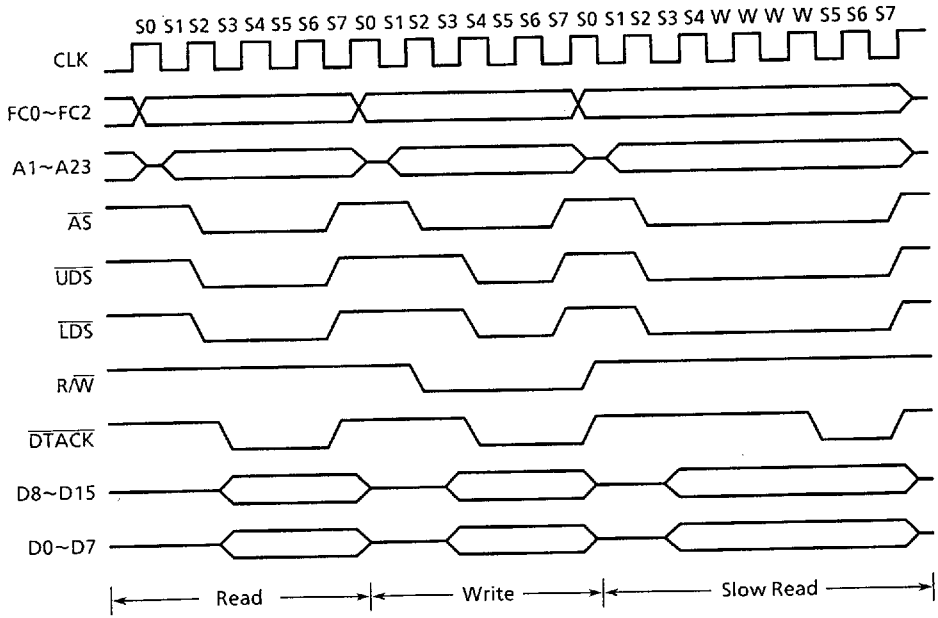



Figure 4.4 Read and Write Cycle Timing Diagram

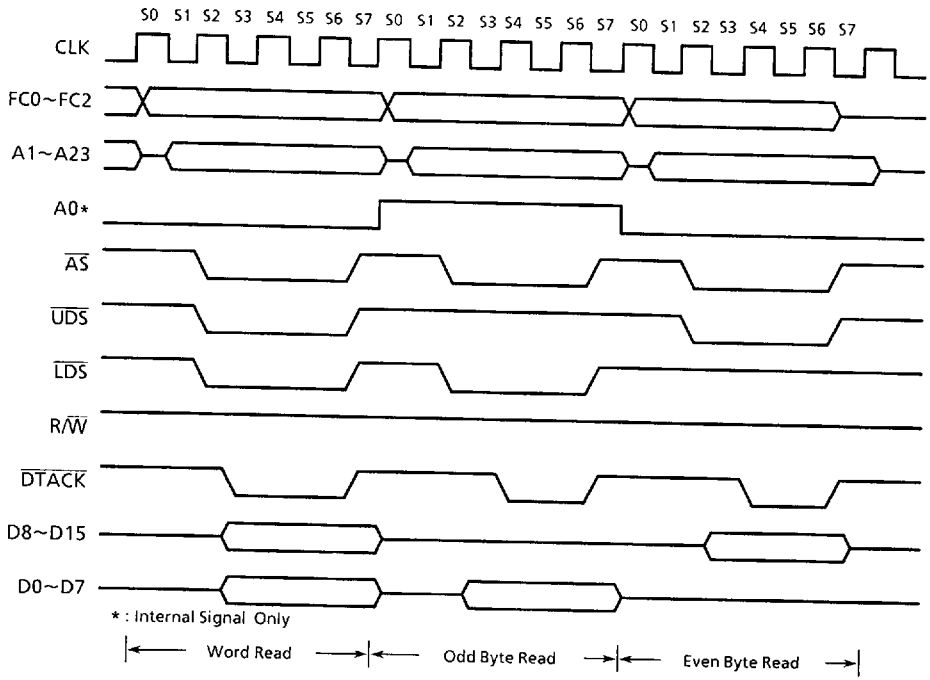



Figure 4.5 Word and Byte Read Cycle Timing Diagram

4.2.1.2 Write Cycle

During a write cycle, the processor sends data to either the memory or a peripheral device. The processor writes bytes of data in all cases. If the instruction specifies a word operation, the processor writes both bytes. When the instruction specifies a byte operation, the processor uses an internal A0 bit to determine which byte to write and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. A word write flowchart is given in Figure 4.6. A byte write cycle flowchart is given in Figure 4.7. Write cycle timing is given in Figure 4.4. Figure 4.8 details word and byte write cycle operation.

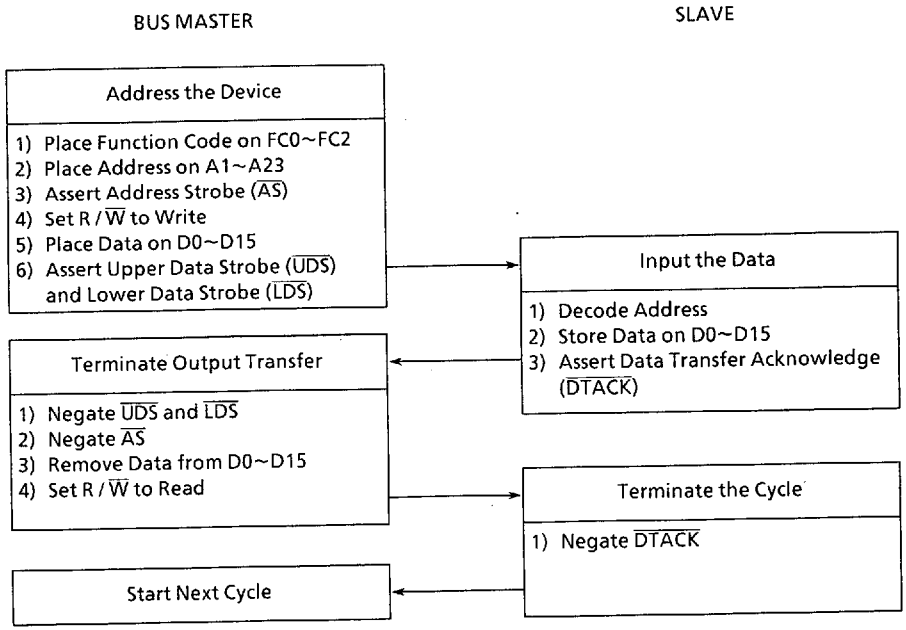



Figure 4.6 Word Write Cycle Flowchart

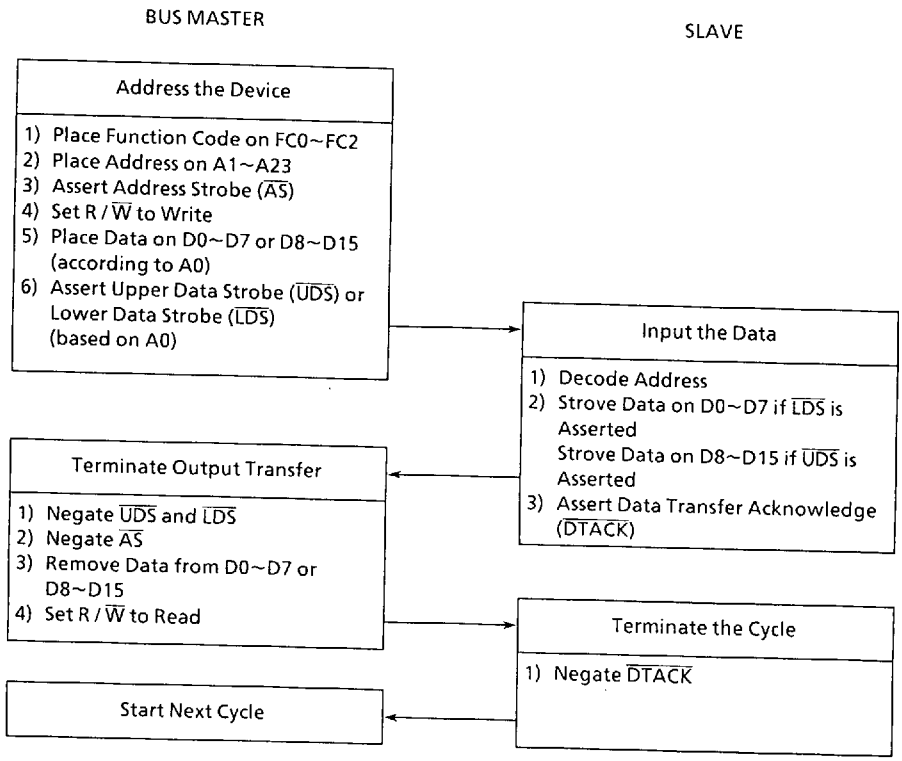



Figure 4.7 Byte Write Cycle Flowchart

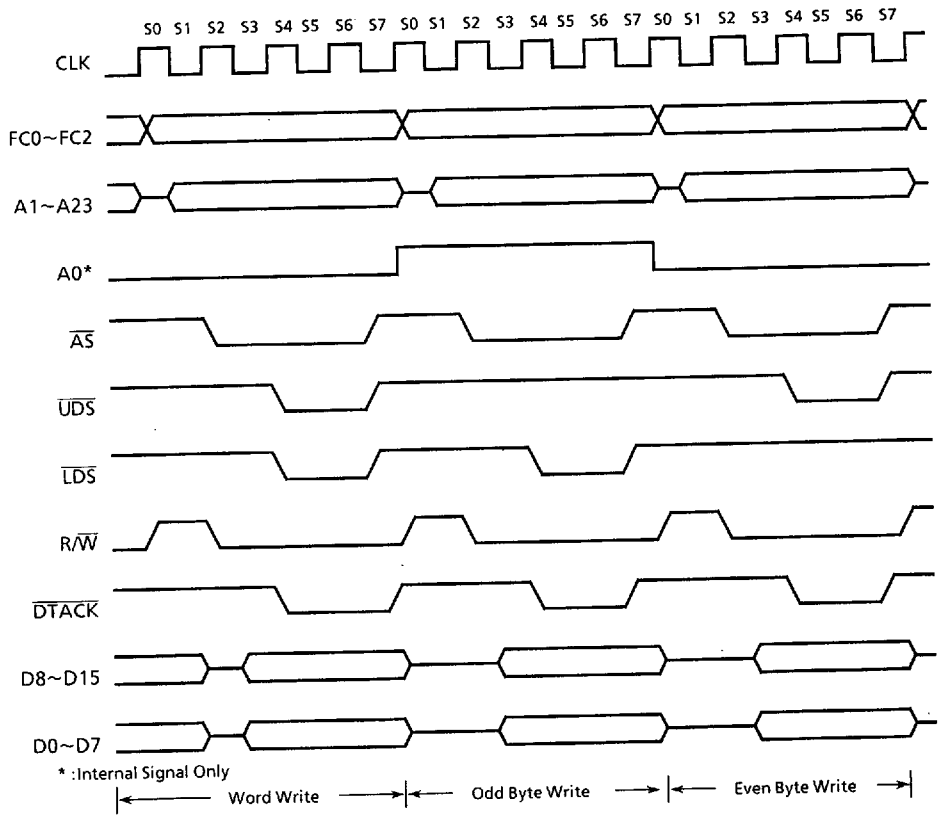



Figure 4.8 Word and Byte Write Cycle Timing Diagram

4.2.1.3 Read-Modify-Write Cycle

The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the TMP68HC000, this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycles and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write flowchart is given in Figure 4.9 and a timing diagram is given in Figure 4.10.

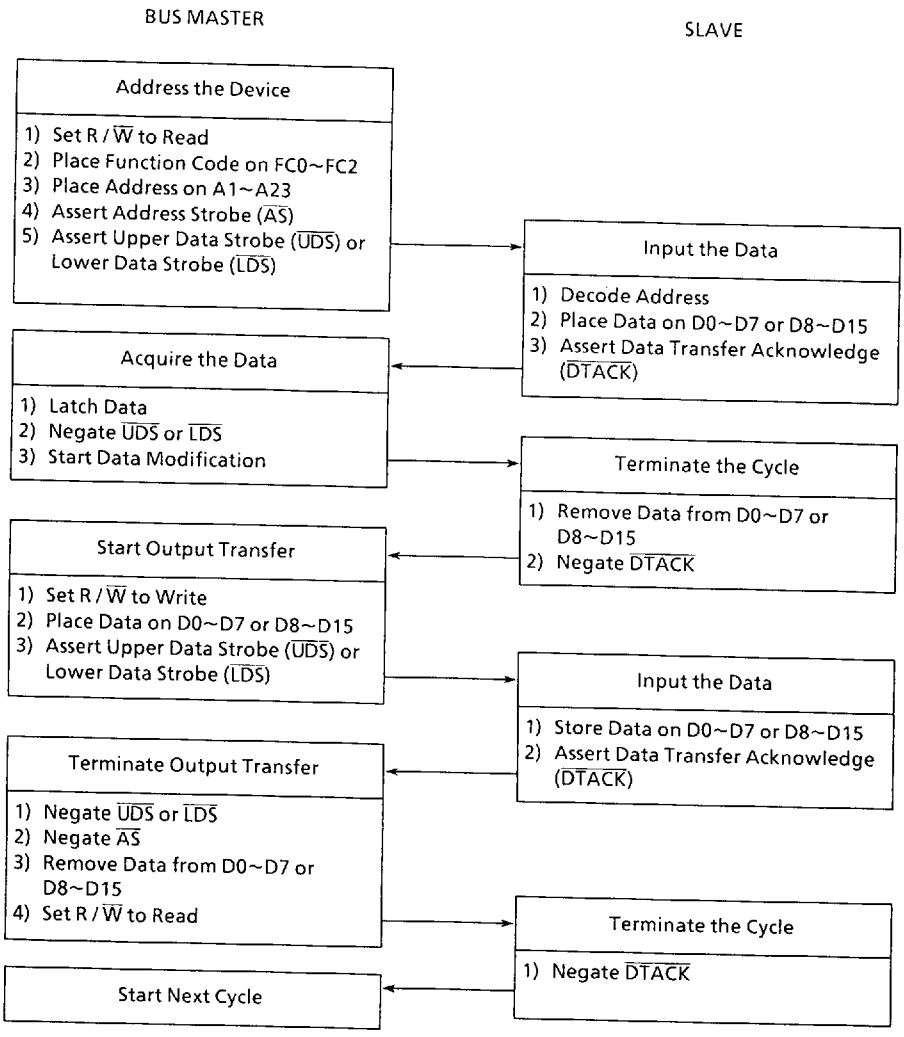



Figure 4.9 Read-Modify-Write Cycle Flowchart

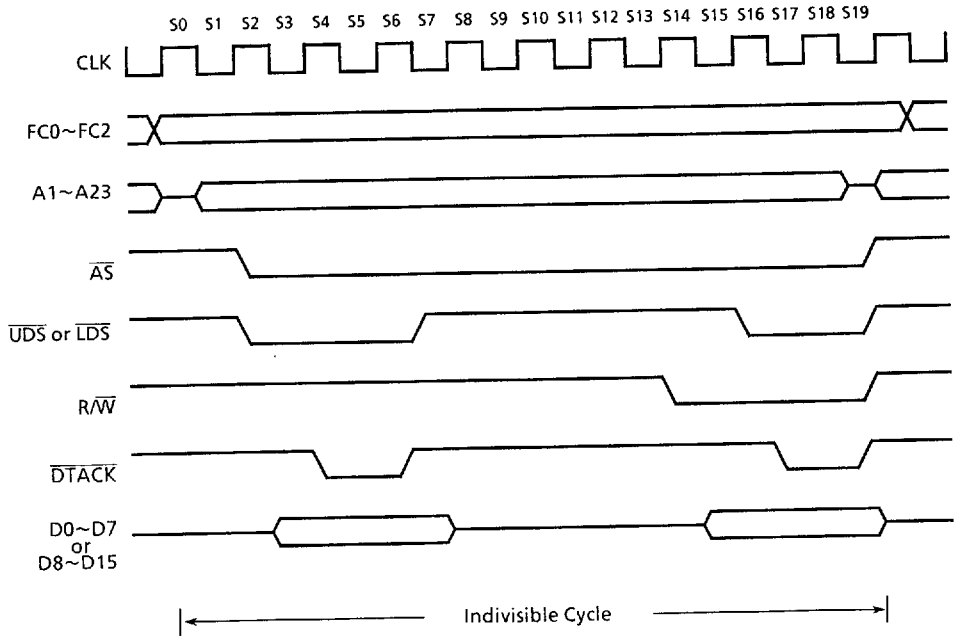



Figure 4.10 Read-Modify-Write Cycle Timing Diagram

4.2.2 Bus Arbitration

Bus arbitration is 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
3. acknowledging that mastership has been assumed

Figure 4.11 is a flowchart showing the detail involved in a request from a single device. Figure 4.12 is a timing diagram for the same operation. This technique allows processing of bus requests during data transfer cycles. The timing diagram shows that the bus request is negated at the time that an acknowledge is asserted. This type of operation would be true for a system consisting of the processor and one device capable of bus mastership. In systems having a number of devices capable of the busmastership, 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 ($\overline{BGACK}$) signal.

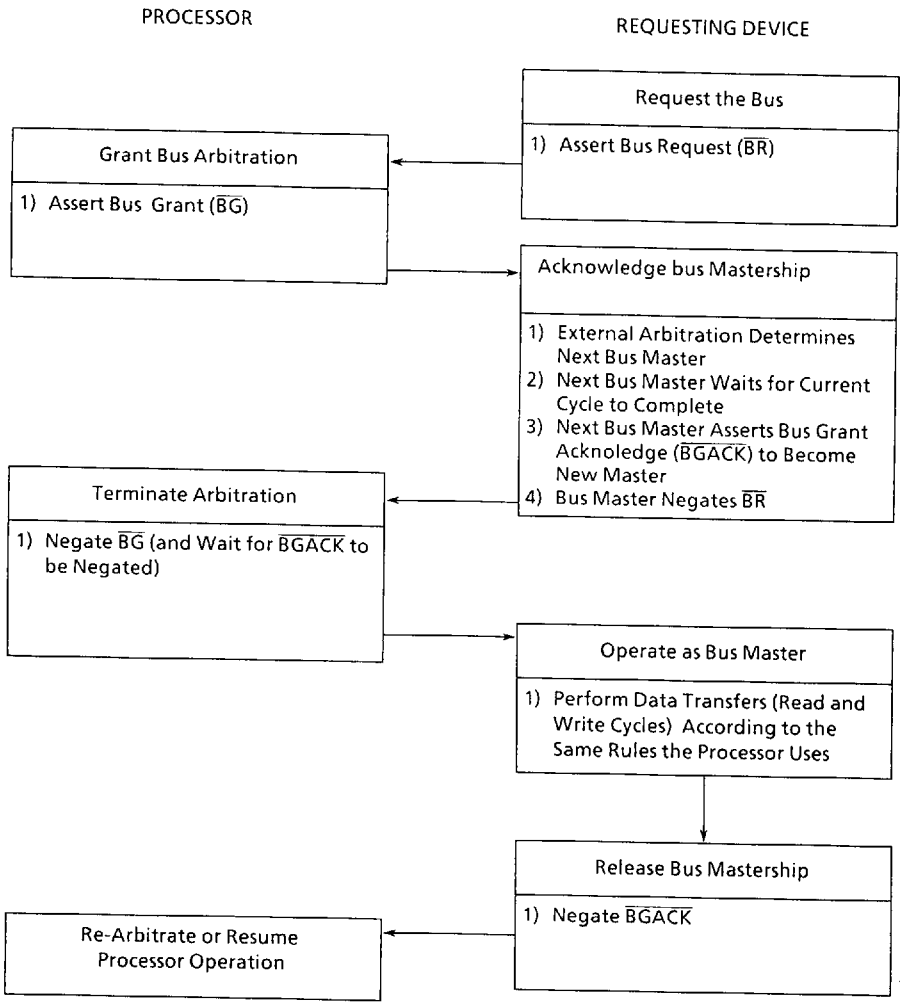



Figure 4.11 Bus Arbitration Cycle Flowchart

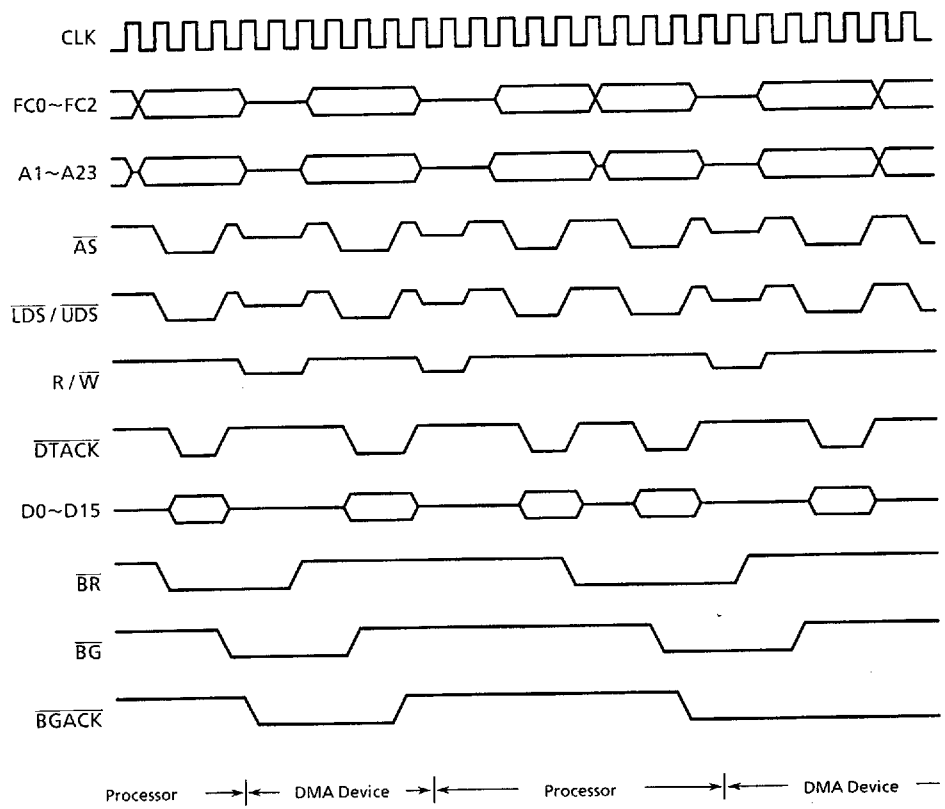



Figure 4.12 Bus Arbitration Cycle Timing Diagram

4.2.2.1 Requesting the Bus

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

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

4.2.2.2 Receiving the Bus Grant

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

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

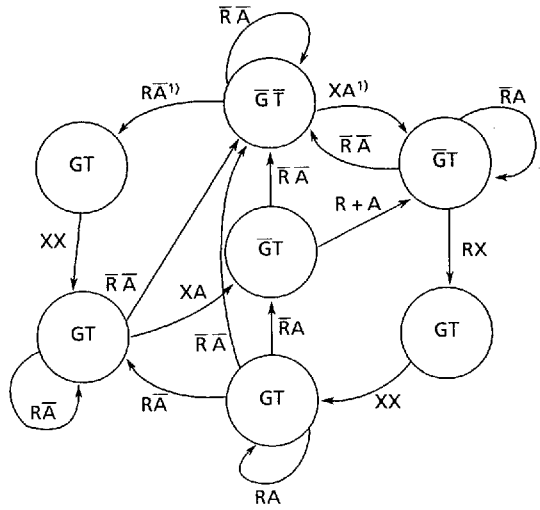
4.2.2.3 Acknowledgement of Mastership

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

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

4.2.3 Bus Arbitration Control

The bus arbitration control unit in the TMP68HC000 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 4.13. All asynchronous signals to the TMP68HC000 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 4.14) . The input signal is sampled on the falling edge of the clock and is valid internally after the next falling edge.




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

Notes :
1) State machine will not change if 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 4.13 TMP68HC000 Bus Arbitration Unit State Diagram

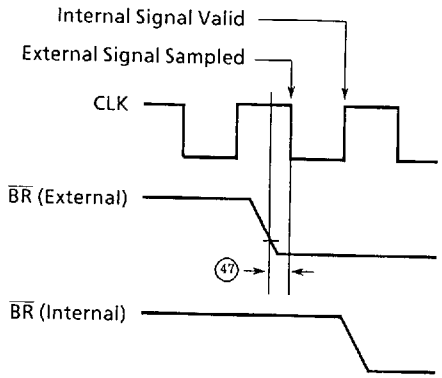



Figure 4.14 Timing Relationship of External Asynchronous Inputs to Internal Signals

As shown in Figure 4.13, input signals labeled R and A are internally synchronized on the bus request and bus grant acknowledge pins respectively. The bus grant output is labeled G and the internal three-state control signal T. If T is true, the address, data, and control buses are placed in a high-impedance state when $\overline{AS}$ is negated. All signals are shown in positive logic (active high) regardless of their true active voltage level. State changes (valid outputs) occur on the next rising edge after the internal signal is valid.

A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 4.15. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 4.16.

If a bus request is made at a time when the MPU has already begun a bus cycle but $\overline{AS}$ has not been asserted (bus state S0) , $\overline{BG}$ will not be asserted on the next rising edge. Instead, $\overline{BG}$ will be delayed until the second rising edge following its internal assertion. This sequence is shown in Figure 4.17.

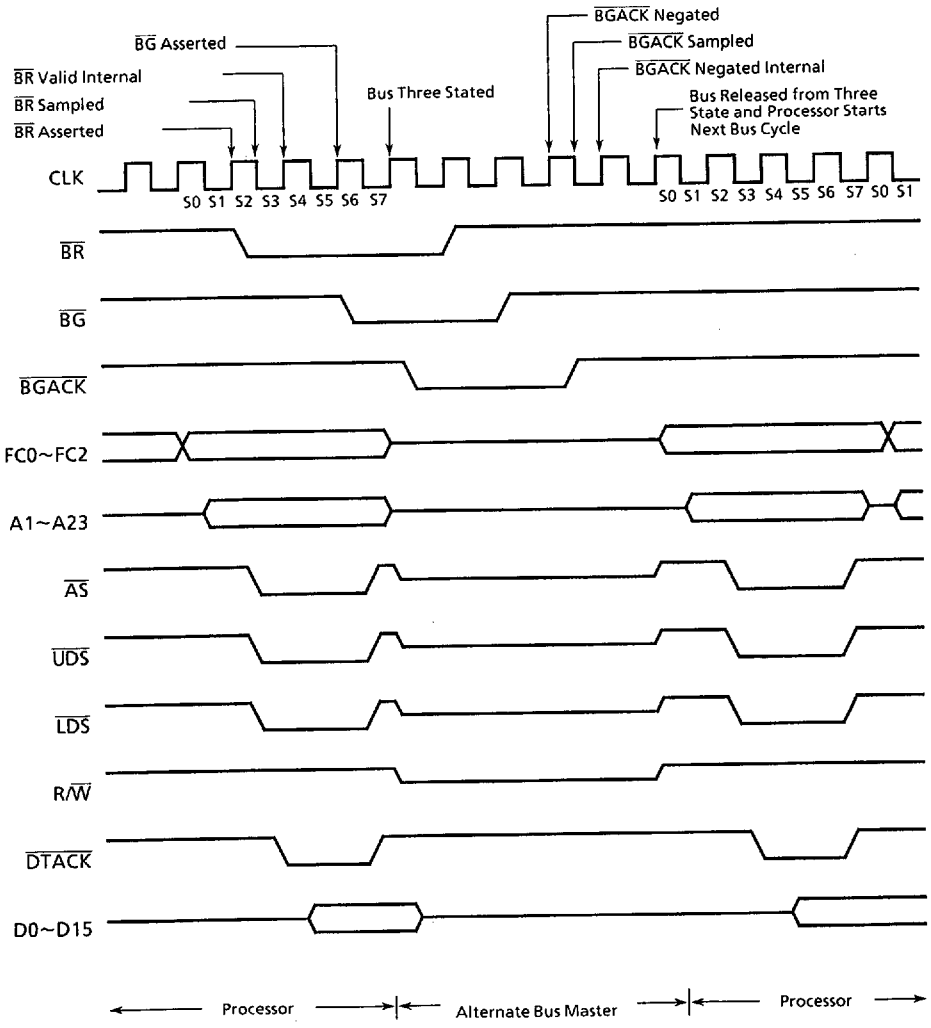



Figure 4.15 Bus Arbitration Timing Diagram – Processor Active

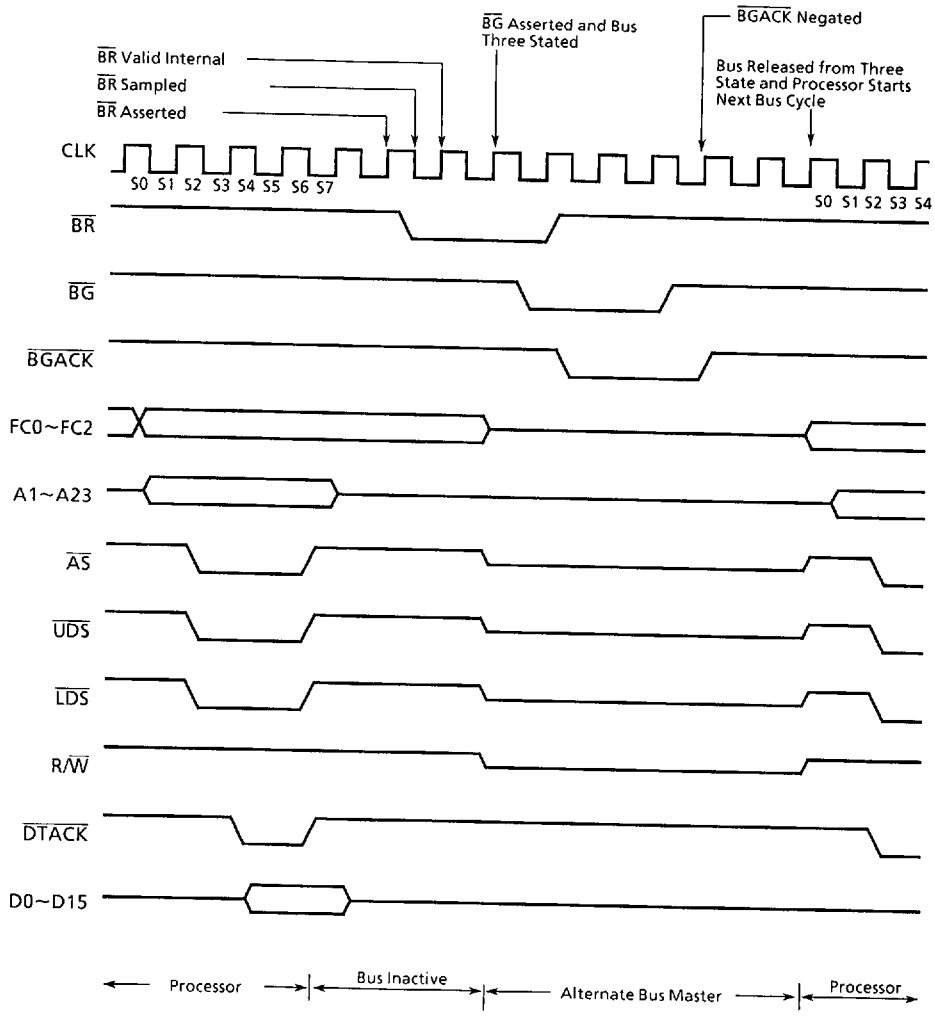



Figure 4.16 Bus Arbitration Timing Diagram – Bus Inactive

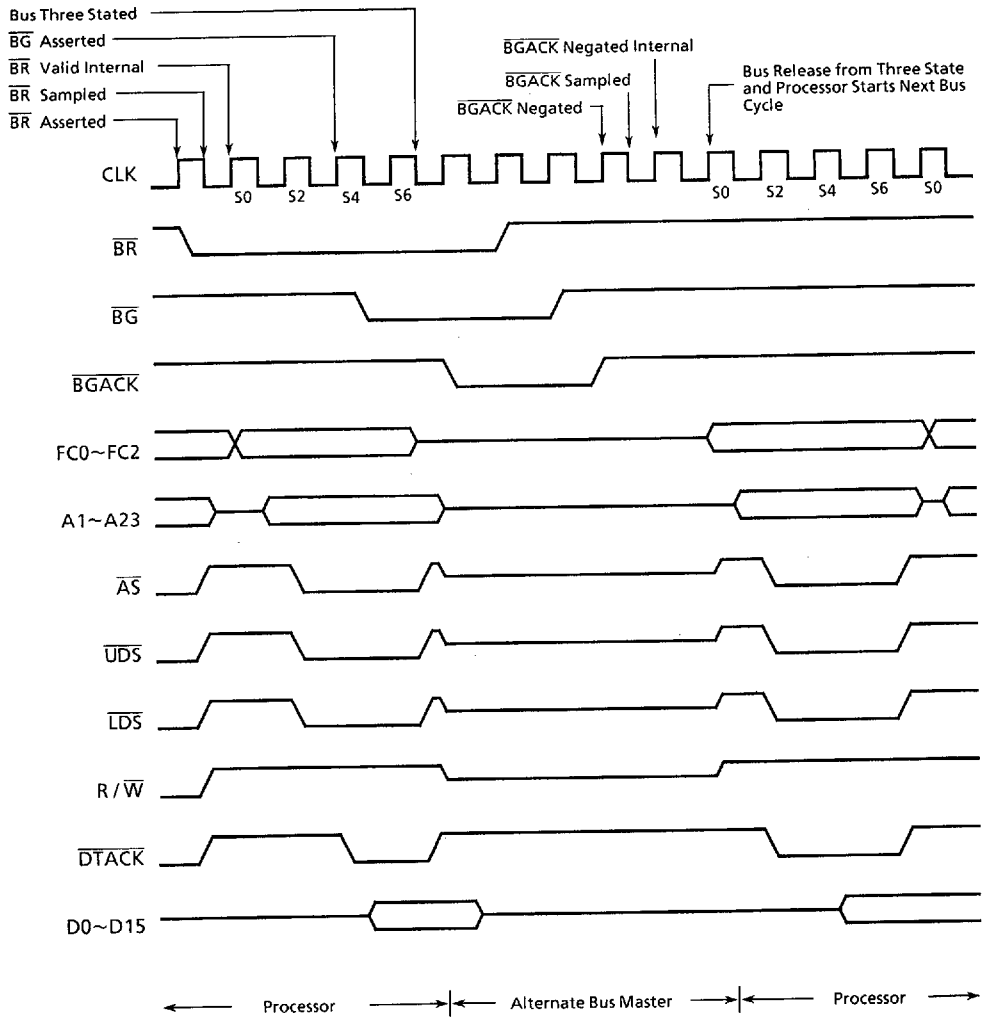



Figure 4.17 Bus Arbitration Timing Diagram – Special Case

4.2.4 Bus Error and Halt Operation

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

4.2.4.1 Bus Error Operation

When the bus error signal is asserted, the current bus cycle is terminated. If $\overline{\text{BERR}}$ is asserted before the falling edge of S2, $\overline{\text{AS}}$ will be negated in S7 in either a read or write cycle. As long as $\overline{\text{BERR}}$ remains asserted, the data and address buses will be in the high-impedence state. When $\overline{\text{BERR}}$ is negated, the processor will begin stacking for exception processing. Figure 4.18 is a timing diagram for the exception sequence. The sequence is composed of the following elements:

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

The stacking of the program counter and status register is the same as if an interrupt had occurred. Several additional items are stacked when a bus error occurs. These items are used to determine the nature of the error and correct it, if possible. The bus error vector is vector number two located at address $000008. The processor loads the new program counter from this location. A software bus error handler routine is then executed by the processor. Refer to "5.2 EXCEPTION PROCESSING" for additional information.

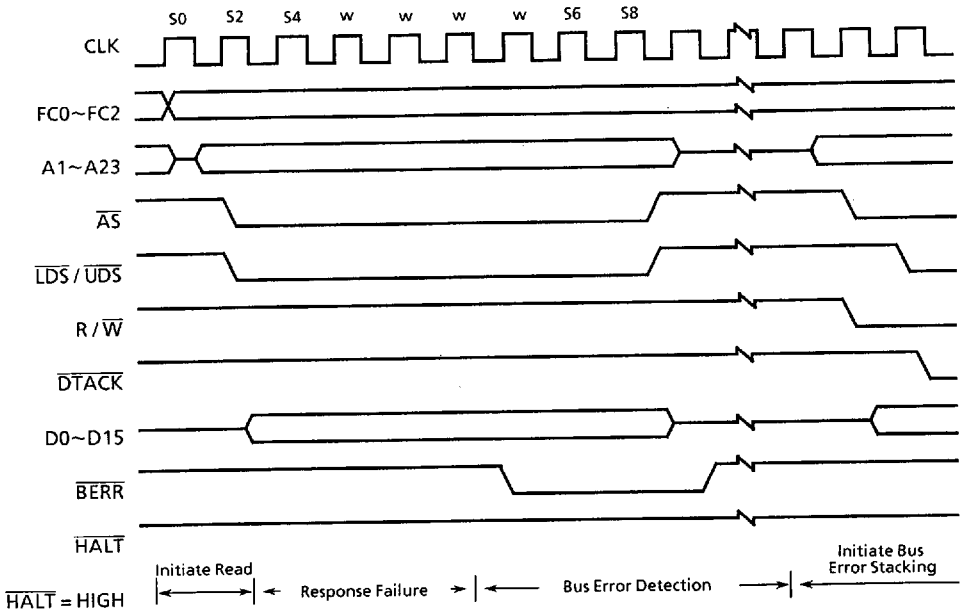



Figure 4.18 Bus Error Timing Diagram

4.2.4.2 Re-Run Operation

When, during a bus cycle, the processor receives a bus error signal and the halt pin is being driven by an external device, the processor enters the re-run sequence. Figure 4.19 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-impedence state. The processor remains "halted", and will not run another bus cycle until the halt signal is removed by external logic. Then the processor will re-run the previous cycle using the same function codes, the same data (for a write operation), and the same controls. The bus error signal should be removed at least one clock cycle before the halt signal is removed.

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

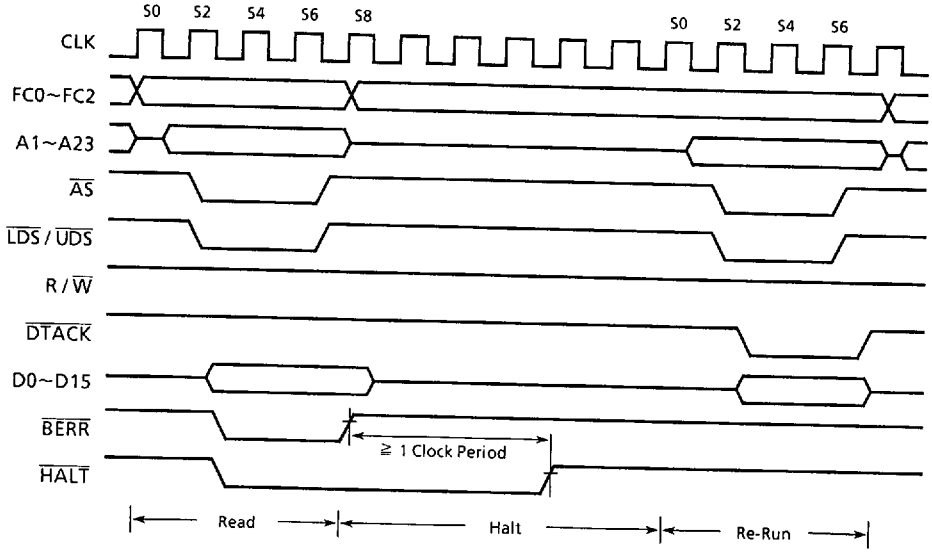



Figure 4.19 Re-Run Bus Cycle Timing Diagram

4.2.4.3 Halt Operation

The halt input signal to the TMP68HC000 performs a halt/run/single-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 nothing) and when the halt signal is constantly inactive the processer "runs" (does something).

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

Figure 4.20 details the timing required for correct single-step operations. Some care must be exercised to avoid harmful interactions between the bus error signal and the halt pin when using the single-cycle mode as a debugging tool. This is also true of interactions between the halt and reset lines since these can reset the machine.

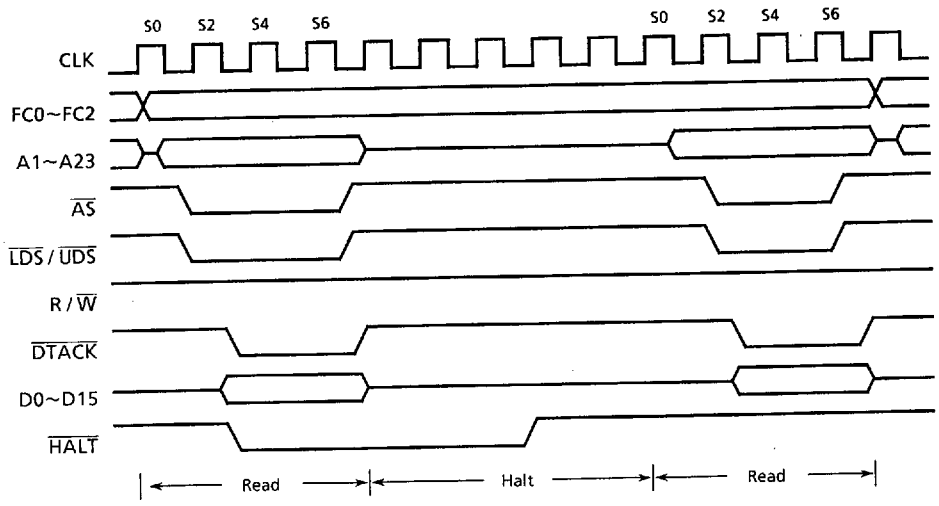



Figure 4.20 Halt Processor Timing Diagram

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

1. address line
2. data lines

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

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

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

4.2.4.4 Double Bus Faults

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

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

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

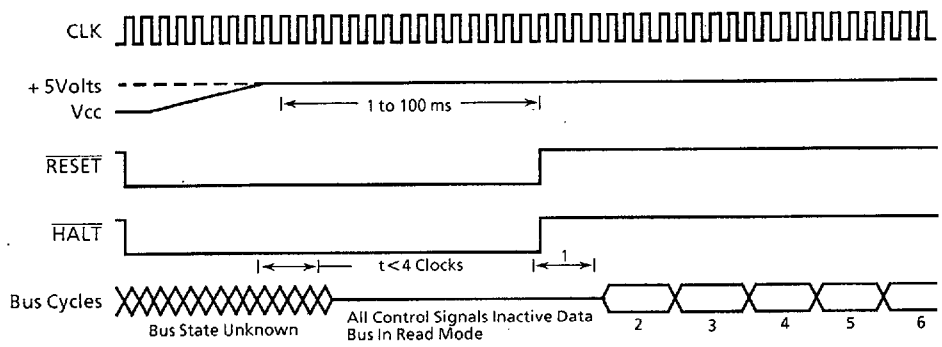
4.2.5 Reset Operation

The reset signal is a bidirectional signal that allows either the processor or an external signal to reset the system. Figure 4.21 is a timing diagram for the reset operation. Both the halt and reset lines must be asserted to ensure total reset of the processor.

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

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

Asserting the reset and halt lines for ten clock cycles will cause a processor reset, except when Vcc is initially applied to the processor. In this case, an external reset must be applied for at least 100 milliseconds.




Notes :

(1) Internal start-up time
(2) SSP High read in here
(3) SSP Low read in here
(4) PC High read in here
(5) PC Low read in here
(6) First instruction fetched here

Figure 4.21 Reset Operation 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 TMP68HC000 clock. This will assure that when two signals are asserted simultaneously, the required setup time (#47) for both of them will be met during the same bus state.

This, or some equivalent precaution, should be designed external to the TMP68HC000. Parameter #48 is intended to ensure this operation in a totally asynchronous system, and may be ignored if the above conditions are met.

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

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

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

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

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

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

Table 4.4 $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ Assertion Results

| Case No. | control Signal | Asserted on Rising Edge State | | Result |
|---|---|---|---|---|
| | | N | N + 2 | |
| 1 | $\overline{\text{DTACK}}$<br>$\overline{\text{BERR}}$<br>$\overline{\text{HALT}}$ | A<br>NA<br>NA | S<br>X<br>X | Normal cycle terminate and continue |
| 2 | $\overline{\text{DTACK}}$<br>$\overline{\text{BERR}}$<br>$\overline{\text{HALT}}$ | A<br>NA<br>A | S<br>X<br>S | Normal cycle terminate and halt. Continue when $\overline{\text{HALT}}$ removed. |
| 3 | $\overline{\text{DTACK}}$<br>$\overline{\text{BERR}}$<br>$\overline{\text{HALT}}$ | NA<br>NA<br>A | A<br>NA<br>S | Normal cycle terminate and halt. Continue when $\overline{\text{HALT}}$ removed. |
| 4 | $\overline{\text{DTACK}}$<br>$\overline{\text{BERR}}$<br>$\overline{\text{HALT}}$ | X<br>A<br>NA | X<br>S<br>NA | Terminate and take bus error trap. |
| 5 | $\overline{\text{DTACK}}$<br>$\overline{\text{BERR}}$<br>$\overline{\text{HALT}}$ | NA<br>A<br>NA | X<br>S<br>A | Terminate and re-run. |
| 6 | $\overline{\text{DTACK}}$<br>$\overline{\text{BERR}}$<br>$\overline{\text{HALT}}$ | X<br>A<br>A | X<br>S<br>S | Terminate and re-run when $\overline{\text{HALT}}$ removed. |
| 7 | $\overline{\text{DTACK}}$<br>$\overline{\text{BERR}}$<br>$\overline{\text{HALT}}$ | NA<br>NA<br>A | X<br>A<br>S | Terminate and re-run when $\overline{\text{HALT}}$ removed. |

Legend :

N : the number of the current even bus state (e.g., S4, S6, etc.)

A : signal is asserted in this bus state

NA : signal is not asserted in this state

X : don't care

S : signal was asserted in previous state and remains asserted in this state

Table4.5 $\overline{BERR}$ and $\overline{HALT}$ 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 | $\overline{BERR}$<br>$\overline{HALT}$ | ●<br>● | or<br>or | ●<br>● | Takes bus error trap. |
| Re-run | $\overline{BERR}$<br>$\overline{HALT}$ | ●<br>● | or | ● | Illegal sequence; usually traps to vector number 0. |
| Re-run | $\overline{BERR}$<br>$\overline{HALT}$ | ● | | ● | Re-runs the bus cycle. |
| Normal | $\overline{BERR}$<br>$\overline{HALT}$ | ●<br>● | or | ● | May lengthen next cycle. |
| Normal | $\overline{BERR}$<br>$\overline{HALT}$ | ● | or | ●<br>none | If next cycle is started it will be terminated as a bus error. |

● : Signal is negated in this bus state.

EXAMPLE A : A system uses a watch-dog timer to terminate accesses to unpopulated address space. The timer asserts $\overline{DTACK}$ and $\overline{BERR}$ simultaneously after time out (case 4).

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

(a) delay $\overline{DTACK}$ until data verified and return $\overline{BERR}$ and $\overline{HALT}$ simultaneously to re-run error cycle (case 6), or if valid, return $\overline{DTACK}$ (case 1)

(b) delay $\overline{DTACK}$ until data verified and return $\overline{BERR}$ at same time as $\overline{DTACK}$ if data in error (case 4).

## 4.4 ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION

### 4.4.1 Asynchronous Operation

To achieve clock frequency independence at a system level, the TMP68HC000 can be used in an asynchronous manner. This entails using only the bus handshake lines ($\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, $\overline{DTACK}$, $\overline{BERR}$, $\overline{HALT}$, and $\overline{VPA}$) to control the data transfer. Using this method, $\overline{AS}$ signals the start of a bus cycle and the data strobes are used as a condition for valid data on a write cycle. The slave device (memory or peripheral) then responds by placing the requested data on the data bus for a read cycle or latching data on a write cycle and asserting the data transfer acknowledge signal ($\overline{DTACK}$) to terminate the bus cycle. If no slave responds or the access is invalid, external control logic asserts the $\overline{BERR}$, or $\overline{BERR}$ and $\overline{HALT}$, signal to abort or rerun the bus cycle.

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

### 4.4.2 Synchronous Operation

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

Note : During an active bue cycle, $\overline{BERR}$ is sampled on every falling edge of the clock starting with S2. $\overline{DTACK}$ is sampled on every falling edge of the clock starting with S4 and data is latched on the falling edge of S6 during a read. The bus cycle will then be terminated in S7 except when $\overline{BERR}$ is asserted in the absence of $\overline{DTACK}$, in which case it will terminate one clock cycle later in S9, $\overline{VPA}$ issampled only on the third falling edge of the system clock before the rising edge of the E clock.

# 5. PROCESSING STATES

This section describes the actions of the TMP68HC000 which are outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the processor interrupt priority mask. Finally, the sequence of memory references and actions taken by the processor on exception conditions are detailed.

The TMP68HC000 is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of the normal state is the stopped state which the processor enters when a stop instruction is executed. In this state, no further references are made.

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

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

## 5.1 PRIVILEGE STATES

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

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

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

5.1.1 Supervisor State

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

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

5.1.2 User State

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

Most instructions execute the same in user state as in the supervisor state. However, some instructions which have important system effects are made privileged. User programs are not permitted to execute the stop instruction or the reset instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole state register are privileged. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE to USP) and move from user stack pointer (MOVE from USP) instructions are also privileged. The bus cycles generated by an instruction executed in the user state are classified as user state references. This allows an external memory management device to translate the address and to control access to protected portions of the address space. While the processor is in the user privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly, access the user stack pointer.

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 state, such as interrput acknowledge. Table 5.1 lists the classification of references.

Table 5.1 Bus Cycle 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 |

Note: L: LOW H: HIGH

## 5.2 EXCEPTION PROCESSING

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

5.2.1 Exception Vectors

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

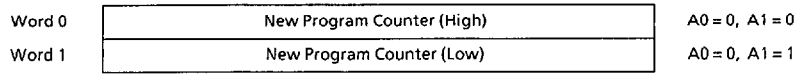



Figure5.1 Format of Vector Table Entries

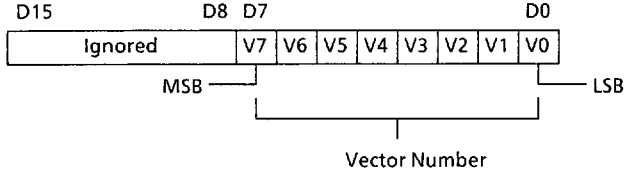



Figure5.2 Vector Number Format

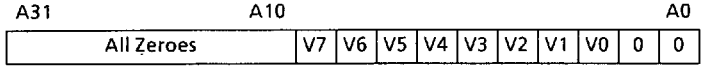



Figure5.3 Exception Vector Address Calculation

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 Exception Vector Table

| Vector | | Address | | | Assignment |
|---|---|---|---|---|---|
| Dec | Hex | Dec | Hex | Space | |
| 0 | 0 | 0 | 000 | SP | Reset:Initial SSP |
| — | — | 4 | 004 | SP | Reset:Initial PC |
| 2 | 2 | 8 | 008 | SD | Bus Error |
| 3 | 3 | 12 | 00C | SD | Address Error |
| 4 | 4 | 16 | 010 | SD | Illegal Instruction |
| 5 | 5 | 20 | 014 | SD | Zero Divide |
| 6 | 6 | 24 | 018 | SD | CHK Instruction |
| 7 | 7 | 28 | 01C | SD | TRAPV Instruction |
| 8 | 8 | 32 | 020 | SD | Privilege Violation |
| 9 | 9 | 36 | 024 | SD | Trace |
| 10 | A | 40 | 028 | SD | Line 1010 Emulator |
| 11 | B | 44 | 02C | SD | Line 1111 Emulator |
| 12* | C | 48 | 030 | SD | (Unassigned, Reserved) |
| 13* | D | 52 | 034 | SD | (Unassigned, Reserved) |
| 14* | E | 56 | 038 | SD | (Unassigned, Reserved) |
| 15 | F | 60 | 03C | SD | Uninitialized Interrupt Vector |
| 16 to 23* | 10 to 17 | 64<br>95 | 040<br>05F | SD | (Unassigned, Reserved)<br>— |
| 24 | 18 | 96 | 060 | SD | Spurious Interrupt |
| 25 | 19 | 100 | 064 | SD | Level 1 Interrupt Autovector |
| 26 | 1A | 104 | 068 | SD | Level 2 Interrupt Autovector |
| 27 | 1B | 108 | 06C | SD | Level 3 Interrupt Autovector |
| 28 | 1C | 112 | 070 | SD | Level 4 Interrupt Autovector |
| 29 | 1D | 116 | 074 | SD | Level 5 Interrupt Autovector |
| 30 | 1E | 120 | 078 | SD | Level 6 Interrupt Autovector |
| 31 | 1F | 124 | 07C | SD | Level 7 Interrupt Autovector |
| 32 to 47 | 20 to 2F | 128<br>191 | 080<br>0BF | SD | TRAP Instruction Vectors<br>— |
| 48 to 63* | 30 to 3F | 192<br>255 | 0C0<br>0FF | SD | (Unassigned, Reserved)<br>— |
| 64 to 255 | 40 to FF | 256<br>1023 | 100<br>3FF | SD | User Interrupt Vectors<br>— |

* Vector numbers 12, 13, 14, 16 to 23, and 48 to 63 are re-served for future enhancements. 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 data register against upper bounds (CHK) , and divide (DIV) instructions all can generate exceptions as part of their instruction execution. In addition, illegal instructions, word fetches from odd addresses, and privilege violations cause exceptions. Tracing behaves like a very high-priority internally-generated interrupt after each instruction execution.

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

The third step is save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer as shown in Figure 5.4. The program counter value stacked usually points to the next unexecuted instruction; however, for bus error and address error, the value stacked for the program counter is unpredictable, and may be incremented from the address of the instruction which caused the error. Additional information defining the current context is stacked for the bus error and address error exceptions.

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

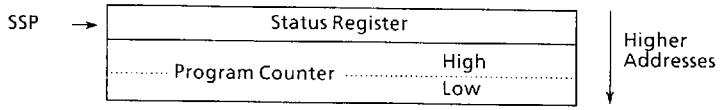



Figure 5.4 Exception Stack Order
(Groups 1 and 2)

5.2.4 Multiple Exceptions

These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped according to their occurrence and priority. The group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted and the exception processing to commence within two clock cycles.

The group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instruction. These exceptions allow the current instruction to execute to completion, but pre-empt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed) . The group 2 exception 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. Thesefore, 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 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 5.3 Exception Grouping and Priority

| Group | Exception | Processing |
|---|---|---|
| 0 | Reset<br>Address Error<br>Bus Error | Exception processing begins within two clock cycles |
| 1 | Trace<br>Interrupt<br>Illegal<br>Privilege | Exception processing begins before the next instruction |
| 2 | TRAP, TRAPV<br>CHK,<br>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 $\overline{\text{RESET}}$ signal is designed for system initiation and recovery from catastrophic failure. Any processing in progress at the time of the $\overline{\text{RESET}}$ is adorted and cannot be recoverd. 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 progrom counter. The power-up/restart code should be pointed to by the initial program counter.

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

5.3.2 Interrupts

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

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

If the priorty of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. A copy of the status register is saved, the privilege state is sent to the supervisor stack, tracing is suppressed, and the processor priority level is set to the level of the interrupt acknowledged. The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests an automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the program counter and status register on the supervisor stack. the saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine. A flowchart for the interrupt acknowledge sequence is given in Figure 5.5, a timing diagram is given in Figure 5.6, and the interrupt processing sequence is shown in Figure 5.7.

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 requestlevel changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

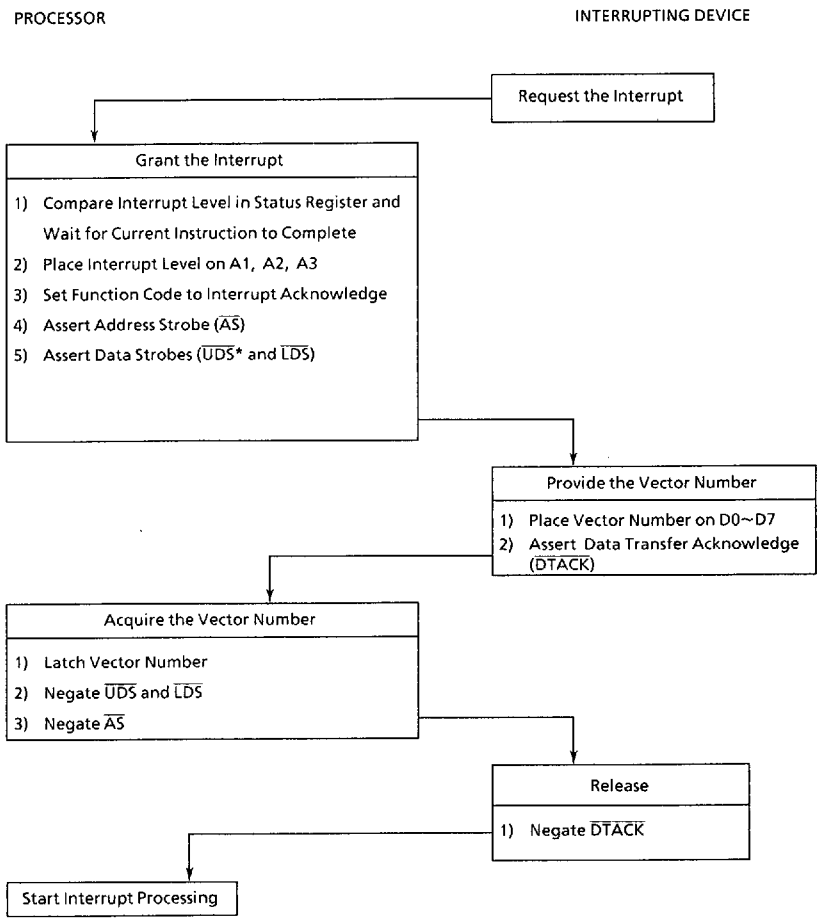



Figure5.5 Vector Acquisition Flowchart

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

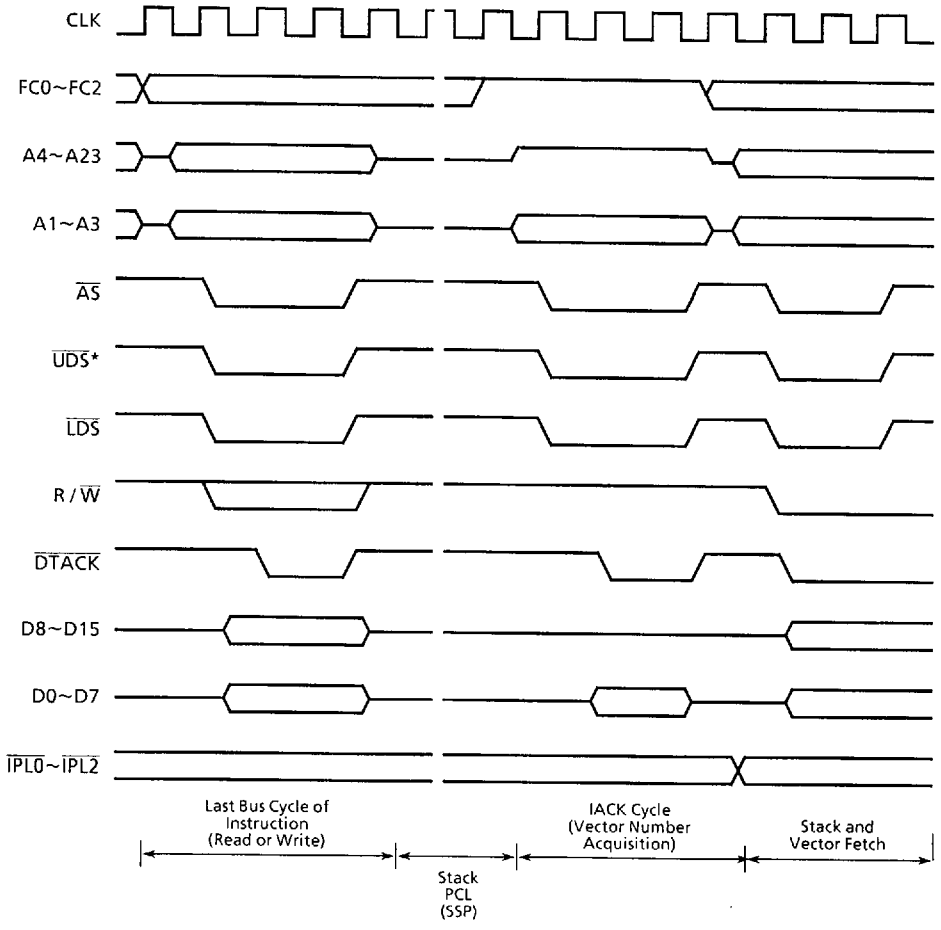
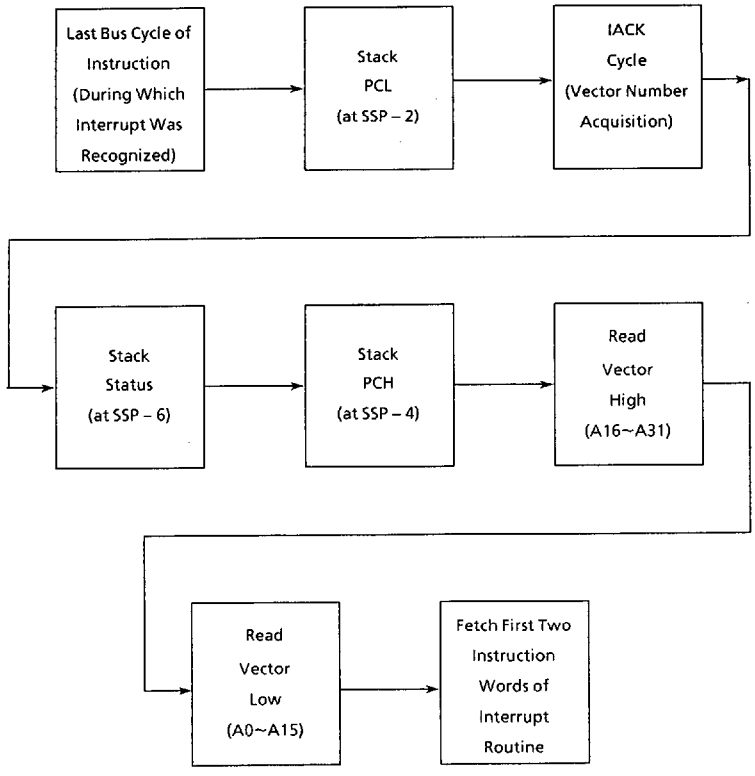



Figure5.6 Interrupt Acknowledge Cycle Timing Diagram

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




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

Figure5.7 Interrupt Processing Sequence

5.3.3 Uninitialized Interrupt

An interrupting device asserts $\overline{VPA}$ or provides an interrupt during an interrupt acknowledge cycle to the TMP68HC000. If the vector register has not been initialized, the responding TLCS-68000 Family peripheral will provide vector 15, the uninitialized, interrupt vector. This provides a uniform way to recover from a programming error.

5.3.4 Spurious Interrupt

If during the interrupt acknowledge cycle no device responds by asserting $\overline{DTACK}$ or $\overline{VPA}$, 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 instruction. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

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

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

5.3.6 Illegal and Unimplemented Instructions

"Illegal instruction" is the term used to refer to any of the word bit patterns which are not the bit pattern of the first word of a legal instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs. Three bit patterns will always force an illegal instruction trap on all TLCS-68000 Family compatible microprocessors. They are: $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for the system. The third pattern, $4AFC, is reserved for customer use.

Word patterns with bits 15~12 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 TMP68HC000 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 because an interrupt is taken, or the instruction is illegal or privileged, the trance 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. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing.

Exception processing for the bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus error exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. The value saved for the program counter is advanced by some amount,

one to five words beyond the address of the first word of the instruction which made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved; whether it was a read or a write, whether or not the processor was processing an instruction, and the classification displayed on the function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a group 2 exception;the processor is not processing an instruction if it is processing a group 0 or a group 1 exception. Figure 5.8 illustrates how this information is organized on the supervisor stack. Although this information is not sufficient in general to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in vector number two. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

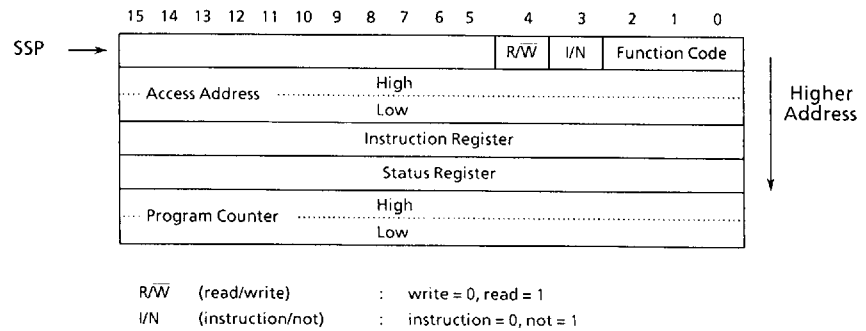



Figure 5.8 Exception Stack Order (Group 0)

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

### 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. The effect is much like an internally generated bus error, so that the bus cycle is aborted and the processor ceases whatever processing it is currently doing and begins exception processing. After the exception processing commences, the sequence is the same as that for bus error including the

information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted. As shown in Figure 5.9, an address error will execute a short bus cycle followed by exception processing.

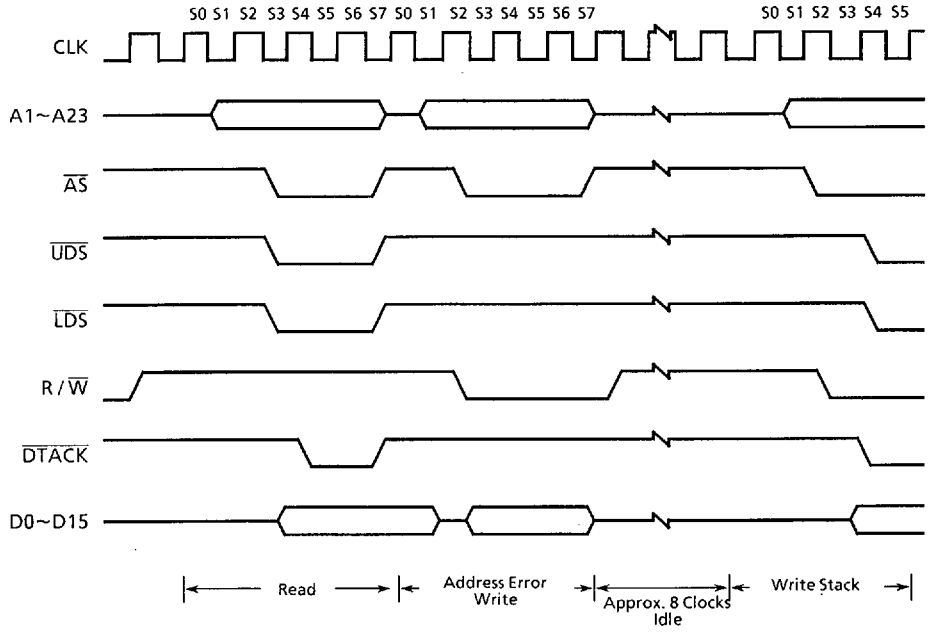



Figure5.9 Address Error Timing Diagram

## 6. INTERFACE WITH 6800 PERIPHERALS

Extensive line of 6800 peripherals are directly compatible with the TMP68HC000. Some of these devices that are particularly useful are:

| | |
|---|---|
| 6821 | Peripheral Interface Adapter |
| 6840 | Programmable Timer Module |
| 6843 | Floppy Disk Controller |
| 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 TMP68HC000, the processor modifies its bus cycle to meet the 6800 cycle requirements 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.

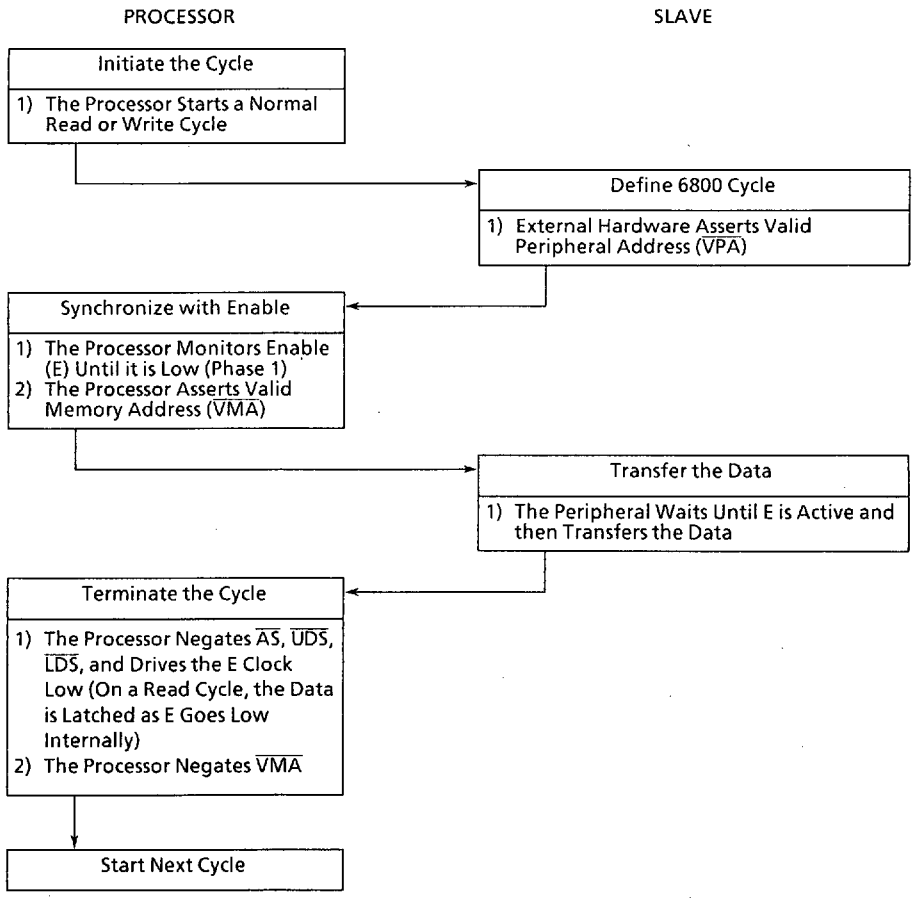



Figure 6.1 6800 Interfacing Flowchart

## 6.1 DATA TRANSFER OPERATION

Three signals on the processor provide the 6800 interface. They are: enable (E) , valid memory address ($\overline{VMA}$) , and valid peripheral address ($\overline{VPA}$) . Enable corresponds to the E or phase 2 signal in existing 6800 systems. The bus frequency is one tenth of the incoming TMP68HC000 clock frequency. The timing of E allows 1 megahertz peripherals to be used with 8 megahertz TMP68HC000s. Enable has a 60/40 duty cycle; that is, it is low for six input clocks and high for four input clocks. This duty cycle allows the processor to do successive $\overline{VPA}$ accesses on successive E pulses.

6800 cycle timing is given in Figures 6.2, 6.3, 8.7, and 8.8. At state zero (S0) in the cycle, the address bus is in the high-impedance state. A function code is asserted on the function code output lines. One-half clock later, in state 1, the address bus is released from the high-impedance state.

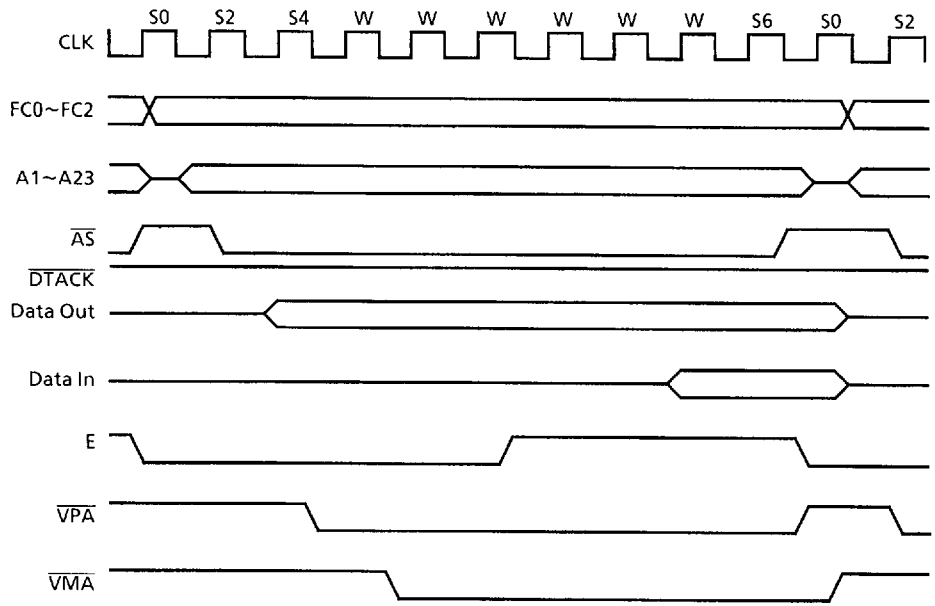



Figure 6.2 TMP68HC000 to 6800 Perpheral Timing – Best Case

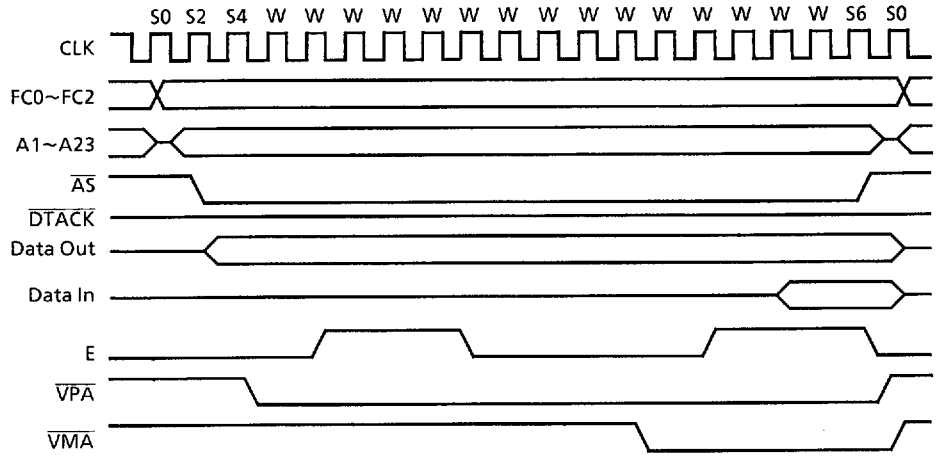



Figure 6.3 TMP68HC000 to 6800 Perpheral Timing – Worst Case

During state 2, the address strobe ($\overline{AS}$) is asserted to indicate that there is a valid address on the address bus. If the bus cycle is a read cycle, the upper and/or lower data strobes are also asserted in state 2.

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

The $\overline{VPA}$ input signals the processor that the address on the bus is the address of an 6800 device (or an area reserved for 6800 devices) and that bus should conform to the phase 2 transfer characteristics of the 6800 bus. Valid peripheral address is derived by decoding the address bus, conditioned by the address strobe. Chip select for the 6800 peripherals should be derived by decoding the address bus conditioned by $\overline{VMA}$.

After recognition of $\overline{VPA}$, the processor assures that the enable (E) is low, by waiting if necessary, and subsequently asserts $\overline{VMA}$. Valid memory address is then used as part of the chip select equation of the peripheral. This ensures that the 6800 peripherals are selected and deselected at the correct time. The peripheral now runs its cycle during the high portion of the E signal. Figures 6.2 and 6.3 depict the best and worst case 6800 cycle timing. This cycle length is dependent strictly upon when $\overline{VPA}$ is asserted in relationship to the E clock.

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

number of clock cycles within the following two extremes :

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

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

$\overline{DTACK}$ should not be asserted while $\overline{VPA}$ is asserted. Notice that the TMP68HC000 $\overline{VMA}$ is active low, contrasted with the active high 6800 VMA. This allows the processor to put its buses in the high-impedance state on DMA requests without inadvertently selecting the peripherals.

## 6.2 INTERRUPT INTERFACE OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector, the $\overline{VPA}$ is asserted, the TMP68HC000 will assert $\overline{VMA}$ and complete a normal 6800 read cycle as shown in Figure 6.4. The processor will then use an internally generated vector that is a function of the interrupt being serviced. This processor 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 TMP68HC000's normal vectored interrupt, the interrupt service routine can be located anywhere in the address space. This is due to the fact that while the vector numbers are fixed, the contents of the vector table entries are assigned by the user.

Since $\overline{VMA}$ is asserted during autovectoring, care should be taken to insure the 6800 peripheral address decoding prevents unintended accesses.

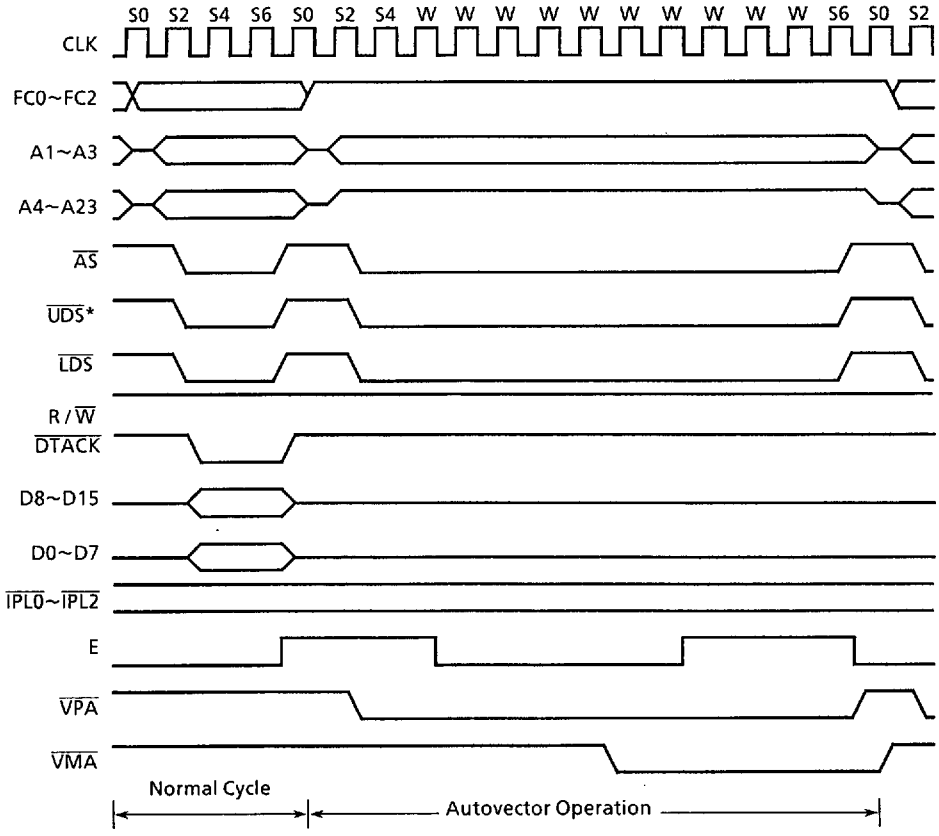



Figure 6.4 Autovector Operation Timing Diagram

* : Although $\overline{UDS}$ and $\overline{LDS}$ are asserted no data is read from the during the autovector cycle. The vector number is generated internally.

# 7. INSTRUCTION SET AND EXECUTION TIMES

## 7.1 INSTRUCTION SET

The following paragraphs provide information about the addressing categories and instruction set of the TMP68HC000.

### 7.1.1 Addressing Categories

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

Data : If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.

Memory : If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.

Alterable : If an effective address mode may be used to refer to alterable (writeable) operands, it is considered an alterable addressing effective address mode.

Control : If an effective address mode may be used to refer to memory operands without an associated size, it is considered a control addressing effective address mode.

These categories may be combined, so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable.

Table 7.1 shows the various categories to which each of the effective address modes belong. Table 7.2 is the instruction set summary.

Table 7.1 Effective Addressing Mode Categories

| Effective Address Modes | Mode | Register | Addressing Categories | | | |
|---|---|---|---|---|---|---|
| | | | Data | 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 5)

| Mnemonic | Description | Operation | Condition Codes | | | | |
|---|---|---|---|---|---|---|---|
| | | | X | N | Z | V | C |
| ABCD | Add Decimal with Extend | $(Destination)_{10} + (Source)_{10} + X \rightarrow Destination$ | * | U | * | U | * |
| ADD | Add Binary | (Destination) + (Source) →Destination | * | * | * | * | * |
| ADDA | Add address | (Destination) + (Source) →Destination | — | — | — | — | — |
| ADDI | Add Immediate | (Destination) + Immediate Data →Destination | * | * | * | * | * |
| ADDQ | Add Quick | (Destination) + Immediate Data →Destination | * | * | * | * | * |
| ADDX | Add Extended | (Destination) + (Source) + X →Destination | * | * | * | * | * |
| AND | AND Logical | (Destination)∧(Source) →Destination | — | * | * | 0 | 0 |
| ANDI | AND Immediate | (Destination)∧Immediate Data →Destination | — | * | * | 0 | 0 |
| ANDI to CCR | AND Immediate to Condition Codes | (Source)∧CCR→CCR | * | * | * | * | * |
| ANDI to SR | AND Immediate to Status Register | (Source)∧SR→SR | * | * | * | * | * |
| ASL,ASR | Arithmetic Shift | (Destination)Shifted by < count > →Destination | * | * | * | * | * |
| Bcc | Branch Conditionally | If cc then PC + d→PC | — | — | — | — | — |
| BCHG | Test a Bit and Change | ~(<bit number>)OF Destination→Z ~(<bit number>)OF Destination→ <bit number> OF Destination | — | — | * | — | — |
| BCLR | Test a Bit and Clear | ~(<bit number>)OF Destination→Z 0→<bit number> OF Destination | — | — | * | — | — |
| BRA | Branch Always | PC + d→PC | — | — | — | — | — |
| BSET | Test a Bit and Set | ~(<bit number>)OF Destination→Z 1→<bit number> OF Destination | — | — | * | — | — |

Table 7.2 Instruction Set (Sheet 2 of 5)

| Mnemonic | Description | Operation | Condition Codes | | | | |
|---|---|---|---|---|---|---|---|
| | | | X | N | Z | V | C |
| BSR | Branch to Subroutine | PC→ – (SP); PC + d→PC | — | — | — | — | — |
| BTST | Test a Bit | ~(<bit number>)OF Destination→Z | — | — | * | — | — |
| CHK | Check Register Against Bounds | If Dn <0 or DN> ( <ea> ) then TRAP | — | * | U | U | U |
| CLR | Clear an Operand | 0→Destination | — | 0 | 1 | 0 | 0 |
| CMP | Compare | (Destination) – (Source) | — | * | * | * | * |
| CMPA | Compare Address | (Destination) – (Source) | — | * | * | * | * |
| CMPI | Compare Immediate | (Destination) – Immediate Data | — | * | * | * | * |
| CMPM | Compare Memory | (Destination) – (Source) | — | * | * | * | * |
| DBcc | Test Condition, Decrement and Branch | If ~ cc then Dn – 1→Dn ; if Dn ≠ – 1 then PC + d→PC | — | — | — | — | — |
| DIVS | Signed Divide | (Destination) / (Source) →Destination | — | * | * | * | 0 |
| DIVU | Unsigned Divide | (Destination) / (Source) →Destination | — | * | * | * | 0 |
| EOR | Exclusive OR Logical | (Destination)⊕(Source) →Destination | — | * | * | 0 | 0 |
| EORI | Exclusive OR Immediate | (Destination)⊕Immediate Data →Destination | — | * | * | 0 | 0 |
| EORI to CCR | Exclusive OR Immediate to Condition Codes | (Source)⊕CCR→CCR | * | * | * | * | * |
| EORI to SR | Exclusive OR Immediate to Status Register | (Source)⊕SR→SR | * | * | * | * | * |
| EXG | Exchange Register | Xx↔Xy | — | — | — | — | — |
| EXT | Sign Extend | (Destination) Sign-Extended →Destination | — | * | * | 0 | 0 |
| JMP | Jump | Destination→PC | — | — | — | — | — |
| JSR | Jump to Subroutine | PC→ – (SP); Destination→PC | — | — | — | — | — |
| LEA | Load Effective Address | <ea>→An | — | — | — | — | — |
| LINK | Link and Allocate | An→ – (SP); SP→An; SP + Displacement→SP | — | — | — | — | — |

Table 7.2 Instruction Set (Sheet 3 of 5)

| Mnemonic | Description | Operation | Condition Codes | | | | |
|---|---|---|---|---|---|---|---|
| | | | X | N | Z | V | C |
| LSL,LSR | Logical Shift | (Destination) Shifted by <count> →Destination | * | * | * | 0 | * |
| MOVE | Move Data from Source to Destination | (Source)→Destination | — | * | * | 0 | 0 |
| MOVE to CCR | Move to Condition Code | (Source)→CCR | * | * | * | * | * |
| MOVE to SR | Move to the Status Register | (Source)→SR | * | * | * | * | * |
| MOVE from SR | Move from the Status Register | SR→Destination | — | — | — | — | — |
| MOVE USP | Move User Stack Pointer | USP→An; An→USP | — | — | — | — | — |
| MOVEA | Move Address | (Source)→Destination | — | — | — | — | — |
| MOVEM | Move Multiple Registers | Registers→Destination (Source)→Registers | — | — | — | — | — |
| MOVEP | Move Peripheral Data | (Source)→Destination | — | — | — | — | — |
| MOVEQ | Move Quick | Immediate Data→Destination | — | * | * | 0 | 0 |
| MULS | Signed Multiply | (Destination) × (Source) →Destination | — | * | * | 0 | 0 |
| MULU | Unsigned Multiply | (Destination) × (Source) →Destination | — | * | * | 0 | 0 |
| NBCD | Negate Decimal with Extend | $0 - (\text{Destination})_{10} - X$ →Destination | * | U | * | U | * |
| NEG | Negate | 0 – (Destination) →Destination | * | * | * | * | * |
| NEGX | Negate with Extend | 0 – (Destination) – X →Destination | * | * | * | * | * |
| NOP | No Operation | — | — | — | — | — | — |
| NOT | Logical Complement | ~(Destination)→Destination | — | * | * | 0 | 0 |
| OR | Inclusive OR Logical | (Destination)∨(Source) →Destination | — | * | * | 0 | 0 |
| ORI | Inclusive OR Immediate | (Destination)∨Immediate Data →Destination | — | * | * | 0 | 0 |
| ORI to CCR | Inclusive OR Immediate to Condition Codes | (Source) ∨ CCR→CCR | * | * | * | * | * |

Table 7.2 Instruction Set (Sheet 4 of 5)

| Mnemonic | Description | Operation | Condition Codes | | | | |
|---|---|---|---|---|---|---|---|
| | | | X | N | Z | V | C |
| ORI to SR | Inclusive OR Immediate to Status Register | (Source) $\vee$ SR→SR | * | * | * | * | * |
| PEA | Push Effective Address | <ea>→ – (SP) | — | — | — | — | — |
| RESET | Reset External Device | — | — | — | — | — | — |
| ROL,ROR | Rotate (Without Extend) | (Destination) Rotated by <count> →Destination | — | * | * | 0 | * |
| ROXL,ROXR | Rotate with Extend | (Destination) Rotated by <count> →Destination | * | * | * | 0 | * |
| RTE | Return from Exception | (SP) + →SR; (SP) + →PC | * | * | * | * | * |
| RTR | Return and Restore Condition Codes | (SP) + →CC; (SP) + →PC | * | * | * | * | * |
| RTS | Return from Subroutine | (SP) + →PC | — | — | — | — | — |
| SBCD | Subtract Decimal with Extend | $(Destination)_{10} - (Source)_{10} - X$ →Destination | * | U | * | U | * |
| Scc | Set According to Condition | If cc then 1's →Destination else 0's→Destination | — | — | — | — | — |
| STOP | Load Status Register and Stop | Immediate Data→SR;STOP | * | * | * | * | * |
| SUB | Subtract Binary | (Destination) – (Source) →Destination | * | * | * | * | * |
| SUBA | Subtract Address | (Destination) – (Source) →Destination | — | — | — | — | — |
| SUBI | Subtract Immediate | (Destination) – Immediate Data →Destination | * | * | * | * | * |
| SUBQ | Subtract Quick | (Destination) – Immediate Data →Destination | * | * | * | * | * |
| SUBX | Subtract with Extend | (Destination) – (source) – X →Destination | * | * | * | * | * |
| SWAP | Swap Register Halves | Register [31:16]↔Register [15:0] | — | * | * | 0 | 0 |
| TAS | Test and Set an Operand | (Destination) Tested→CC; 1→[7] OF Destination | — | * | * | 0 | 0 |

Table 7.2 Instruction Set (Sheet 5 of 5)

| Mnemonic | Description | Operation | Condition Codes | | | | |
|---|---|---|---|---|---|---|---|
| | | | X | N | Z | V | C |
| TRAP | Trap | PC→ – (SSP); SR→ – (SSP); (Vector)→PC | – | – | – | – | – |
| TRAPV | Trap on Overflow | If V then TRAP | – | – | – | – | – |
| TST | Test and Operand | (Destination) Tested→CC | – | * | * | 0 | 0 |
| UNLK | Unlink | An→SP; (SP) + →An | – | – | – | – | – |

→ : the left operand is moved to the right operand
↔ : the two operands are exchanged
+ : the operands are added
– : the right operand is subtracted from the left operand
* : the operands are multiplied
/ : the first operand is divided by the second operand
∧ : logical AND
∨ : logical OR
⊕ : logical exclusive OR
< : 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

* : affected
– : unaffected
0 : cleared
1 : set
U : undefined

7.1.2 Instruction Prefetch

The TMP68HC000 uses a two-word tightly-coupled instruction prefetch mechanism to enhance performance. This mechanism is described in terms of the microcode operations involved. If the execution of an instruction is defined to begin when the microroutine for that instruction is entered, some features of the prefetch mechanism can be described.

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

## 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 raed and write cycle times are four clock periods. Any wait states caused by a longer memory cycle must be added to the total instruction time. The number of bus read and write cycles for each instruction is also included with the timing data. This timing data is enclosed in parenthesis following the execution periods and is shown as (r/w) where r is the number of read cycles and w is the number of write cycles.

Note : The number of periods includes instruction fetch and all applicable operand fetches and stores.

7.2.1 Effective Address Operand Calculation Timing

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 |
|---|---|---|---|
| Dn | Register | | |
| An | Data Register Direct | 0 (0/0) | 0 (0/0) |
| | Address Register Direct | 0 (0/0) | 0 (0/0) |
| (An) | Memory | | |
| (An) + | Address Register Indirect | 4 (1/0) | 8 (2/0) |
| | Address Register Indirect with Postincrement | 4 (1/0) | 8 (2/0) |
| – (An) | Address Register Indirect with Predecrement | 6 (1/0) | 10 (2/0) |
| d16(PC) | Address Register Indirect with Displacement | 8 (2/0) | 12 (3/0) |
| d8(An, Xn)* | Address Register Indirect with Index | 10 (2/0) | 14 (3/0) |
| Abs.W | Absolute Short | 8 (2/0) | 12 (3/0) |
| Abs.L | Absolute Long | 12 (3/0) | 16 (4/0) |
| d16(PC) | Program Counter with Displacement | 8 (2/0) | 12 (3/0) |
| d8(PC, Xn)* | Program Counter with Index | 10 (2/0) | 14 (3/0) |
| #xxx | Immediate | 4 (1/0) | 8 (2/0) |

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

7.2.2 Move Instruction Execution Times

Table 7.4 and 7.5 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 7.4 Move Byte and Word Instruction Excecution Times

| Source | Destination | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Dn | An | (An) | (An) + | – (An) | d16(An) | d8(An,Xn)* | Abs.W | Abs.L |
| Dn | 4 (1/0) | 4 (1/0) | 8 (1/1) | 8 (1/1) | 8 (1/1) | 12 (2/1) | 14 (2/1) | 12 (2/1) | 16 (3/1) |
| An | 4 (1/0) | 4 (1/0) | 8 (1/1) | 8 (1/1) | 8 (1/1) | 12 (2/1) | 14 (2/1) | 12 (2/1) | 16 (3/1) |
| (An) | 8 (2/0) | 8 (2/0) | 12 (2/1) | 12 (2/1) | 12 (2/1) | 16 (3/1) | 18 (3/1) | 16 (3/1) | 20 (4/1) |
| (An) + | 8 (2/0) | 8 (2/0) | 12 (2/1) | 12 (2/1) | 12 (2/1) | 16 (3/1) | 18 (3/1) | 16 (3/1) | 20 (4/1) |
| – (An) | 10 (2/0) | 10 (2/0) | 14 (2/1) | 14 (2/1) | 14 (2/1) | 18 (3/1) | 20 (3/1) | 18 (3/1) | 22 (4/1) |
| d16(An) | 12 (3/0) | 12 (3/0) | 16 (3/1) | 16 (3/1) | 16 (3/1) | 20 (4/1) | 22 (4/1) | 20 (4/1) | 24 (5/1) |
| d8(An,Xn)* | 14 (3/0) | 14 (3/0) | 18 (3/1) | 18 (3/1) | 18 (3/1) | 22 (4/1) | 24 (4/1) | 22 (4/1) | 26 (5/1) |
| Abs.W | 12 (3/0) | 12 (3/0) | 16 (3/1) | 16 (3/1) | 16 (3/1) | 20 (4/1) | 22 (4/1) | 20 (4/1) | 24 (5/1) |
| Abs.L | 16 (4/0) | 16 (4/0) | 20 (4/1) | 20 (4/1) | 20 (4/1) | 24 (5/1) | 26 (5/1) | 24 (5/1) | 28 (6/1) |
| d16(PC) | 12 (3/0) | 12 (3/0) | 16 (3/1) | 16 (3/1) | 16 (3/1) | 20 (4/1) | 22 (4/1) | 20 (4/1) | 24 (5/1) |
| d8(PC,Xn)* | 14 (3/0) | 14 (3/0) | 18 (3/1) | 18 (3/1) | 18 (3/1) | 22 (4/1) | 24 (4/1) | 22 (4/1) | 26 (5/1) |
| #xxx | 8 (2/0) | 8 (2/0) | 12 (2/1) | 12 (2/1) | 12 (2/1) | 16 (3/1) | 18 (3/1) | 16 (3/1) | 20 (4/1) |

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

Table 7.5 Move Long Instruction Excecution Times

| Source | Destination | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Dn | An | (An) | (An) + | – (An) | d16(An) | d8(An,Xn)* | Abs.W | Abs.L |
| Dn | 4 (1/0) | 4 (1/0) | 12 (1/2) | 12 (1/2) | 12 (1/2) | 16 (2/2) | 18 (2/2) | 16 (2/2) | 20 (3/2) |
| An | 4 (1/0) | 4 (1/0) | 12 (1/2) | 12 (1/2) | 12 (1/2) | 16 (2/2) | 18 (2/2) | 16 (2/2) | 20 (3/2) |
| (An) | 12 (3/0) | 12 (3/0) | 20 (3/2) | 20 (3/2) | 20 (3/2) | 24 (4/2) | 26 (4/2) | 24 (4/2) | 28 (5/2) |
| (An) + | 12 (3/0) | 12 (3/0) | 20 (3/2) | 20 (3/2) | 20 (3/2) | 24 (4/2) | 26 (4/2) | 24 (4/2) | 28 (5/2) |
| – (An) | 14 (3/0) | 14 (3/0) | 22 (3/2) | 22 (3/2) | 22 (3/2) | 26 (4/2) | 28 (4/2) | 26 (4/2) | 30 (5/2) |
| d16(An) | 16 (4/0) | 16 (4/0) | 24 (4/2) | 24 (4/2) | 24 (4/2) | 28 (5/2) | 30 (5/2) | 28 (5/2) | 32 (6/2) |
| d8(An, Xn)* | 18 (4/0) | 18 (4/0) | 26 (4/2) | 26 (4/2) | 26 (4/2) | 30 (5/2) | 32 (5/2) | 30 (5/2) | 34 (6/2) |
| Abs.W | 16 (4/0) | 16 (4/0) | 24 (4/2) | 24 (4/2) | 24 (4/2) | 28 (5/2) | 30 (5/2) | 28 (5/2) | 32 (6/2) |
| Abs.L | 20 (5/0) | 20 (5/0) | 28 (5/2) | 28 (5/2) | 28 (5/2) | 32 (6/2) | 34 (6/2) | 32 (6/2) | 36 (7/2) |
| d16(PC) | 16 (4/0) | 16 (4/0) | 24 (4/1) | 24 (4/2) | 24 (4/2) | 28 (5/2) | 30 (5/2) | 28 (5/2) | 32 (5/2) |
| d8(PC, Xn)* | 18 (4/0) | 18 (4/0) | 26 (4/2) | 26 (4/2) | 26 (4/2) | 30 (5/2) | 32 (5/2) | 30 (5/2) | 34 (6/2) |
| #xxx | 12 (3/0) | 12 (3/0) | 20 (3/2) | 20 (3/2) | 20 (3/2) | 24 (4/2) | 26 (4/2) | 24 (4/2) | 28 (5/2) |

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

7.2.3 Standard Instruction Execution Times

The number of clock periods shown in Table 7.6 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 7.6 the headings have the following maenings:

An = address register operand
Dn = data register operand
ea = an operand specified by an effective address
M = memory effective address operand

Table 7.6 Standard Instruction Execution Times

| Instruction | Size | op<ea>, An^ | op<ea>, Dn | opDn, <M> |
|---|---|---|---|---|
| ADD | Byte, Word | 8 (1/0) + | 4 (1/0) + | 8 (1/1) + |
| | Long word | 6 (1/0) + ** | 6 (1/0) + ** | 12 (1/2) + |
| AND | Byte, Word | – | 4 (1/0) + | 8 (1/1) + |
| | Long word | – | 6 (1/0) + ** | 12 (1/2) + |
| CMP | Byte, Word | 6 (1/0) + | 4 (1/0) + | – |
| | Long word | 6 (1/0) + | 6 (1/0) + | – |
| DIVS | – | – | 158 (1/0) + * | – |
| DIVU | – | – | 140 (1/0) + * | – |
| EOR | Byte, Word | – | 4 (1/0) *** | 8 (1/1) + |
| | Long word | – | 8 (1/0) *** | 12 (1/2) + |
| MULS | – | – | 70 (1/0) + * | – |
| MULU | – | – | 70 (1/0) + * | – |
| OR | Byte, Word | – | 4 (1/0) + | 8 (1/1) + |
| | Long word | – | 6 (1/0) + ** | 12 (1/2) + |
| SUB | Byte, Word | 8(1/0) + | 4 (1/0) + | 8 (1/1) + |
| | Long word | 6(1/0) + ** | 6 (1/0) + ** | 12 (1/2) + |

\+ : add effective address calculation time
^ : word or long word only
* : indicates maximum value
** : The base time of six clock periods is increased to eight 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 TMP68HC000 provides less than 10% difference between the best and worst case timings.
MULS, MULU – The multiply algorithm requires 38+2n clocks where n is defined as:

MULU : n = the number of ones in the <ea>
MULS : n = concatanate the <ea> with a zero as the LSB; n is The resultant number of 10 or 01 pattern in the 17-bit source: i.e., worst case happens when the source is $5555.

### 7.2.4 Immediate Instruction Execution Times

The number of clock periods shown in Table 7.7 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 7.7, the headings have the following meanings:

# = immediate operand
Dn = data register operand
An = address register operand
M = memory operand
SR = status register

Table 7.7 Immediate Instruction Execution Times

| Instruction | Size | op #, Dn | op #, An | op #, M |
|---|---|---|---|---|
| ADDI | Byte Word | 8 (2/0) | — | 12 (2/1) + |
| | Long word | 16 (3/0) | — | 20 (3/2) + |
| ADDQ | Byte Word | 4 (1/0) | 8 (1/0) * | 8 (1/1) + |
| | Long word | 8 (1/0) | 8 (1/0) | 12 (1/2) + |
| ANDI | Byte Word | 8 (2/0) | — | 12 (2/1) + |
| | Long word | 16 (3/0) | — | 20 (3/1) + |
| CMPI | Byte Word | 8 (2/0) | — | 8 (2/0) + |
| | Long word | 14 (3/0) | — | 12 (3/0) + |
| EORI | Byte Word | 8 (2/0) | — | 12 (2/1) + |
| | Long word | 16 (3/0) | — | 20 (3/2) + |
| MOVEQ | Long word | 4 (1/0) | — | — |
| ORI | Byte Word | 8 (2/0) | — | 12 (2/1) + |
| | Long word | 16 (3/0) | — | 20 (3/2) + |
| SUBI | Byte Word | 8 (2/0) | — | 12 (2/1) + |
| | Long word | 16 (3/0) | — | 20 (3/2) + |
| SUBQ | Byte Word | 4 (1/0) | 8 (1/0) * | 8 (1/1) + |
| | Long word | 8 (1/0) | 8 (1/0) | 12 (1/2) + |

+ : add effective address calculation time

* : word only

### 7.2.5 Single Operand Instruction Execution Times

Table 7.8 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 7.8 Single Operand Instruction Execution Times

| Instruction | Size | Register | Memory |
|---|---|---|---|
| CLR | Byte, Word | 4 (1/0) | 8 (1/1) + |
| | Long word | 6 (1/0) | 12 (1/2) + |
| NBCD | Byte | 6 (1/0) | 8 (1/1) + |
| NEG | Byte, Word | 4 (1/0) | 8 (1/1) + |
| | Long word | 6 (1/0) | 12 (1/2) + |
| NEGX | Byte, Word | 4 (1/0) | 8 (1/1) + |
| | Long word | 6 (1/0) | 12 (1/2) + |
| NOT | Byte, Word | 4 (1/0) | 8 (1/1) + |
| | Long word | 6 (1/0) | 12 (1/2) + |
| Scc | Byte, False | 4 (1/0) | 8 (1/1) + |
| | Byte, True | 6 (1/0) | 8 (1/1) + |
| TAS | Byte | 4 (1/0) | 10 (1/1) + |
| TST | Byte, Word | 4 (1/0) | 4 (1/0) + |
| | Long word | 4 (1/0) | 4 (1/0) + |

+ : add effective address calculation time

### 7.2.6 Shift/Rotate Instruction Execution Times

Table 7.9 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 7.9 Shift/Rotate Instruction Execution Times

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

+ : add effective address calculation time

n : the shift or rotate count

### 7.2.7 Bit Manipulation Instruction Execution Times

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

Table 7.10 Bit Manipulation Instruction Execution Times

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

\+ : add effective address calculation time

* : indicates maximum value

### 7.2.8 Conditional Instruction Execution Times

Table 7.11 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 7.11 Conditional Instruction Execution Times

| Instruction | Displacement | Branch Taken | Branch Not Taken |
|---|---|---|---|
| Bcc | Byte | 10 (2/0) | 8 (1/0) |
| | Word | 10 (2/0) | 12 (2/0) |
| BRA | Byte | 10 (2/0) | — |
| | Word | 10 (2/0) | — |
| BSR | Byte | 18 (2/2) | — |
| | Word | 18 (2/2) | — |
| DBcc | CC true | — | 12 (2/0) |
| | CC false | 10 (2/0) | 14 (3/0) |

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

Table 7.12 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 7.12 JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

| Instr | Size | (An) | (An) + | – (An) | d16 (An) | d8 (An, Xn) * | Abs.W | Abs.L | d16 (PC) | d8 (PC, Xn) * |
|---|---|---|---|---|---|---|---|---|---|---|
| JMP | — | 8 (2/0) | — | — | 10 (2/0) | 14 (3/0) | 10 (2/0) | 12 (3/0) | 10 (2/0) | 14 (3/0) |
| JSR | — | 16 (2/2) | — | — | 18 (2/2) | 22 (2/2) | 18 (2/2) | 20 (3/2) | 18 (2/2) | 22 (2/2) |
| LEA | — | 4 (1/0) | — | — | 8 (2/0) | 12 (2/0) | 8 (2/2) | 12 (3/0) | 8 (2/0) | 12 (2/0) |
| PEA | — | 12 (1/2) | — | — | 16 (2/2) | 20 (2/2) | 16 (2/2) | 20 (3/2) | 16 (2/2) | 20 (2/2) |
| MOVEM M→R | Word | 12 + 4n (3 + n/0) | 12 + 4n (3 + n/0) | — | 16 + 4n (4 + n/0) | 18 + 4n (4 + n/0) | 16 + 4n (4 + 2n/0) | 20 + 4n (5 + n/0) | 16 + 4n (4 + n/0) | 18 + 4n (4 + n/0) |
| | Long word | 12 + 8n (3 + 2n/0) | 12 + 8n (3 + 2n/0) | — | 16 + 8n (4 + 2n/0) | 18 + 8n (4 + 2n/0) | 16 + 8n (4 + 2n/0) | 20 + 8n (5 + 2n/0) | 16 + 8n (4 + 2n/0) | 18 + 8n (4 + 2n/0) |
| MOVEM R→M | Word | 8 + 4n (2/n) | — | 8 + 4n (2/n) | 12 + 4n (3/n) | 14 + 4n (3/n) | 12 + 4n (3/n) | 16 + 4n (4/n) | — | — |
| | Long word | 8 + 8n (2/2n) | — | 8 + 8n (2/2n) | 12 + 8n (3/2n) | 14 + 8n (3/2n) | 12 + 8n (3/2n) | 16 + 8n (4/2n) | — | — |

n : the number of registers to move

* : the size of the index register (Xn) does not affect the instruction's execution time

### 7.2.10 Multi-Precision Instruction Execution Times

Table 7.13 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 7.13, the headings have the following meaning:

Dn = data register operand

M = memory operand.

Table 7.13 Multi-Precision Instruction Execution Times

| Instruction | Size | op Dn, Dn | op M, M |
|---|---|---|---|
| ADDX | Byte, Word | 4 (1/0) | 18 (3/1) |
| | Long word | 8 (1/0) | 30 (5/2) |
| CMPM | Byte, Word | — | 12 (3/0) |
| | Long word | — | 20 (5/0) |
| SUBX | Byte, Word | 4 (1/0) | 18 (3/1) |
| | Long word | 8 (1/0) | 30 (5/2) |
| ABCD | Byte | 6 (1/0) | 18 (3/1) |
| SBCD | Byte | 6 (1/0) | 18 (3/1) |

### 7.2.11 Miscellaneous Instruction Execution Times

Table 7.14 and 7.15 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 7.14 Miscellaneous Instruction Execution Times

| Instruction | Size | Register | Memory |
|---|---|---|---|
| ANDI to CCR | Byte | 20 (3/0) | — |
| ANDI to SR | Word | 20 (3/0) | — |
| CHK | — | 10 (1/0) + | — |
| EORI to CCR | Byte | 20 (3/0) | — |
| EORI to SR | Word | 20 (3/0) | — |
| ORI to CCR | Byte | 20 (3/0) | — |
| ORI to SR | Word | 20 (3/0) | — |
| MOVE from SR | — | 6 (1/0) | 8 (1/1) + |
| MOVE to CCR | — | 12 (1/0) | 12 (1/0) + |
| MOVE to SR | — | 12 (1/0) | 12 (1/0) + |
| EXG | — | 6 (1/0) | — |
| EXT | Word | 4 (1/0) | — |
| | Long word | 4 (1/0) | — |
| LINK | — | 16 (2/2) | — |
| MOVE from USP | — | 4 (1/0) | — |
| MOVE to USP | — | 4 (1/0) | — |
| NOP | — | 4 (1/0) | — |
| RESET | — | 132 (1/0) | — |
| RTE | — | 20 (5/0) | — |
| RTR | — | 20 (5/0) | — |
| RTS | — | 16 (4/0) | — |
| STOP | — | 4 (0/0) | — |
| SWAP | — | 4 (1/0) | — |
| TRAPV (No Trap) | — | 4 (1/0) | — |
| UNLK | — | 12 (3/0) | — |

+ : add effective address calculation time

Table 7.15 Move Peripheral Instruction Execution Times

| Instruction | Size | Register→Memory | Memory→Register |
|---|---|---|---|
| MOVEP | Word | 16 (2/2) | 16 (4/0) |
| | Long word | 24 (2/4) | 24 (6/0) |

### 7.2.12 Exception Processing Execution Times

Table 7.16 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first two instruction words of the handler routine. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 7.16 Exception Processing Execution Times

| Exception | Periods |
|---|---|
| Address Error | 50 (4/7) |
| Bus Error | 50 (4/7) |
| CHK Instruction (Trap Taken) | 44 (5/3) + |
| Divide by Zero | 42 (5/3) |
| Illegal Instruction | 34 (4/3) |
| Interrupt | 44 (5/3)* |
| Privilege Violation | 34 (4/3) |
| $\overline{\text{RESET}}$** | 40 (6/0) |
| Trace | 34 (4/3) |
| TRAP Instruction | 38 (4/3) |
| TRAPV Instruction (Trap Taken) | 34 (4/3) |

+ : add effective address calculation time

* : The interrupt acknowledge cycle is assumed to take four clock periods.

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

# 8. ELECTRICAL SPECIFICATIONS

This section contains electrical specifications and associated timing information for the TMP68HC000.

## 8.1 MAXIMUM RATINGS

| Rating | Symble | Value | Unit |
|---|---|---|---|
| | | TMP68HC000 | |
| Supply Voltage | Vcc | −0.3~+6.5 | V |
| Input Voltage | Vin | −0.3~+6.5 | V |
| Operating Temperature Range | Ta | 0~+70 | °C |
| Storage Temperature | Tstg | −55~+150 | °C |

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

## 8.2 DC ELECTRICAL CHARACTERISTICS

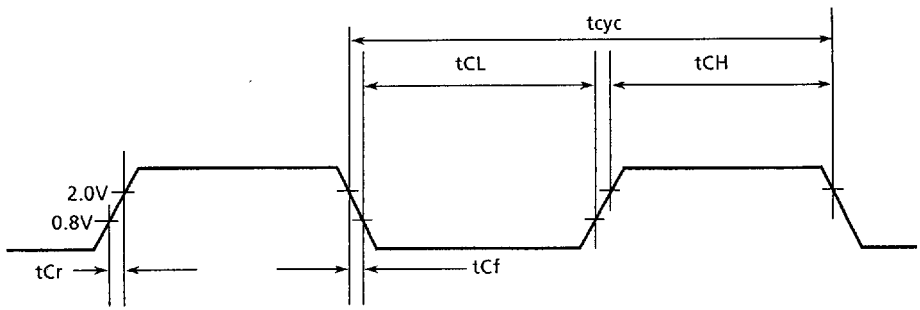
(Vcc = 5.0V ± 5%, GND = 0V, Ta = 0℃ ~ +70℃ ; see Figures 8.1)

| Characteristic | | Symbol | TMP68HC000 | | Unit |
|---|---|---|---|---|---|
| | | | Min | Max | |
| Input High Voltage | | $V_{IH}$ | 2.0 | $V_{CC}$ | V |
| Input Low Voltage | | $V_{IL}$ | GND-0.3 | 0.8 | V |
| Input Leakage Current (5.25V) | $\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$,CLK, $\overline{IPL0}$~$\overline{IPL2}$, $\overline{VPA}$ $\overline{HALT}$, $\overline{RESET}$ | $I_{IN}$ | – – – | 2.5 2.5 20 | μA |
| Three-State (Off State) Input Current | (2.4V/0.4V) $\overline{AS}$, A1~A23, D0-D15, $\overline{FC0}$~$\overline{FC2}$, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$ | $I_{TSI}$ | – – – | 20 20 20 | μA |
| Output High Voltage ($I_{OH}$ = -400μA) | E* E, $\overline{AS}$, A1~A23, $\overline{BG}$, D0~D15, $\overline{FC0}$-$\overline{FC2}$, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$ | $V_{OH}$ | – $V_{CC}$-0.75 $V_{CC}$-0.75 $V_{CC}$-0.75 $V_{CC}$-0.75 | – – – – | V |
| Output Low Voltage ($I_{OL}$ = 1.6mA) ($I_{OL}$ = 3.2mA) ($I_{OL}$ = 5.0mA) ($I_{OL}$ = 5.3mA) | $\overline{HALT}$ A1~A23, $\overline{BG}$, FC0~FC2 $\overline{RESET}$ E, $\overline{AS}$, D0~D15, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$ | $V_{OL}$ | – – – – – | 0.5 0.5 0.5 0.5 0.5 | V |
| Current Dissipation*** | f = 8MHz f = 10MHz f = 12.5MHz f = 16.67MHz | $I_D$ | – – – – | 25 30 35 50 | mA |
| Power Dissipation | f = 8MHz f = 10MHz f = 12.5MHz f = 16.67MHz | $P_D$ | – – – – | 0.13 0.16 0.19 0.26 | W |
| Capacitance (Vin = 0V, Ta = 25℃ : Frequency = 1MHz)** | | $C_{IN}$ | – | 20.0 | pF |
| Load Capacitance | $\overline{HALT}$ All Others | $C_L$ | – – | 70 130 | pF |

* : With external pullup resistor of 1.1kΩ.

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

## 8.3 AC ELECTRICAL SPECIFICATIONS – CLOCK TIMING




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

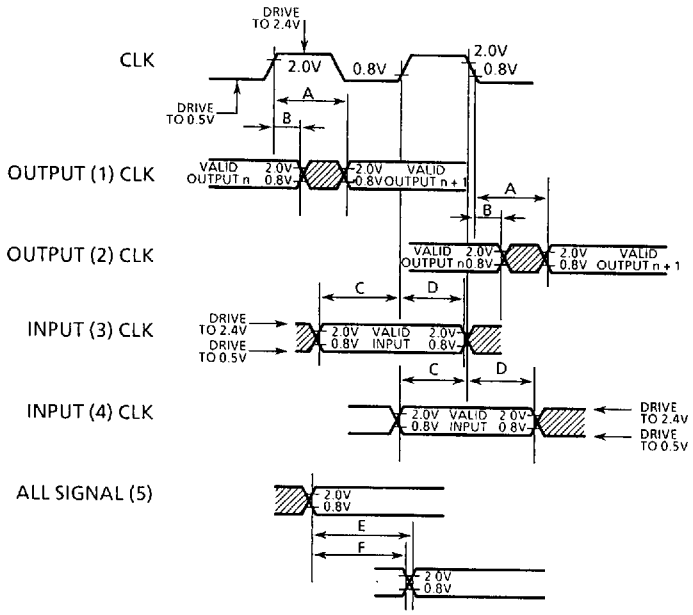
Figure 8.1 Clock Input Timing Diagram

## 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 (1/4)

($V_{CC}$ = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.3 and 8.4)

| Num. | Characteristic | Symbol | 10MHz | | 12.5MHz | | 16.67MHz | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | |
| 1 | Clock Period | tCYC | 100 | 250 | 80 | 250 | 60 | 125 | ns |
| 2 | Clock Width Low | tCL | 45 | 125 | 35 | 125 | 27 | 62.5 | ns |
| 3 | Clock Width High | tCH | 45 | 125 | 35 | 125 | 27 | 62.5 | ns |
| 4 | Clock Fall Time | tCf | – | 10 | – | 5 | – | 5 | ns |
| 5 | Clock Rise Time | tCr | – | 10 | – | 5 | – | 5 | ns |
| 6 | Clock Low to Address Valid | tCLAV | – | 50 | – | 50 | – | 30 | ns |
| 6A | Clock High to FC Valid | tCHFCV | – | 50 | – | 45 | 0 | 30 | ns |
| 7 | Clock High to Address, Data Bus High Impedance (Maximum) | tCHADZ | – | 70 | – | 60 | – | 50 | ns |
| 8 | Clock High to Address, FC Invalid (Minimum) | tCHAFI | 0 | – | 0 | – | 0 | – | ns |
| 9[1] | Clock High to $\overline{AS}$, $\overline{DS}$ Low | tCHSL | 3 | 50 | 3 | 40 | 3 | 30 | ns |
| 11[2] | Address Valid to $\overline{AS}$, $\overline{DS}$ Low (Read) / $\overline{AS}$ Low (Write) | tAVSL | 20 | – | 15 | – | 15 | – | ns |
| 11A[2] | FC Valid to $\overline{AS}$, $\overline{DS}$ Low (Read) / $\overline{AS}$ Low (Write) | tFCVSL | 70 | – | 60 | – | 45 | – | ns |
| 12[1] | Clock Low to $\overline{AS}$, $\overline{DS}$ High | tCLSH | – | 50 | – | 40 | 3 | 30 | ns |
| 13[2] | $\overline{AS}$, $\overline{DS}$ High to Address / FC Invalid | tSHAFI | 30 | – | 20 | – | 15 | – | ns |
| 14[2] | $\overline{AS}$, $\overline{DS}$ Width Low (Read) / $\overline{AS}$ Low (Write) | tSL | 195 | – | 160 | – | 120 | – | ns |
| 14A | $\overline{DS}$ Width Low (Write) | tDSL | 95 | – | 80 | – | 60 | – | ns |
| 15[2] | $\overline{AS}$, $\overline{DS}$ Width High | tSH | 105 | – | 65 | – | 60 | – | ns |

## 8.5 AC ELECTRICAL SPECIFICATIONS – READ AND WRITE CYCLES (2/4)

($V_{CC}$ = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.3 and 8.4)

| Num. | Characteristic | Symbol | 10MHz | | 12.5MHz | | 16.67MHz | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | |
| 16 | Clock High to Control Bus High Impedance | tCHCZ | – | 70 | – | 60 | – | 50 | ns |
| 17[2] | $\overline{AS}$, $\overline{DS}$, High to R / $\overline{W}$ High (Read) | tSHRH | 30 | – | 20 | – | 15 | – | ns |
| 18[1] | Clock High to R / $\overline{W}$ High | tCHRH | 0 | 45 | 0 | 40 | 0 | 30 | ns |
| 20[1] | Clock High to R / $\overline{W}$ Low (Write) | tCHRL | 0 | 45 | 0 | 40 | 0 | 30 | ns |
| 20A[2,6] | $\overline{AS}$ Low to R / $\overline{W}$ Valid (Write) | tASRV | – | 10 | – | 10 | – | 10 | ns |
| 21[2] | Address Valid to R / $\overline{W}$ Low (Write) | tAVRL | 0 | – | 0 | – | 0 | – | ns |
| 21A[2] | FC Valid to R / $\overline{W}$ Low (Write) | tFCVRL | 50 | – | 30 | – | 30 | – | ns |
| 22[2] | R / $\overline{W}$ Low to $\overline{DS}$ Low (Write) | tRLSL | 50 | – | 30 | – | 30 | – | ns |
| 23 | CLock Low to Data Out Valid (Write) | tCLDO | – | 50 | – | 50 | – | 30 | ns |
| 25[2] | $\overline{AS}$, $\overline{DS}$ High to Data Out Invalid (Write) | tSHDOI | 30 | – | 20 | – | 15 | – | ns |
| 26[2] | Data Out Valid to $\overline{DS}$Low (Write) | tDOSL | 30 | – | 20 | – | 15 | – | ns |
| 27[5] | Data in to Clock Low (Setup Time on Read) | tDICL | 10 | – | 10 | – | 5 | – | ns |
| 28[2] | $\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High (Asynchronous Hold) | tSHDAH | 0 | 190 | 0 | 150 | 0 | 110 | ns |
| 29 | ($\overline{AS}$, $\overline{DS}$ High to Data-In Invalid (Hold Time on Read) | tSHDII | 0 | – | 0 | – | 0 | – | ns |

## 8.5 AC ELECTRICAL SPECIFICATIONS – READ AND WRITE CYCLES (3/4)

($V_{CC}$ = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.3 and 8.4)

| Num. | Characteristic | Symbol | 10MHz | | 12.5MHz | | 16.67MHz | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min. | Max. | Min. | Max. | Min. | Max. | |
| 30 | $\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High | tSHBEH | 0 | – | 0 | – | 0 | – | ns |
| 31[2,5] | $\overline{DTACK}$ Low to Data In (Setup Time) | tDALDI | – | 65 | – | 50 | – | 50 | ns |
| 32 | $\overline{HALT}$ and $\overline{RESET}$ Input Transition | tRHr, f | 0 | 200 | 0 | 200 | – | 150 | ns |
| 33 | Clock High to $\overline{BG}$ Low | tCHGL | – | 50 | – | 40 | 0 | 30 | ns |
| 34 | CLock High to $\overline{BG}$ Low | tCHGH | – | 50 | – | 40 | 0 | 30 | ns |
| 35 | $\overline{BR}$ Low to $\overline{BG}$ Low | tBRLGL | 1.5 | 3.5 | 1.5 | 3.5 | 1.5 | 3.5 | Clk. Per. |
| 36[7] | $\overline{BR}$ High to $\overline{BG}$ Low | tBRHGH | 1.5 | 3.5 | 1.5 | 3.5 | 1.5 | 3.5 | Clk. Per. |
| 37 | $\overline{BGACK}$ Low to $\overline{BG}$ Low | tGALGH | 1.5 | 3.5 | 1.5 | 3.5 | 1.5 | 3.5 | Clk. Per. |
| 37A[8] | $\overline{BGACK}$ Low to $\overline{BG}$ Low | tGALBRH | 20 | 1.5 Clocks | 20 | 1.5 Clocks | 10 | 1.5 Clocks | ns |
| 38 | $\overline{BG}$ Width High | tGLZ | – | 70 | – | 60 | – | 50 | ns |
| 39 | $\overline{BG}$ Width High | tGH | 1.5 | – | 1.5 | – | 1.5 | – | Clk. Per. |
| 40 | Clock Low to $\overline{VMA}$ Low | tCLVML | – | 70 | – | 70 | – | 50 | ns |
| 41 | Clock Low to E Transition | tCLET | – | 45 | – | 35 | – | 35 | ns |
| 42 | E Output Rise and Fall Time | tEr, f | – | 15 | – | 15 | – | 15 | ns |
| 43 | $\overline{VMA}$ Low to E High | tVMLEH | 150 | – | 90 | – | 80 | – | ns |
| 44 | $\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High | tSHVPH | 0 | 90 | 0 | 70 | 0 | 50 | ns |
| 45 | E Low to Control, Address Bus Invalid (Address Hold Time) | tELCAI | 10 | – | 10 | – | 10 | – | ns |

## 8.5 AC ELECTRICAL SPECIFICATIONS – READ AND WRITE CYCLES (4/4)

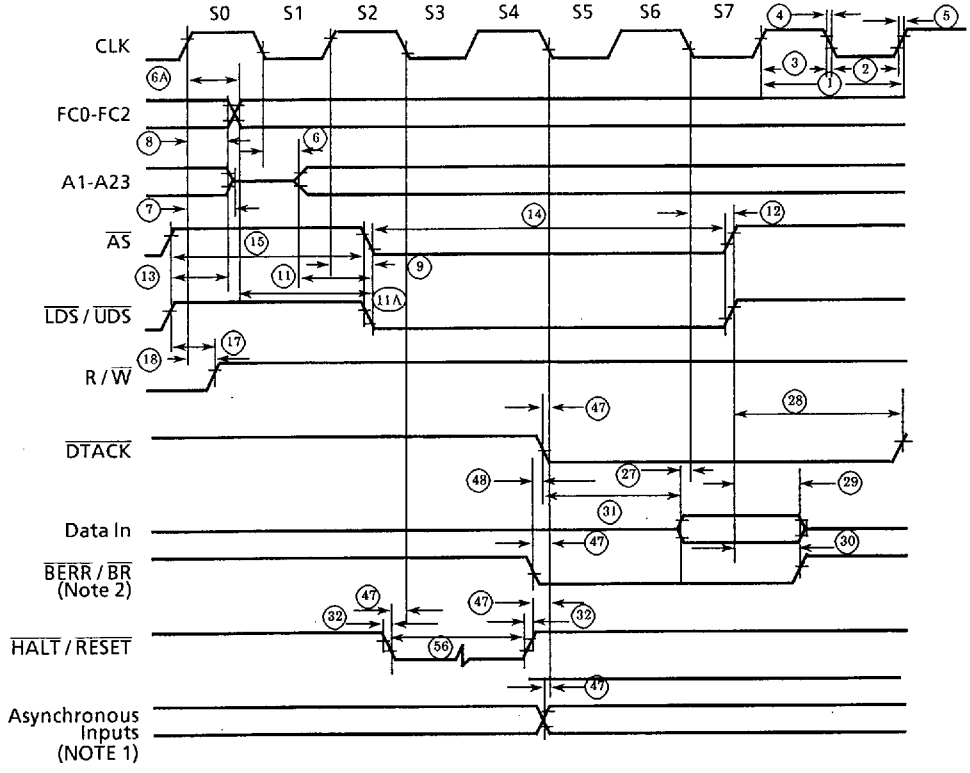
($V_{CC}$ = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.3 and 8.4)

| Num. | Characteristic | Symbol | 10MHz | | 12.5MHz | | 16.67MHz | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min. | Max. | Min. | Max. | Min. | Max. | |
| 46 | $\overline{BGACK}$ Width Low | tGAL | 1.5 | – | 1.5 | – | 1.5 | – | Clk. Per. |
| 47[5] | Asynchronous Input Setup Time | tASI | 10 | – | 10 | – | 5 | – | ns |
| 48[2,3] | $\overline{BERR}$ Low to $\overline{DTACK}$ Low | tBELDAL | 20 | – | 20 | – | 10 | – | ns |
| 49[9] | $\overline{AS}$, $\overline{DS}$ High to E Low | tSHEL | −55 | 55 | −45 | 45 | −35 | 35 | ns |
| 50 | E Width High | tEH | 350 | – | 280 | – | 220 | – | ns |
| 51 | E Width High | tEL | 550 | – | 440 | – | 340 | – | ns |
| 53 | Clock High to Data Out Invalid | tCHDOI | 0 | – | 0 | – | 0 | – | ns |
| 54 | E Low to Data Out Invalid | tELDOI | 20 | – | 15 | – | 10 | – | ns |
| 55 | R / $\overline{W}$ to Data Bus Driven | tRLDBD | 20 | – | 10 | – | 0 | – | ns |
| 56[4] | $\overline{HALT}$ / $\overline{RESET}$ Pulse Width | tHRPW | 10 | – | 10 | – | 10 | – | Clk. Per. |
| 57 | $\overline{BGACK}$ High to Control Bus Driven | tGASD | 1.5 | – | 1.5 | – | 1.5 | – | Clk. Per. |
| 58[7] | $\overline{BG}$ High to Control Bus Driven | tRHSD | 1.5 | – | 1.5 | – | 1.5 | – | Clk. Per. |

Note :

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

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.

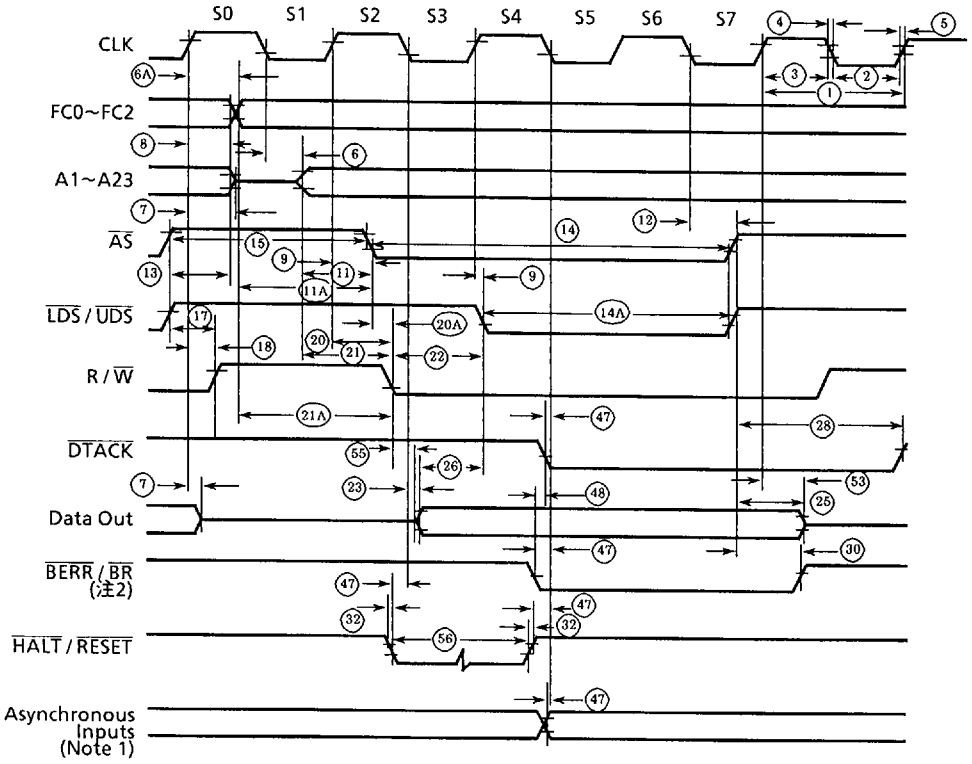



Note :

1. Setup time for the asynchronous inputs $\overline{IPL0}$~$\overline{IPL2}$, and $\overline{VPA}$ guarantees their recognition at the next falling edge of the clock.
2. $\overline{BR}$ 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 2.0 volts, unless otherwise noted. The voltage swing through this range should start outside and pass through the the range such that the rise or fall will be linear between 0.8 volt and 2.0 volts.

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.




Note :

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.
   The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 volt 2.0 volts.
2. Because of loading variation, R / $\overline{W}$ may be valid after $\overline{AS}$ even through both are initiated by the rising edge of S2 (Specification 20A).

Figure 8.4 Write Cycle Timing Diagram

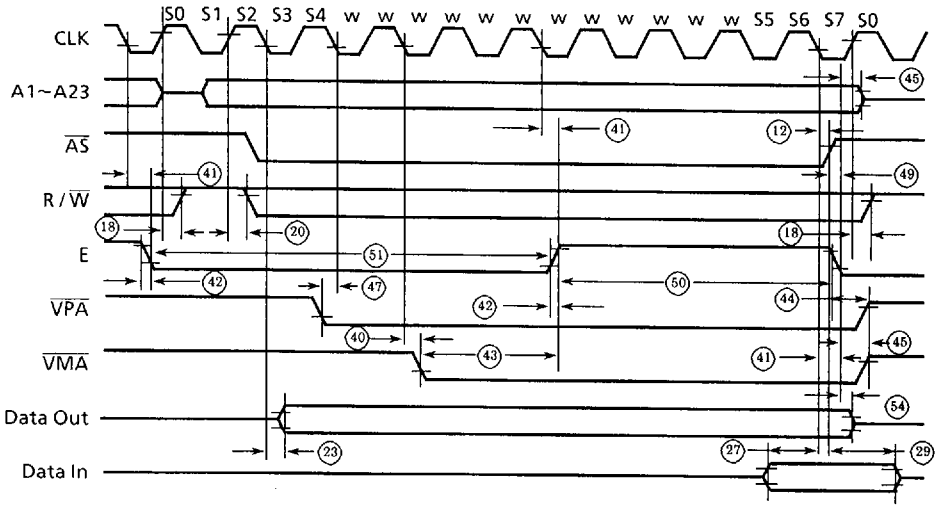
## 8.6 AC ELECTRICAL SPECIFICATIONS – TMP68HC000 TO 6800 PERIPHERAL

($V_{CC}$ = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.5 and 8.6)

| Num. | Characteristic | Symbol | 8MHz | | 10MHz | | 12.5MHz | | 16.67MHz | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | Min | Max | |
| 12[1] | CLock Low to $\overline{AS}$, $\overline{DS}$ High | tCLSH | – | 62 | – | 50 | – | 40 | 3 | 30 | ns |
| 18[1] | Clock High to R / $\overline{W}$ High | tCHRH | 0 | 55 | 0 | 45 | 0 | 40 | 0 | 30 | ns |
| 20[1] | Clock High to R / $\overline{W}$ Low (Write) | tCHRL | 0 | 55 | 0 | 45 | 0 | 40 | 0 | 30 | ns |
| 23 | Clock Low to Data Out Valid (Write) | tCLDO | – | 62 | – | 50 | – | 50 | – | 30 | ns |
| 27 | Data In to Clock Low (Setup Time on Read) | tDICL | 10 | – | 10 | – | 10 | – | 5 | – | ns |
| 29 | $\overline{AS}$, $\overline{DS}$ High to Data in Invalid (Hold Time on Read) | tSHDII | 0 | – | 0 | – | 0 | – | 0 | – | ns |
| 40 | Clock Low to $\overline{VMA}$ Low | tCLVML | – | 70 | – | 70 | – | 70 | – | 50 | ns |
| 41 | Clock Low to E Transition | tCLET | – | 55 | – | 45 | – | 35 | – | 35 | ns |
| 42 | E Output Rise and Fall Time | tEr, f | – | 15 | – | 15 | – | 15 | – | 15 | ns |
| 43 | $\overline{VMA}$ Low to E High | tVMLEH | 200 | – | 150 | – | 90 | – | 80 | – | ns |
| 44 | $\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High | tSHVPH | 0 | 120 | 0 | 90 | 0 | 70 | 0 | 50 | ns |
| 45 | E Low to Control, Address Bus Invalid (Address Hold Time) | tELCAI | 30 | – | 10 | – | 10 | – | 10 | – | ns |
| 47 | Asynchronous Input Setup Time | tASI | 10 | – | 10 | – | 10 | – | 5 | – | ns |
| 49[2] | $\overline{AS}$, $\overline{DS}$ High to E Low | tSHEL | –70 | 70 | –55 | 55 | –45 | 45 | –35 | 35 | ns |
| 50 | E Width High | tEH | 450 | – | 350 | – | 280 | – | 220 | – | ns |
| 51 | E Width Low | tEL | 700 | – | 550 | – | 440 | – | 340 | – | ns |
| 54 | E Low to Data Out Invalid | tELDOI | 30 | – | 20 | – | 15 | – | 10 | – | ns |

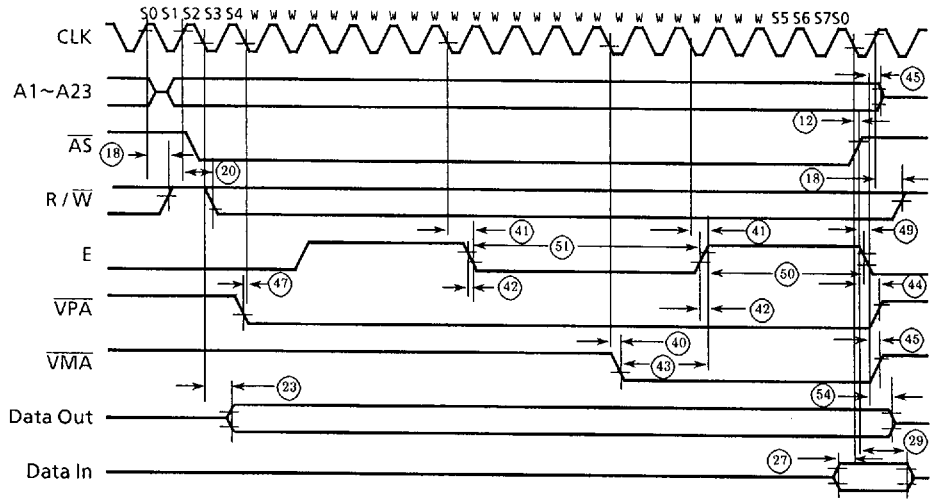
Note 1 : For a loading capacitance of less than or equal to 50 picofarads, subtract 5 nanoseconds from the value given in the maximum columns.

2 : The falling edge of S6 triggers both the negation of the strobes ($\overline{AS}$ and $\overline{xDS}$) and the falling edge of E. Either of these events can occur first, depending upon the loading on each signal. Specification #49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and 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 TMP68HC000 to 6800 Peripheral Timimg Diagram – Best Case




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

Figure 8.6 TMP68HC000 to 6800 Peripheral Timing Diagram – Worst Case

## 8.7 AC ELECTRICAL SPECIFICATIONS – BUS ARBITRATION

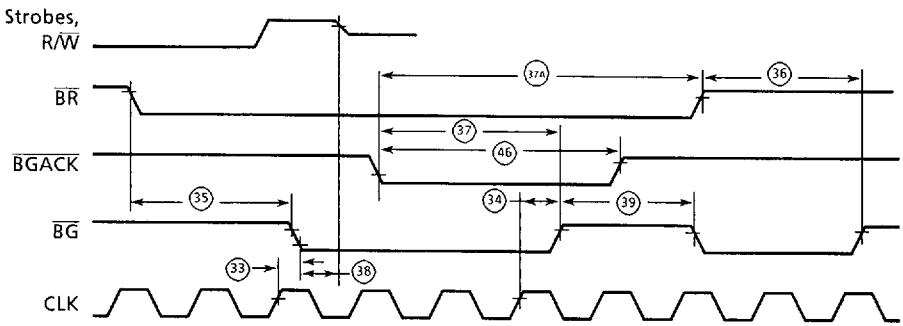
($V_{CC}$ = 5.0V ± 5%, GND = 0V, Ta = 0~70°C; See Figure 8.7)

| Num. | Characteristic | Symbol | 8MHz | | 10MHz | | 12.5MHz | | 16.67MHz | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | Min | Max | |
| 7 | Clock High to Address, Data Bus High Impedance | tCHADZ | – | 80 | – | 70 | – | 60 | – | 50 | ns |
| 16 | Clock High to Control Bus High Impedance | tCHCZ | – | 80 | – | 70 | – | 60 | – | 50 | ns |
| 33 | Clock High to $\overline{BG}$ Low | tCHGL | – | 62 | – | 50 | – | 40 | 0 | 30 | ns |
| 34 | Clock High to $\overline{BG}$ High | tCHGH | – | 62 | – | 50 | – | 40 | 0 | 30 | ns |
| 35 | $\overline{BR}$ Low to $\overline{BG}$ Low | tBRLGL | 1.5 | 3.5 | 1.5 | 3.5 | 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 | 1.5 | 3.5 | 1.5 | 3.5 | Clk. Per. |
| 37 | $\overline{BGACK}$ Low to $\overline{BG}$ High | tGALGH | 1.5 | 3.5 | 1.5 | 3.5 | 1.5 | 3.5 | 1.5 | 3.5 | Clk. Per. |
| 37A[2] | $\overline{BGACK}$ Low to $\overline{BR}$ High | tGALBRH | 20 | 1.5 Clocks | 20 | 1.5 Clocks | 20 | 1.5 Clocks | 10 | 1.5 Clocks | ns |
| 38 | $\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High) | tGLZ | – | 80 | – | 70 | – | 60 | – | 50 | ns |
| 39 | $\overline{BG}$ Width High | tGH | 1.5 | – | 1.5 | – | 1.5 | – | 1.5 | – | Clk. Per. |
| 46 | $\overline{BGACK}$ Width Low | tGAL | 1.5 | – | 1.5 | – | 1.5 | – | 1.5 | – | Clk. Per. |
| 47 | Asynchronous Input Setup Time | tASI | 10 | – | 10 | – | 10 | – | 5 | – | ns |
| 57 | $\overline{BGACK}$ High to Control Bus Driven | tGABD | 1.5 | – | 1.5 | – | 1.5 | – | 1.5 | – | Clk. Per. |
| 58[1] | $\overline{BG}$ High to Control Bus Driven | tGHBD | 1.5 | – | 1.5 | – | 1.5 | – | 1.5 | – | Clk. Per. |

Note: 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 to guarantee proper operation. If the maximum value is exceeded, $\overline{BG}$ may be reasserted.

The waveforms shown in Figures 8.9, 8.10, and 8.11 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 : Setup time to the clock (#47) for the asynchronous imputs $\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{IPL0}$~$\overline{IPL2}$ and $\overline{VPA}$ guarantees their recognition at the next falling edge of the clock.

Figure 8.7 Bus Arbitration Diagram