HMOS 16/32–BIT MICROPROCESSOR

PRELIMINARY INFORMATION

The TS68000 is the first implementation of the 68000 16/32 microprocessor architecture. The TS68000 has a 16-bit data bus and 24-bit address bus while the full architecture provides for 32-bit address and data buses. It is completely code-compatible with the TS68008 8-bit data bus implementation of the 68000 and is downward code-compatible with the TS68020 32-bit implementation of the architecture. Any user-mode programs written using the TS68000 instruction set will run unchanged on the TS68008 and TS68020. This is possible because the user programming model is identical for all three processors and the instruction sets are proper subsets of the complete architecture.

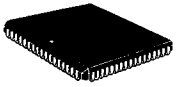
The resources available to the TS68000 user consist of the following :

- 16 32-bit data and address registers
- 16 megabyte direct addressing range
- 56 powerful instruction types
- Operations on five main data types
- Memory mapped I/O
- 14 addressing modes
- 4 available versions : 8MHz, 10MHz, 12.5MHz and 16MHz

As shown in the user programming model, the TS68000 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.

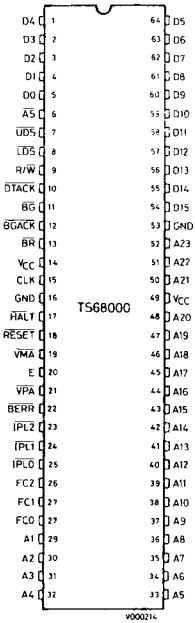


P
(Plastic Package)



FN
(PLCC 68)

PIN CONNECTIONS



SECTION 1

INTRODUCTION

The TS68000 is the first implementation of the 68000 16/32 microprocessor architecture. The TS68000 has a 16-bit data bus and 24-bit address bus while the full architecture provides for 32-bit address an data buses. It is completely code-compatible with the TS68008 8-bit data bus implementation of the 68000 and is downward code-compatible with the TS68020 32-bit implementation of the architecture. Any user-mode programs written using the TS68000 instruction set will run unchanged on the TS68008 and TS68020. This is possible because the user programming model is identical for all four processors and the instruction sets are proper subsets of the complete architecture.

The resources available to the TS68000 user consist of the following :

- 17 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- 56 Powerful Instruction Types
- Operations on Five Main Data Types
- Memory Mapped I/O
- 14 Addressing Modes

As shown in the user programming model (figure 1-1), the TS68000 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.

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.

Figure 1.1 : User Programming Model.

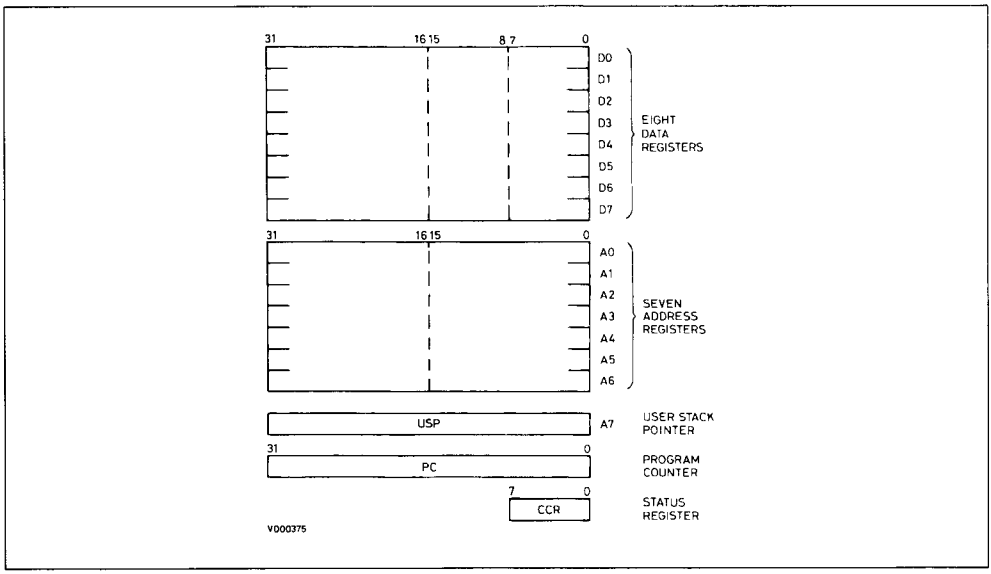


Figure 1.2 : Supervisor Programming Model Supplement.

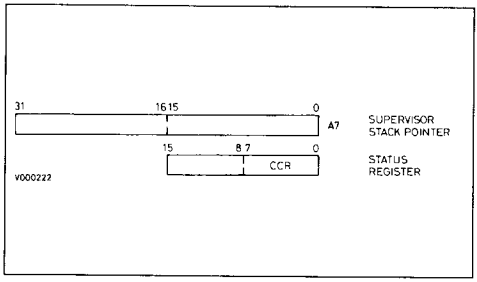


Figure 1.3 : Status Register.

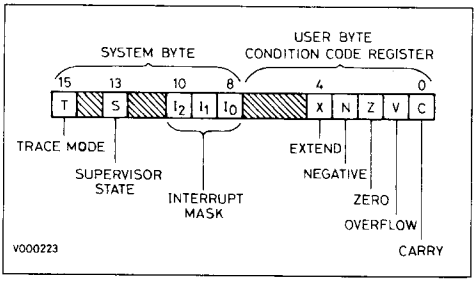


Table 1.1 : Addressing Modes.

Addressing Modes	Syntax
Register Direct Addressing	
Data Register Direct	Dn
Address Register Direct	An
Absolute Data Addressing	
Absolute Short	xxx W
Absolute Long	xxx L
Program Counter Relative Addressing	
Relative with Offset	d_{16}(PC)
Relative with Index Offset	d_8(PC, Xn)
Register Indirect Addressing	
Register Indirect	(An)
Postincrement Register Indirect	(An) +
Predecrement Register Indirect	– (An)
Register Indirect with Offset	d_{16}(An)
Indexed Register Indirect with Offset	d_8(An, Xn)
Immediate Data Addressing	
Immediate	#xxx
Quick Immediate	#1–#8
Implied Addressing	
Implied Register	SR USP SP PC

Notes :
Dn = Data Register
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
d_8 = 8-Bit Offset (displacement)
d_{16} = 16-Bit Offset (displacement)
#XXX = Immediate Data

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

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

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

1.2. INSTRUCTION SET OVERVIEW

The TS68000 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.

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

Table 1.3 : Variations of Instruction Types.

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

SECTION 2

DATA ORGANIZATION AND ADDRESSING CAPABILITIES

This section contains a description of the registers and the data organization of the TS68000.

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 operands of 32 bits.

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

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

2.2.2. ADDRESS REGISTERS. Each address register and 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.

The data types supported by the TS68000 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 would be accessed from the processor.

2.4. ADDRESSING

Instructions for the TS68000 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.

Figure 2.1 : Word Organization in Memory.

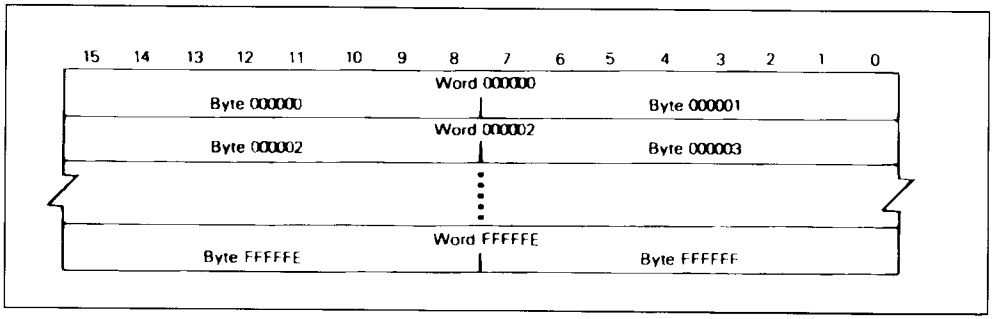
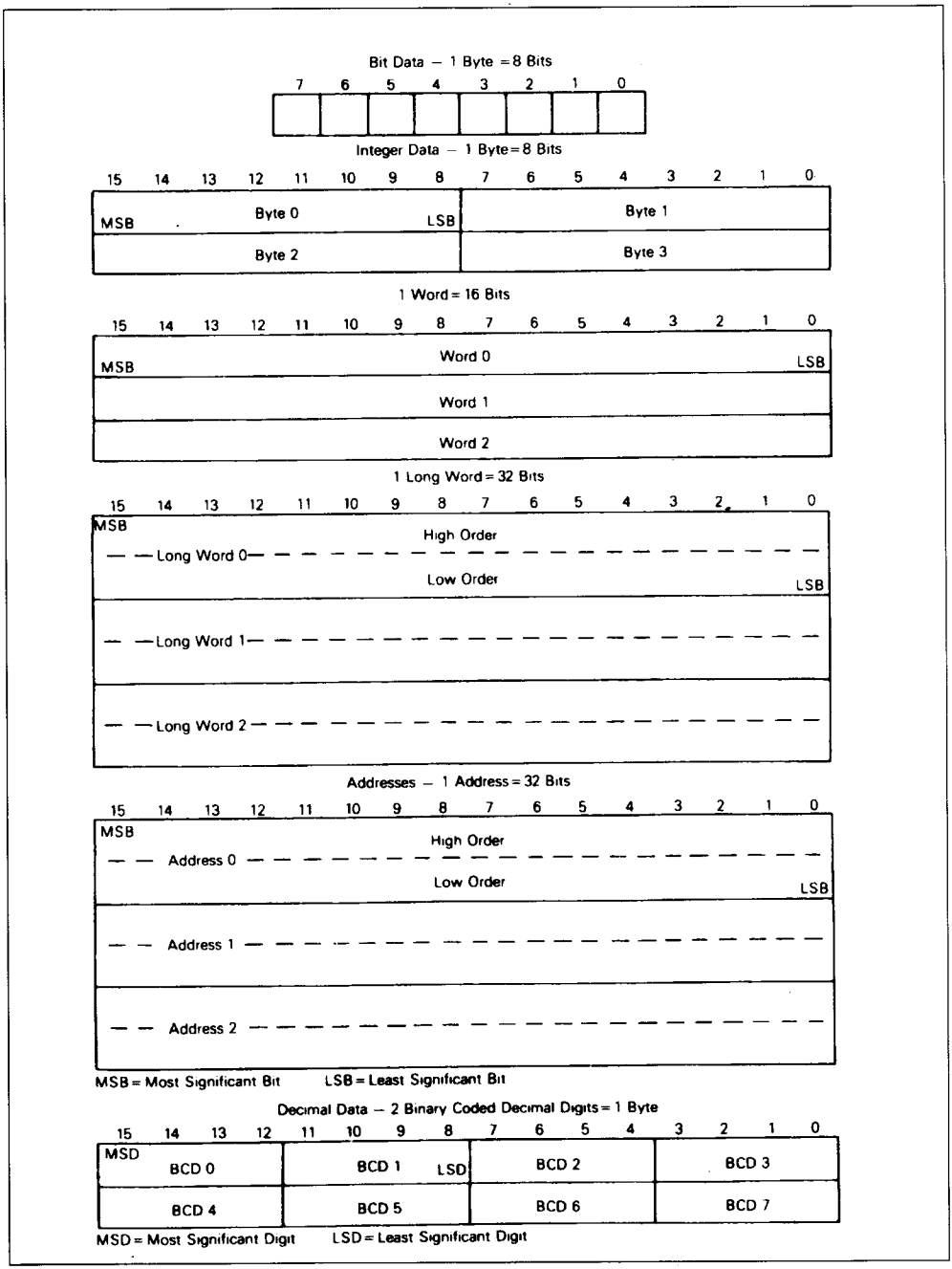


Figure 2.2 : Memory Data Organization.



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.

2.6. PROGRAM/DATA REFERENCES

The TS68000 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. Operands 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.

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.

Figure 2.3 : Instruction Operation Word General Format.

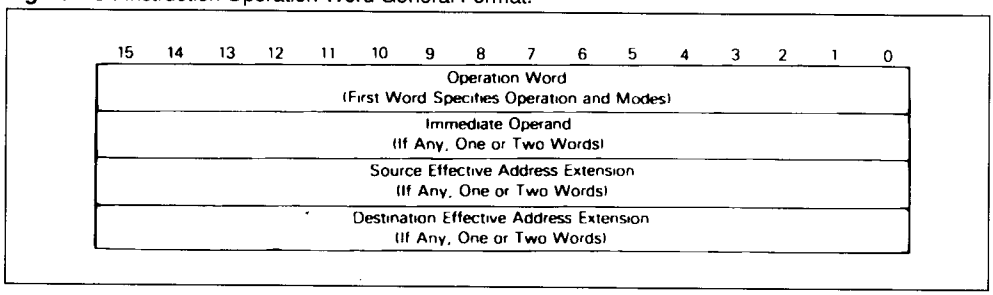
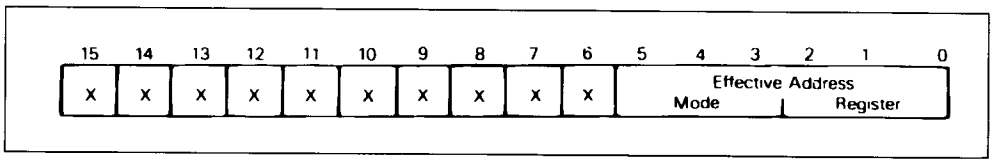


Figure 2.4 : Single–Effective–Address Instruction Operation Word.



2.8. EFFECTIVE ADDRESS (continued)

2.8.2.2. Address Register Indirect with Postincrement.

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

2.8.2.3. Address Register Indirect with Predecrement.

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

2.8.2.4. Address Register Indirect with Displacement.

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

2.8.2.5. Address Register Indirect with Index.

This addressing mode requires one word of extension. The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The 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 instructions.

2.8.3.3. Program Counter with Displacement.

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

2.8.3.4. Program Counter with Index.

This addressing mode requires one word of extension. The address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The 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	EORI to SR	MOVE to CCR
ANDI to SR	ORI to CCR	MOVE to SR
EORI to CCR	ORI to SR	MOVE from SR

2.9. EFFECTIVE ADDRESS ENCODING SUMMARY

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

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.

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

SECTION 3

INSTRUCTION SET SUMMARY

This section contains an overview of the form and structure of the TS68000 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	$R_X \leftrightarrow R_Y$
LEA	32	EA ← An
LINK	–	An → (SP) SP → An SP + Displacement → SP
MOVE	8, 16, 32	s → d
MOVEM	16, 32	(EA) → An, Dn An, Dn → EA

Table 3.1 : (continued)

Instruction	Operand Size	Operation
MOVEP	16, 32	(EA) → Dn Dn → (EA)
MOVEQ	8	#xxx → Dn
PEA	32	EA → – (SP)
SWAP	32	Dn[31:16] ↔ Dn[15:0]
UNLK	–	An → Sp (SP) + → An

Notes : s = source
d = destination
[ ] = bit number
– ( ) = indirect with predecrement
( ) + = indirect with postdecrement
= 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	Dn + (EA) → Dn
		(EA) + Dn → (EA)
		(EA) + #xxx → (EA)
	16, 32	An + (EA) → An
ADDX	8, 26, 32	Dx + Dy + X → Dx
	16, 32	– (Ax) + – (Ay)+ X →, (Ax)
CLR	8, 16, 32	O → EA
CMP	8, 16, 32	Dn – (EA)
		(EA) – #xxx
		(Ax) + – (Ay) –
	16, 32	An – (EA)
DIVS	32 ÷ 16	Dn ÷ (EA) → Dn
DIVU	32 ÷ 16	Dn ÷ (EA) → Dn
EXT	8 → 16	$(Dn)_8 \rightarrow Dn_{16}$
	16 → 32	$(Dn)_{16} \rightarrow Dn_{32}$
MULS	16 x 16 → 32	dN x (EA) → Dn
MULU	16 x 16 → 32	dN x (EA) → Dn
NEG	8, 16, 32	0 – (EA) → (EA)
NEGX	8, 16, 32	O – (EA) – X → (EA)
SUB	8, 16, 32	Dn + (EA) → Dn
		(EA) + Dn → (EA)
		(EA) + #xxx → (EA)
	16, 32	An + (EA) → An
SUBX	8, 16, 32	Dx – Dy – X → Dx
		– (Ax) – – (Ay) – X → (Ax)
TAS	8	[EA] – 0, 1 → EA [7]
TST	8, 16, 32	(EA) – 0

Notes : [ ] = bit number
– ( ) = indirect with predecrement
( ) + = indirect with postdecrement
= 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	DnΛ(EA) → Dn
		(EA)ΛDn → (EA)
		(EA)Λ#xxx → (EA)
OR	8, 16, 32	Dn v (EA) → Dn
		(EA) v Dn → (EA)
		(EA) v #xxx → (EA)
EOR	8, 16, 32	(EA) ⊕ Dy → (EA)
		(EA) ⊕ #xxx → (EA)
NOT	8, 16, 32	~ (EA) → (EA)

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

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.

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

Table 3.4 : Shift and Rotate Operations.

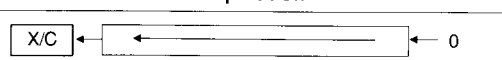
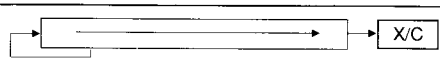
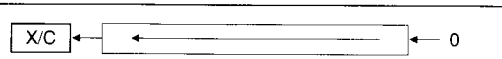
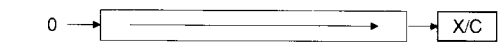
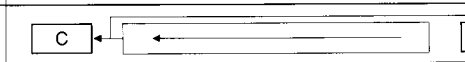
Instruction	Operand Size	Operation
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	

Table 3.4 : Shift and Rotate Operations (continued).

Instruction	Operand Size	Operation
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	

3.5. BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instructions : bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). Table 3-5 is a summary of the 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 1 → bit of EA
BCLR	8, 32	~ bit of (EA) → Z 0 → bit of EA
BCHG	8, 32	~ bit of (EA) → Z ~ bit of (EA) → bit of EA

Note : ~ = Invert

3.6. BINARY CODED DECIMAL OPERATIONS

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

Table 3.6 : Binary Coded Decimal Operations.

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

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
CS - Carry Set
EQ - Equal
F - Never True
GE - Greater or Equal
GT - Greater Than
HI - High
LE - Less or Equal
LS - Low or Same
LT - Less Than
MI - Minus
NE - Not Equal
PL - Plus
T - Always True
VC - no Overflow
VS - Overflow

Table 3.7 : Program Control Operations.

Instruction	Operation
Conditional	
Bcc	Branch Conditionally (14 conditions) 8- and 16-bit Displacement
DBcc	Test Condition, Decrement, and Branch 16-bit Displacement
Scc	Set Byte Conditionally (16 condtions)
Unconditional	
BRA	Branch always 8- and 16-bit Displacement
BSR	Branch to Subroutine 8- and 16-bit Displacement
JMP	Jump
JSR	Jump to Subroutine
Returns	
RTR	Return and Restore Condition Codes
RTS	Return from Subroutine

3.8. SYSTEM CONTROL OPERATIONS

System control 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	Chek 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 Condtion Codes
MOVE EA to CCR	Load New Condition Codes
MOVE SR to EA	Store Status Register
ORI to CCR	Logical OR to Condition Codes

SECTION 4

SIGNAL AND BUS OPERATION DESCRIPTION

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

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

4.1. SIGNAL DESCRIPTION

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

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

4.1.2. DATA BUS (D0 through D15). This 16-bit, bidirectional, three-state bus is the general purpose data path. It can transfer and accept data in either word or byte length. During an interrupt acknowledge cycle, the external device supplies the vector number on data lines D0-D7.

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.

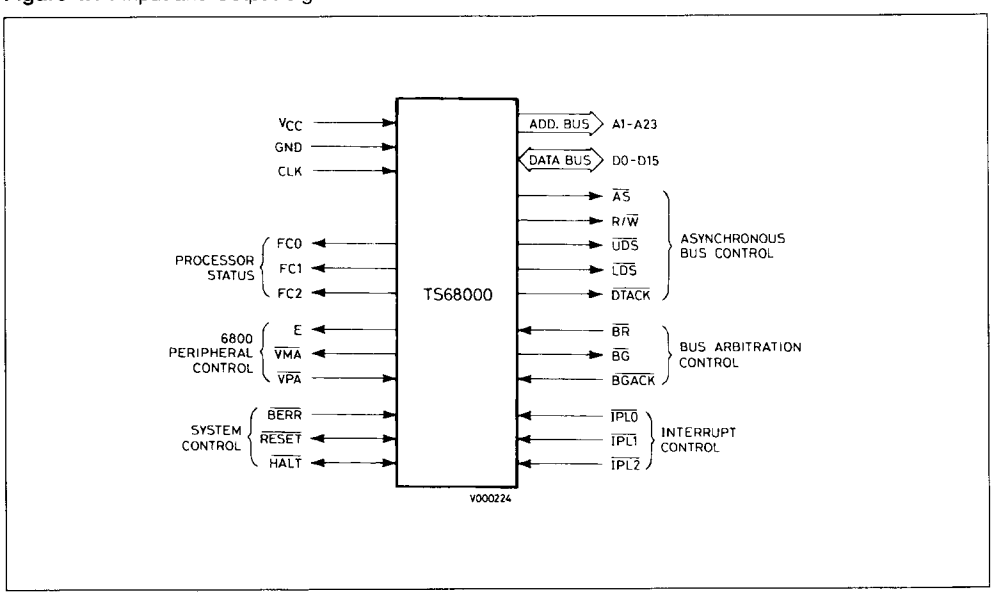
4.1.3.1. Address Strobe ($\overline{AS}$).

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

4.1.3.2. Read/Write (R/W).

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

Figure 4.1 : Input and Output Signals.



4.1.3.3. Upper and Lower Data Strobe (UDS, LDS).

These signals control the flow of data on the data bus, as shown in table 4-1. When the R/W line is high, the processor will read from the data bus as indicated. When the R/W line is low, the processor will write to the data bus as shown.

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 $\overline{DTACK}$ is recognized during a write cycle, the bus cycle is terminated. (Refer to 4.4 Asynchronous Versus Synchronous Operation).

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

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

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

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

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

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

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

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

4.1.5. INTERRUPT CONTROL ($\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$). These input pins indicate the encoded priority level of the device requesting an interrupt. Level seven is the highest priority while level zero indicates that no interrupts are requested. Level seven cannot be masked. The least significant bit is 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.

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 a result of current implementation and may not appear on future devices.

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

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

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

The bus error signal interacts with the halt signal to determine if the current bus cycle should be 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 $\overline{RESET}$ instruction) causes all external devices to be reset and the internal state of the processor is not affected. A total system reset (processor and external devices) is the result of external $\overline{HALT}$ and $\overline{RESET}$ signals applied at the same time. Refer to **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 processor has been halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state (refer to table 4.3). Refer to **4.2.4. Bus Error and Halt Operation** for additional information about the interaction between the $\overline{HALT}$ 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. EF6800 PERIPHERAL CONTROL. These control signals are used to allow the interfacing of synchronous EF6800 peripheral devices with the asynchronous TS68000. These signals are explained in the following paragraphs.

4.1.7.1. Enable (E)

This signal is the standard enable signal common to all EF6800 type peripheral devices. The period for this output is ten TS68000 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 EF6800 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 Ef6800 Peripherals.

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

This output is used to indicate to EF6800 peripheral devices that there is a valid address on the address bus and the processor is synchronized to enable. This signal only responds to a valid peripheral address ($\overline{VPA}$) input which indicates that the peripheral is an EF6800 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.

Function Code output			Cycle Type
FC2	FC1	FC0	
Low	Low	Low	(undefined, reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(undefined, reserved)
High	Low	Low	(undefined, reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

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

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

4.2. BUS OPERATION

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

4.2.1. DATA TRANSFER OPERATIONS. Transfer of data between devices involves the following leads.

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

The address and data buses are separate parallel buses 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 TS68000 for interlocked multiprocessor communications.

Table 4.3 : Signal Summary.

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

Note : 1. Open drain

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

A word read cycle flowchart is given in figure 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.

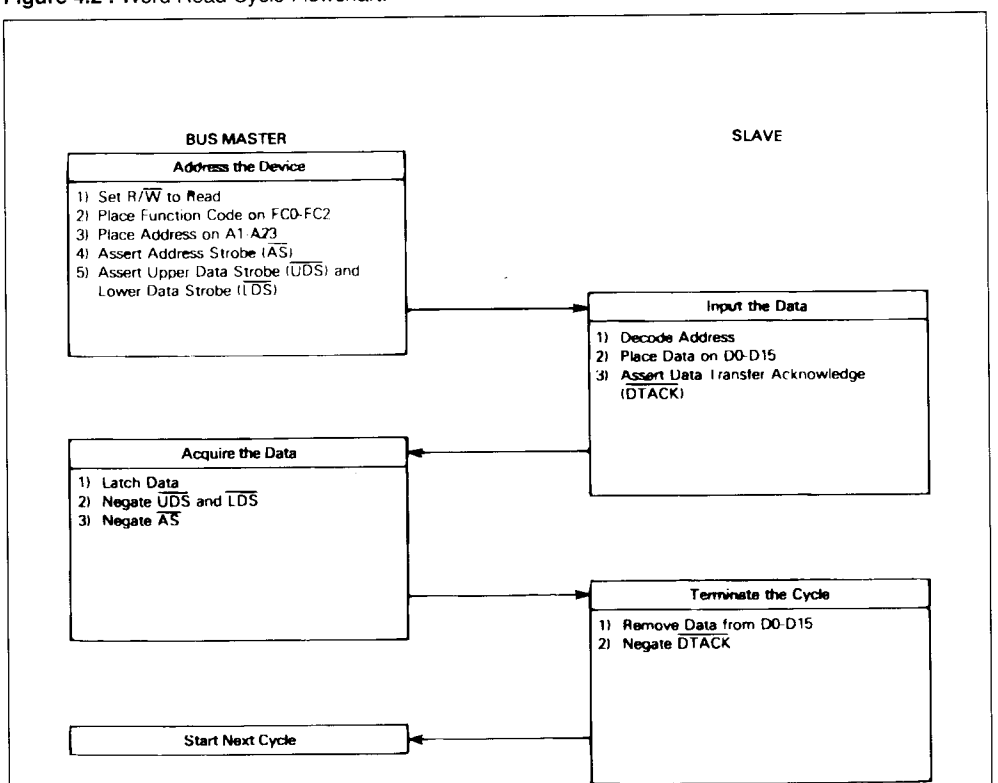


Figure 4.3 : Byte Read Cycle Flowchart.

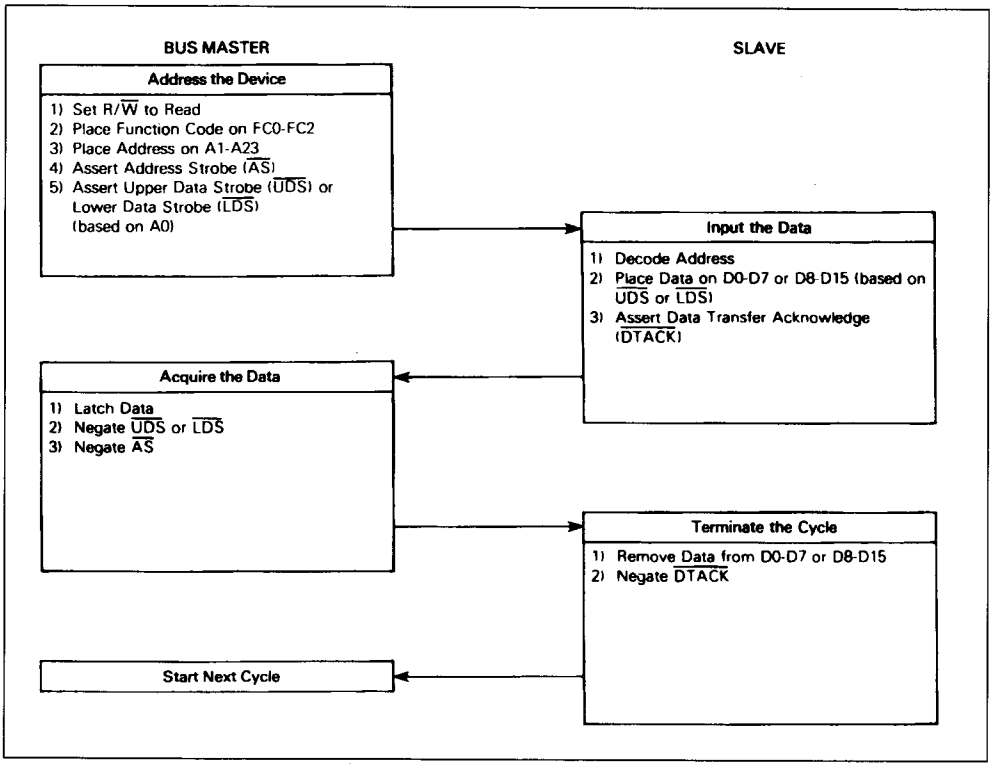


Figure 4.4 : Read and Write Cycle Timing Diagram.

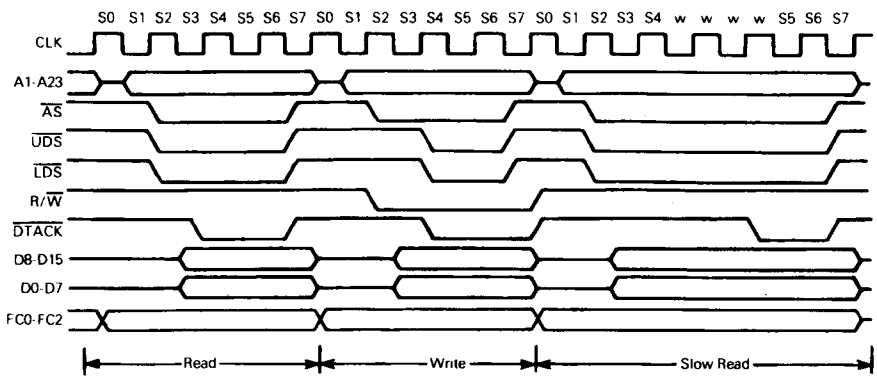
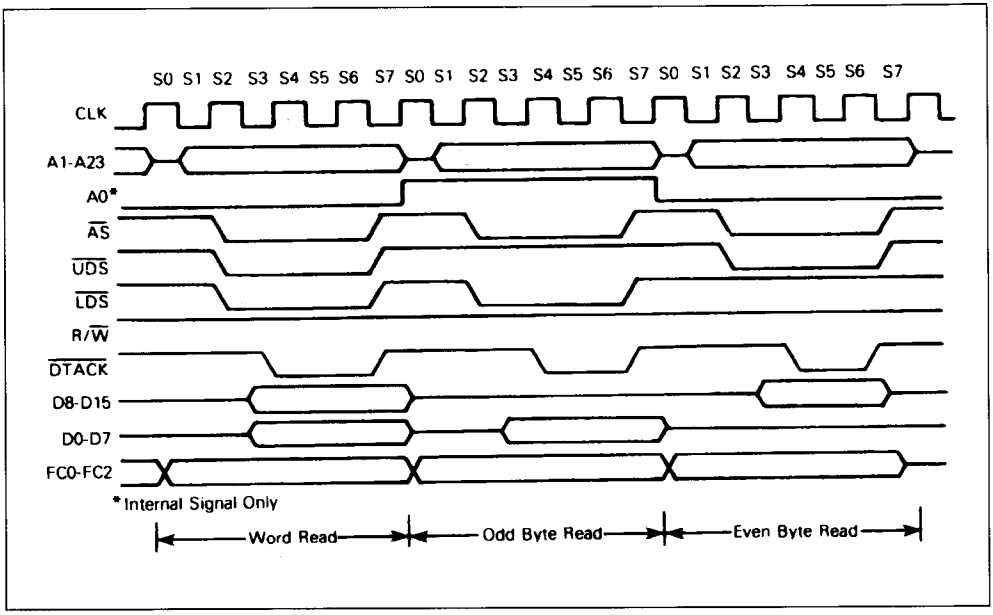


Figure 4.5 : Word and Byte Read Cycle Timing Diagram.



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

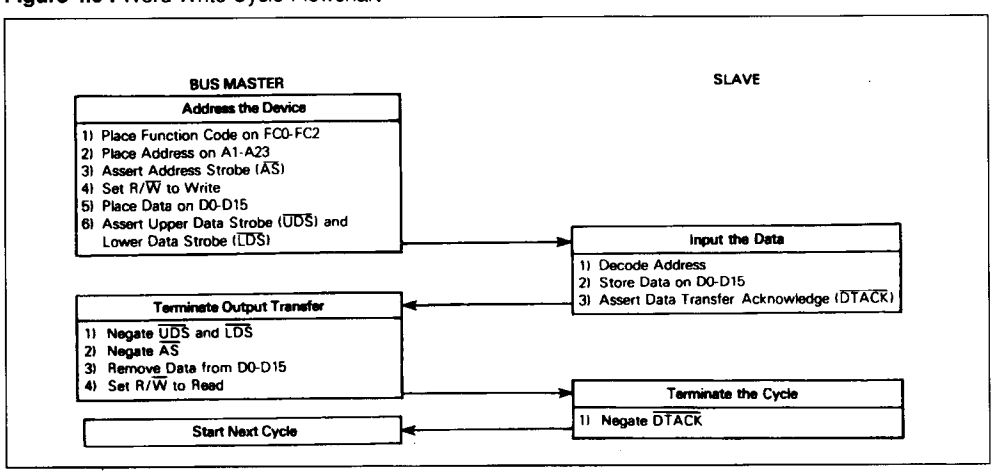


Figure 4.7 : Byte Write Cycle Flowchart.

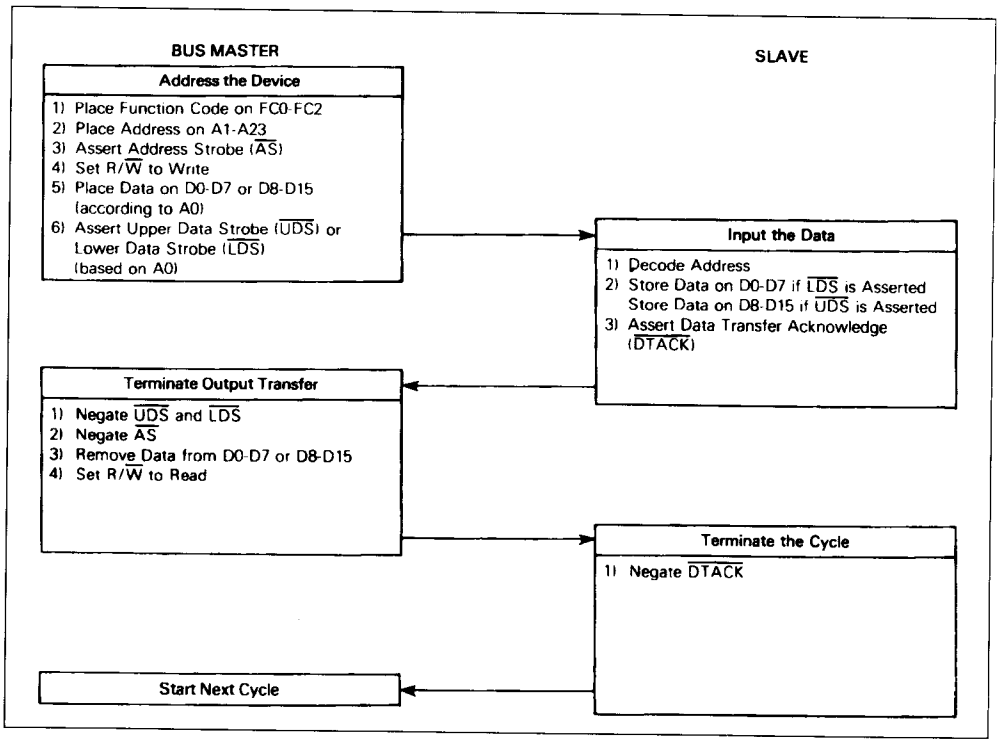
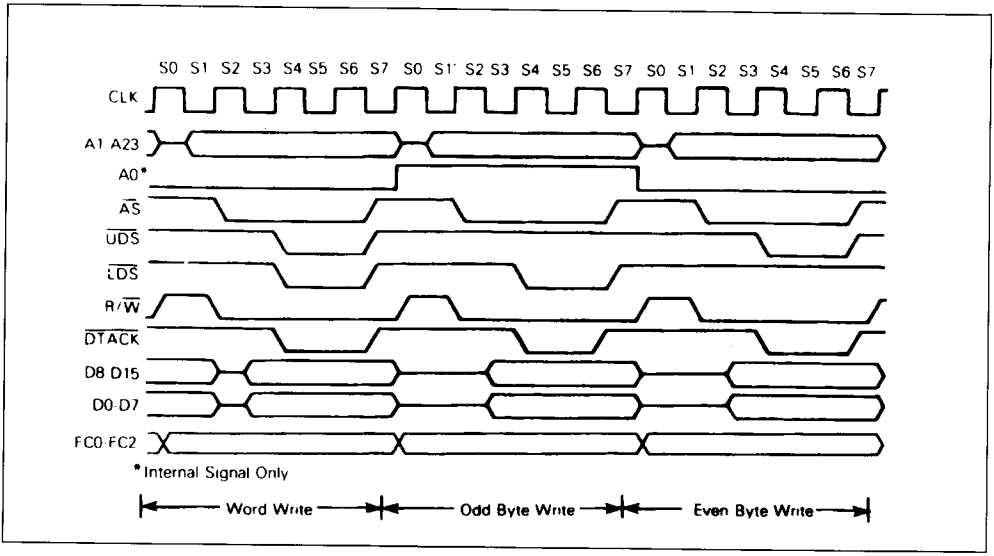


Figure 4.8 : Word and Byte Write Cycle Timing Diagram.



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 TS68000, 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.

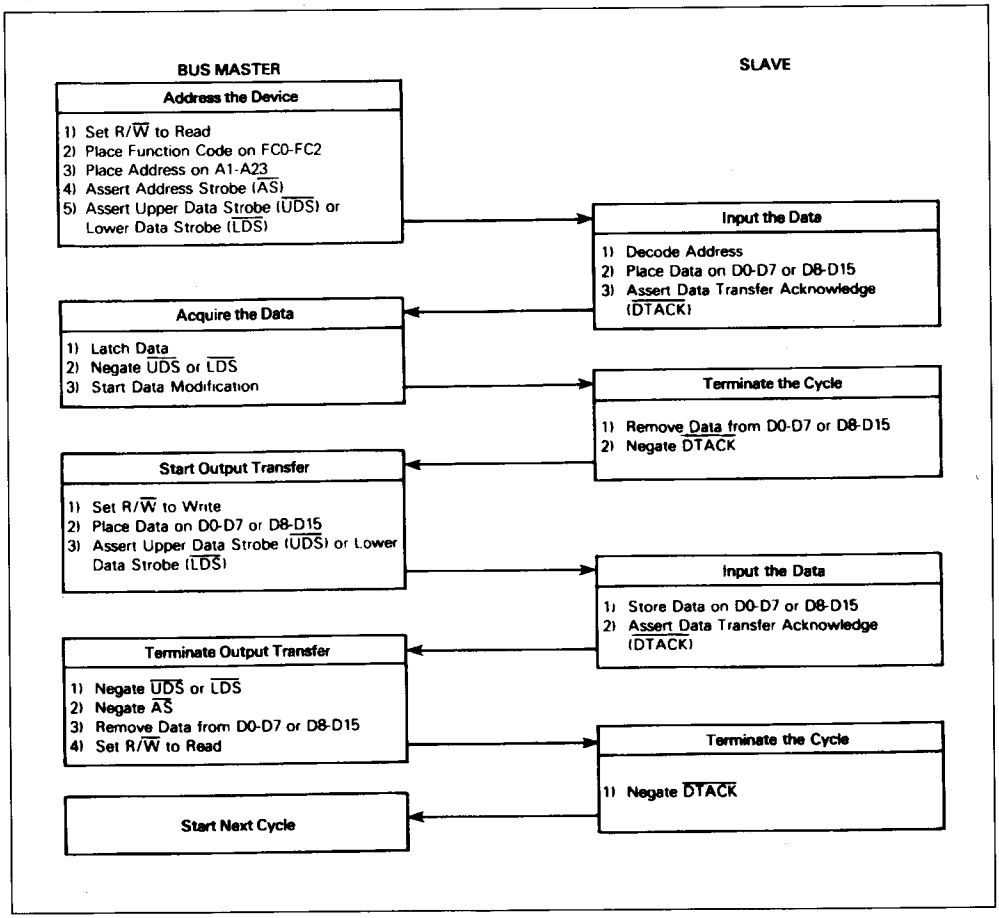
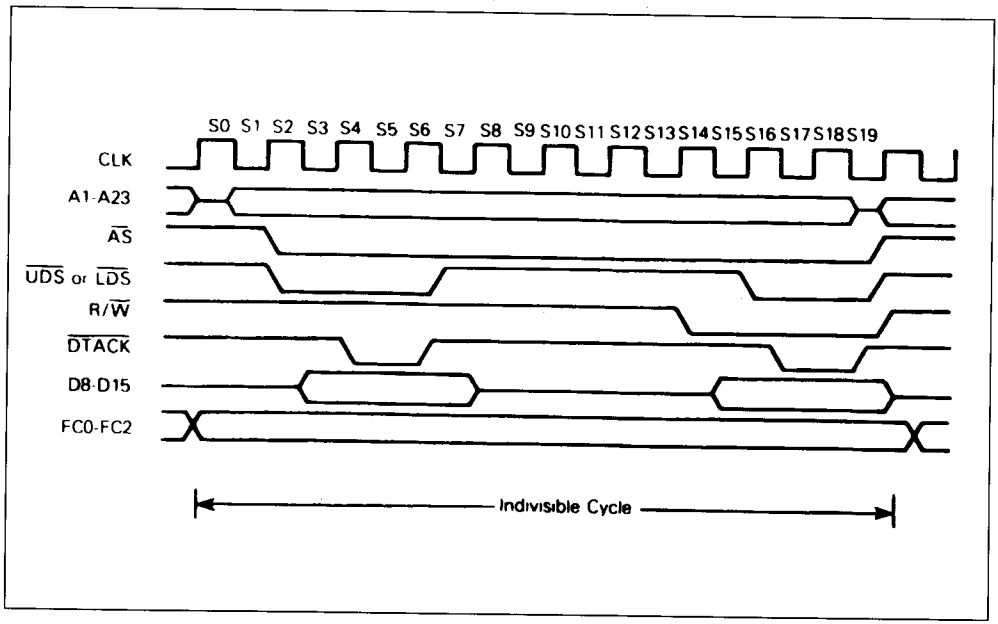


Figure 4.10 : Read–Modify–Write Cycle Timing Diagram.



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

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

Figure 4.11 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 bus mastership, the bus request line from each device is wire ORed to the processor. In this system, it is easy to see that there could be more than one bus request being made. The timing diagram shows that the bus grant signal is negated a few clock cycles after the transition of the acknowledge (BGACK) signal.

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

Figure 4.11 : Bus Arbitration Cycle Flowchart.

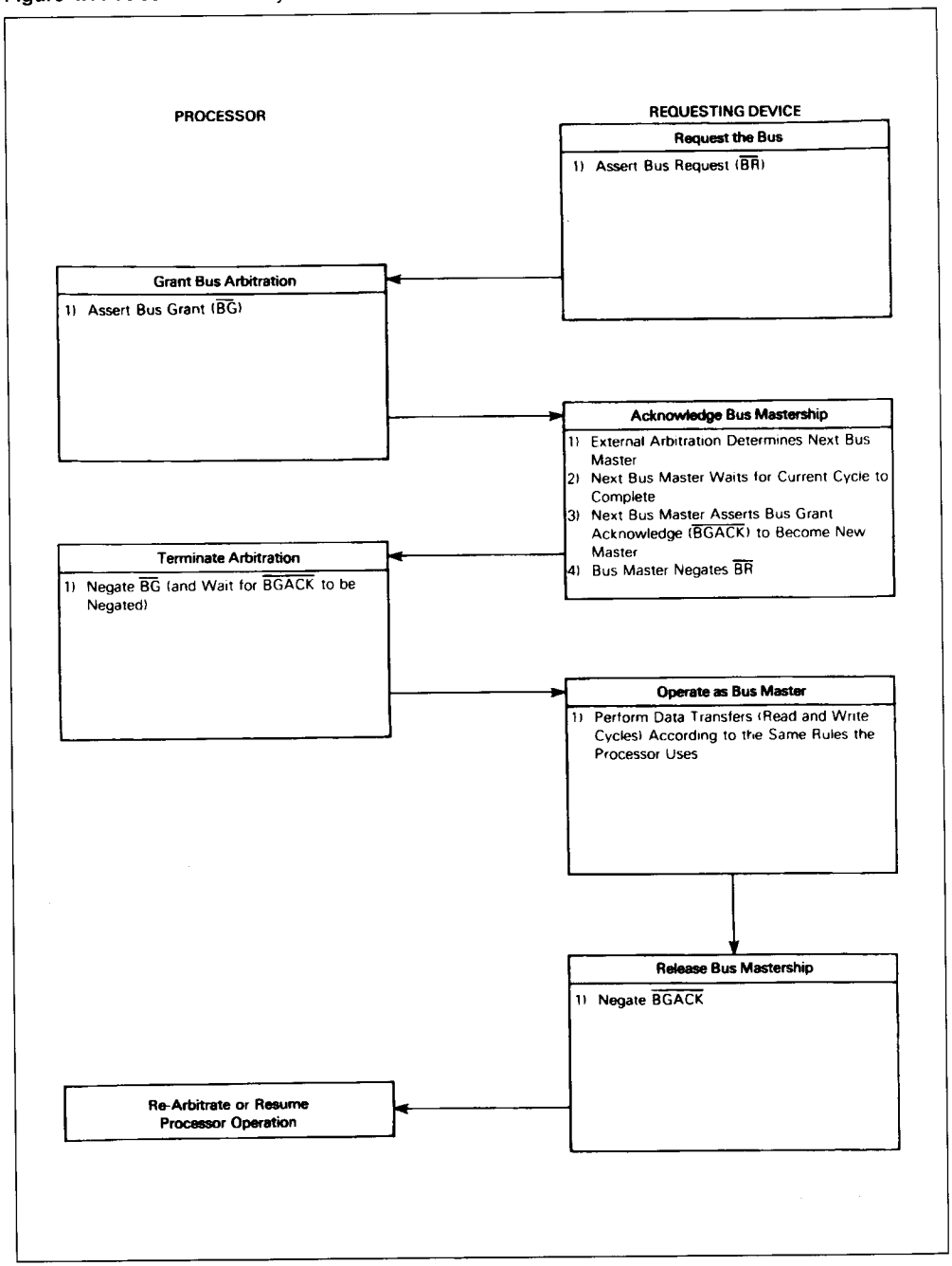
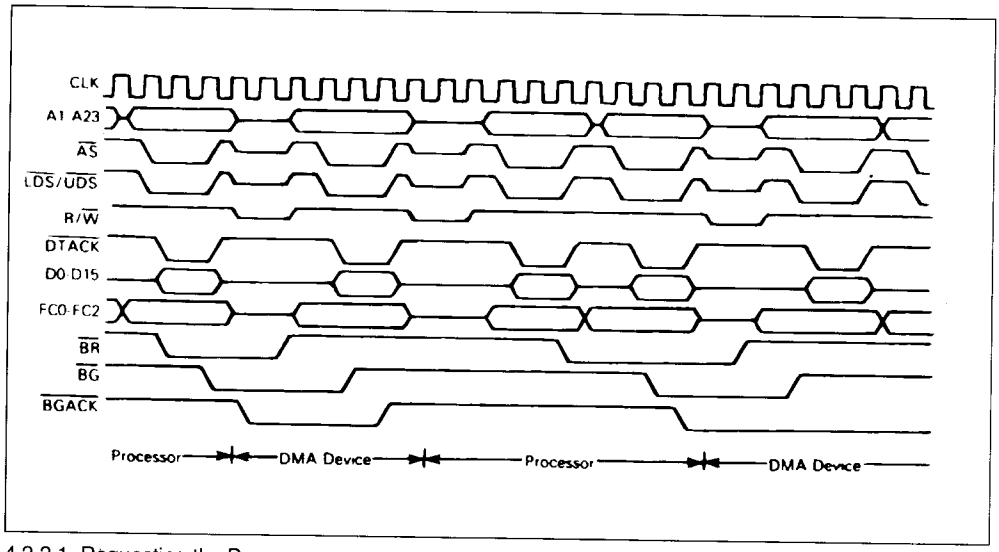


Figure 4.12 : Bus Arbitration Cycle Timing Diagram.



4.2.2.1. Requesting the Bus

External devices capable of becoming bus masters request the bus by asserting the bus request (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 (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 (AS) signal. In this case, bus grant will be delayed until AS is asserted to indicate to external devices that a bus cycle is being executed.

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

4.2.2.3. Acknowledgement of Mastership

Upon receiving a bus grant, the requesting device waits until address strobe, data transfer acknowledge, and bus grant acknowledge are negated before issuing its own 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 TS68000 is implemented with a finite state machine. A state diagram of this machine is shown in figure 4.13. All asynchronous signals to the TS68000 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.

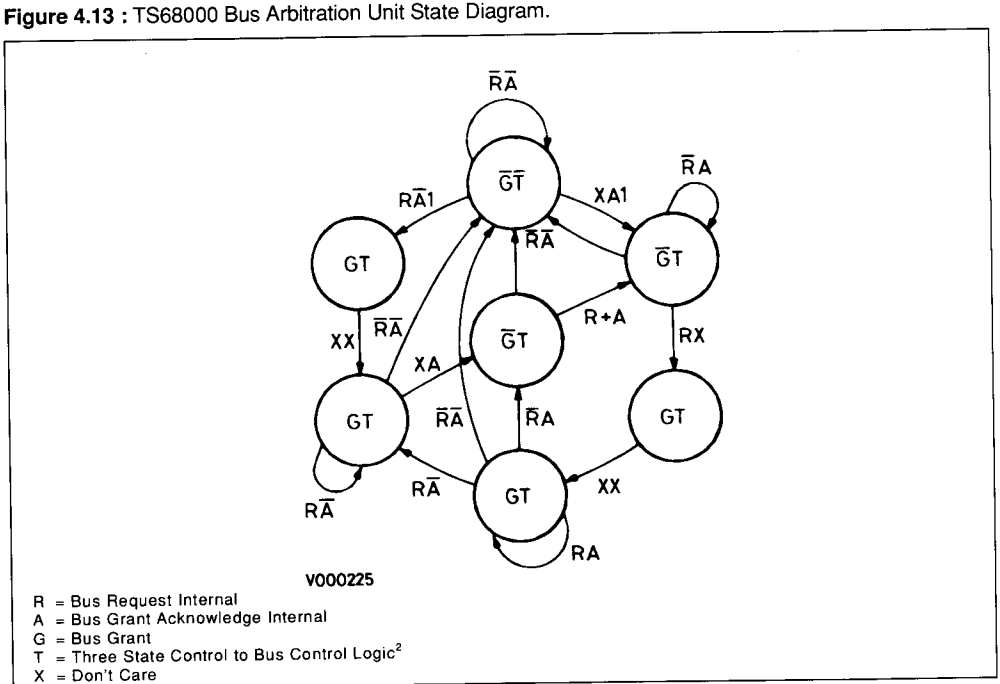
As shown in figure 4.13, input signals labeled R and A are internally synchronized on the bus request and bus grant acknowldege pins respectively. The bus grant output is labeled G and the internal three-state control signal T. If T is true, the address, data, and control buses are placed in a high-impedance state when 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 show in figure 4.16.

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

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.

Figure 4.13 : TS68000 Bus Arbitration Unit State Diagram.



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

Notes : 1. State machine will not change if 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 AS is negated.

Figure 4.14 : Timing Relationship of External Asynchronous Inputs to Internal Signals.

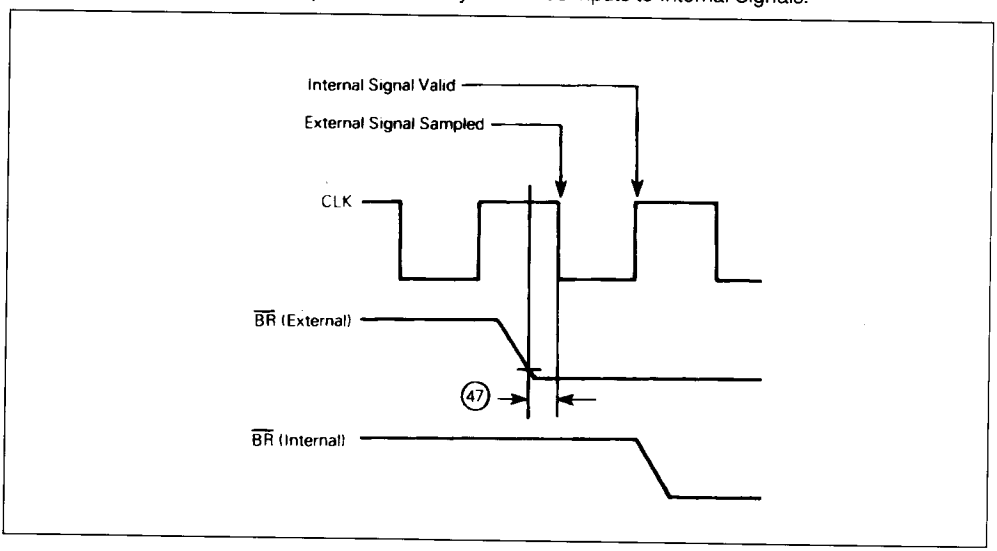


Figure 4.15 : Bus Arbitration Timing Diagram – Processor Active.

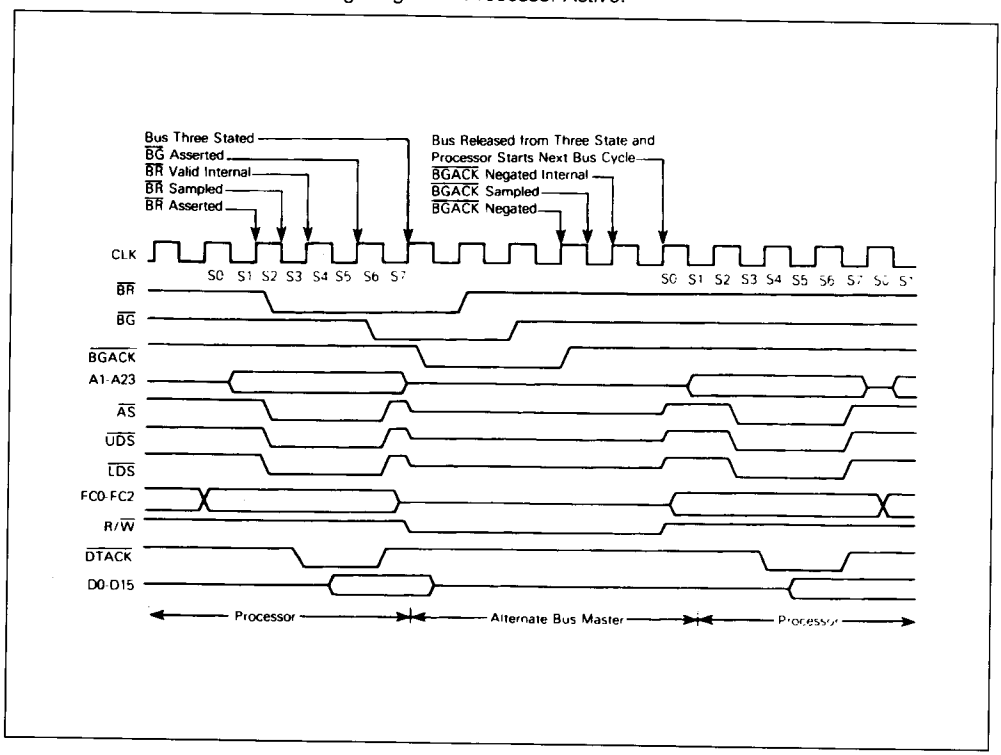


Figure 4.16 : Bus Arbitration Timing Diagram – Bus Inactive.

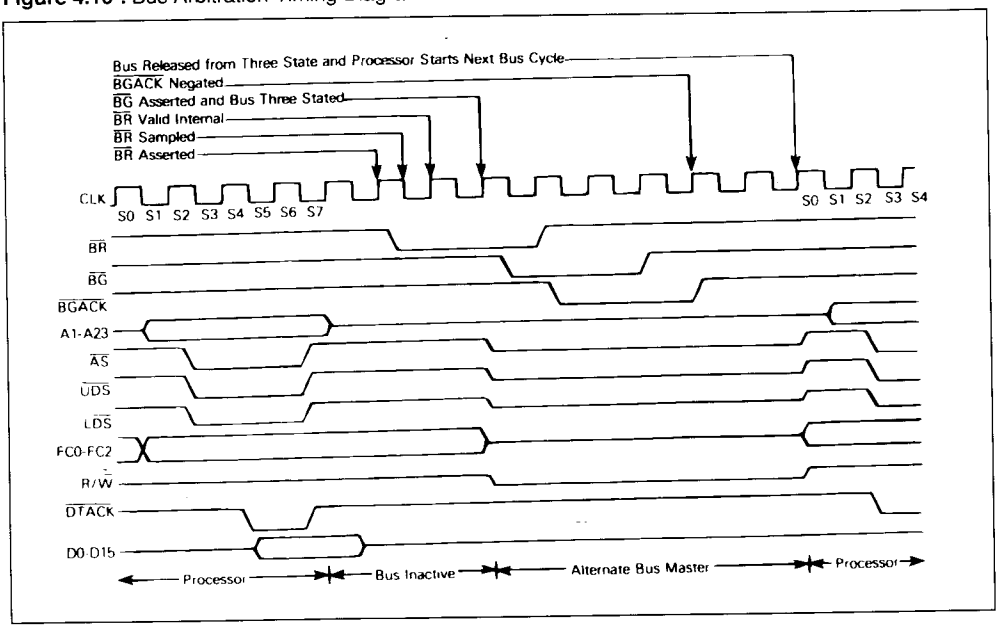
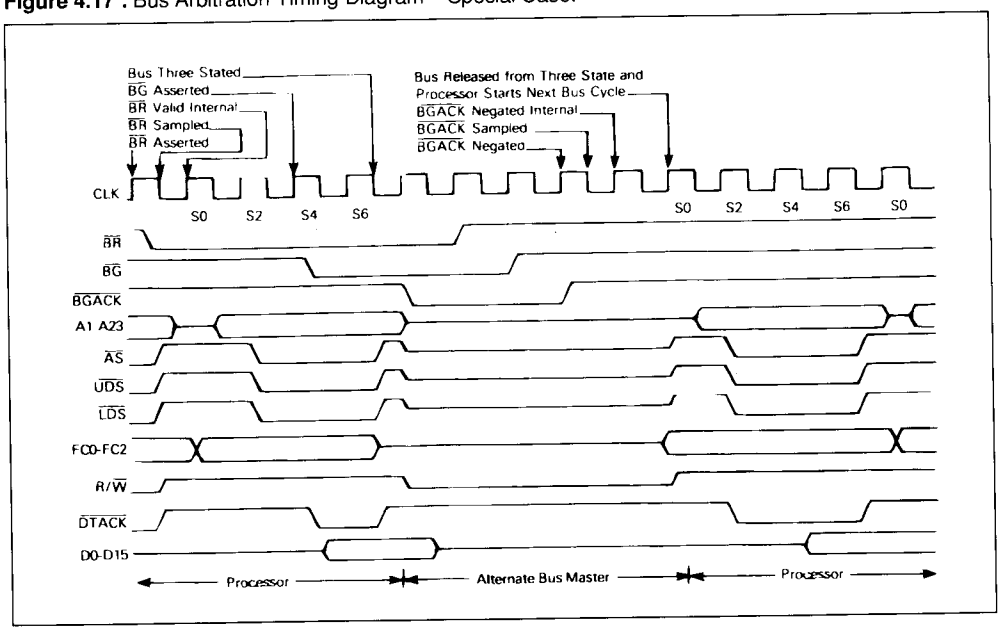


Figure 4.17 : Bus Arbitration Timing Diagram – Special Case.



4.2.4.1. Bus Error Operation

When the bus error signal is asserted, the current bus cycle is terminated. If BERR is asserted before the falling edge of S2, AS will be negated in S7 in either a read or write cycle. As long as BERR remains asserted, the data and address buses will be in the high-impedance state. When 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, and
- 4. executing the bus error handler routine.

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

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

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 AS. If BERR and HALT are asserted during a read-modify-write bus cycle, a bus error operation results.

Figure 4.18 : Bus Error Timing Diagram.

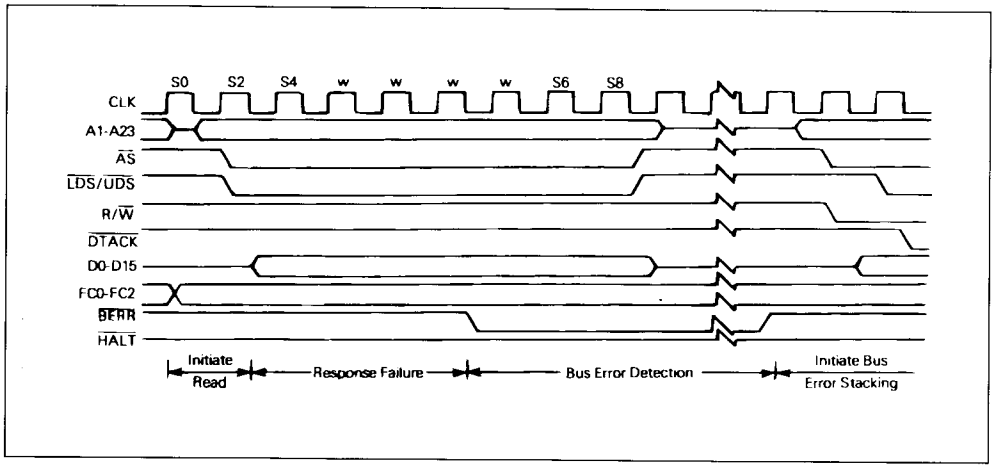
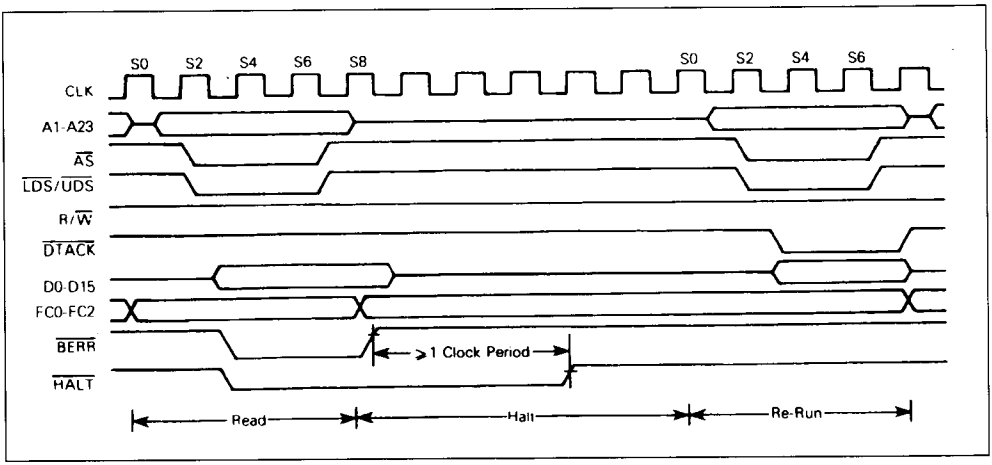


Figure 4.19 : Re–Run Bus Cycle Timing Diagram.



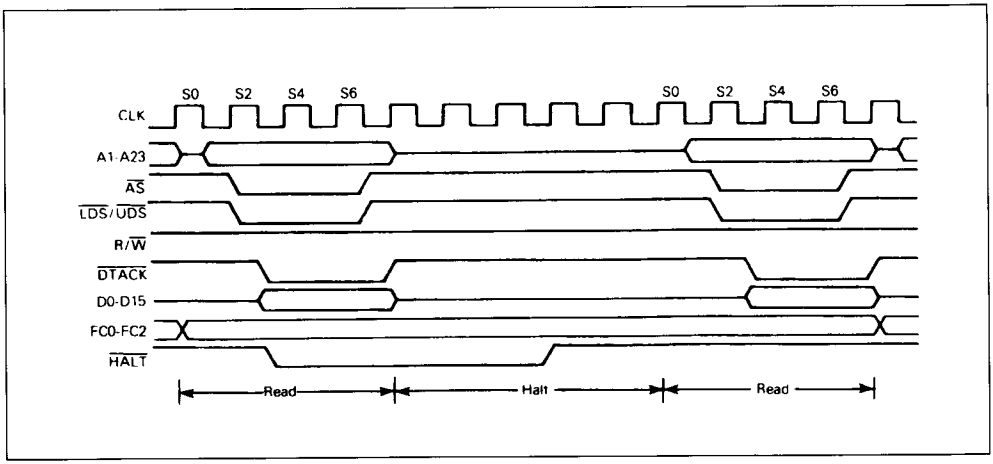
4.2.4.3. Halt Operation

The halt input signal to the TS68000 performs a halt/run/single-step function in a similar fashion to the EF6800 halt function. The halt and run modes are somewhat self explanatory in that when the halt signal is constantly active the processor "halts" (does nothing) and when the halt signal is constantly inactive the processor "runs" (does something).

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

Figure 4.20 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-impedance state, these include :

- 1. address lines, and
- 2. data lines.

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

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

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

4.2.4.4. Double bus faults

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

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

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

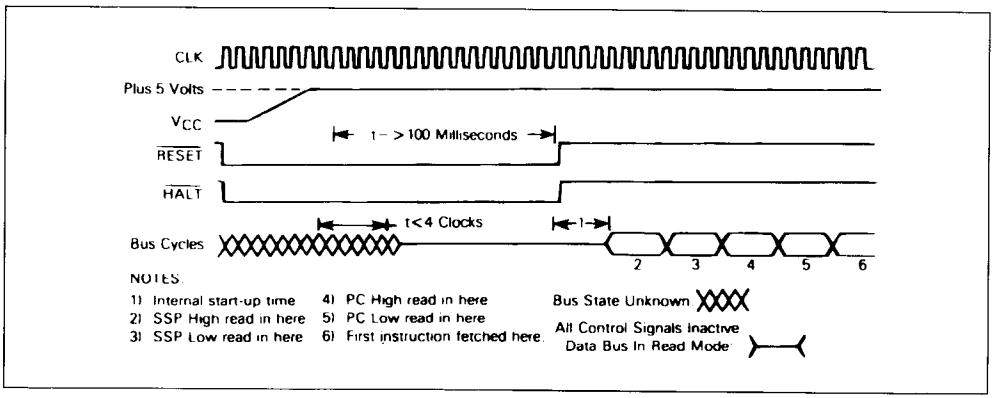
4.2.5. RESET OPERATION. The reset signal is a bidirectional signal that allows either the processor or an external signal to reset the system. Figure 4.21 is a timing diagram for the reset operation. Both the halt and reset lines must be asserted to ensure total reset of the processor.

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 V_{CC} is initially applied to the processor. In this case, an external reset must be applied for at least 100 milliseconds.

Figure 4.21 : Reset Operation Timing Diagram.



4.3. THE RELATIONSHIP OF DTACK, BERR, AND HALT

In order to properly control termination of a bus cycle for a re-run or a bus error condition, DTACK, BERR, and HALT should be asserted and negated on the rising edge of the TS68000 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 TS68000. 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 : DTACK occurs first (case 1).

Halt Termination : HALT is asserted at the same time or before DTACK and BERR remains negated (cases 2 and 3).

Bus Error Termination BERR is asserted in lieu of, at the same time, or before DTACK (case 4) ; BERR is negated at the same time or after DTACK.

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

Table 4.4 : DTACK, BERR and HALT Assertion Results.

Case N°	Control Signal	Asserted. on Rising Edge of State		Result
		N	N + 2	
1	DTACK	A	S	Normal cycle terminate and continue
	BERR	NA	X	
	HALT	NA	X	
2	DTACK	A	S	Normal cycle terminate and halt. Continue when HALT removed.
	BERR	NA	X	
	HALT	A	S	
3	DTACK	NA	A	Normal cycle terminate and halt. Continue when HALT removed.
	BERR	NA	NA	
	HALT	A	S	
4	DTACK	X	X	Terminate and take bus error trap.
	BERR	A	S	
	HALT	NA	NA	
5	DTACK	NA	X	Terminate and re-run.
	BERR	A	S	
	HALT	NA	A	
6	DTACK	X	X	Terminate and re-run when HALT removed.
	BERR	X	S	
	HALT	A	S	
7	DTACK	NA	X	Terminate and re-run when HALT removed.
	BERR	NA	A	
	HALT	A	S	

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

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 (DTACK is assumed to be negated normally in all cases ; for best results, both DTACK and BERR should be negated when address strobe is negated).

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

Conditions of Termination in Table 4–4	Control Signal	Negated an Rising Edge of State			Results - Next Cycle
		n		n + 2	
Bus Error	$\overline{BERR}$	•	or	•	Takes bus error trap
	$\overline{HALT}$	•	or	•	
Re-run	$\overline{BERR}$	•	or	•	Illegal sequence, usually traps to vector number 0.
	$\overline{HALT}$	•			
Re-run	$\overline{BERR}$	•			Re-runs the bus cycle
	$\overline{HALT}$			•	
Normal	$\overline{BERR}$	•			May Lengthen Next Cycle
	$\overline{HALT}$	•	or	•	
Normal	$\overline{BERR}$			•	If next cycle is started it will be terminated as a bus error.
	$\overline{HALT}$	•	or	None	

• = 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 DTACK and BERR simultaneously after time out (case 4).

EXAMPLE B :

A system uses error detection on RAM contents. Designer may (a) delay DTACK until data verified and return BERR and HALT simultaneously to re-run error cycle (case 6), or if valid, return DTACK (case 1) ; (b) delay DTACK until data verified and return BERR at same time as DTACK if data in error (case 4).

4.4. ASYNCHRONOUS VERSUS SYNCRHONOUS OPERATION

4.4.1. ASYNCHRONOUS OPERATION. To achieve clock frequency independence at a system level, the TS68000 can be used in an asynchronous manner. This entails using only the bus handshake lines (AS, UDS, LDS, DTACK, BERR, HALT, and VPA) to control the data transfer. Using this method, AS signals the start of a bus cycle and the data strobes are used as a condition for valid data on a write cycle. The slave device (memory or peripheral) then responds by placing the requested data on the data bus for a read cycle or latching data on a write cycle and asserting the data transfer acknowledge signal (DTACK) to terminate the bus cycle. If no slave responds or the access is invalid, external control logic asserts the BERR, or BERR and HALT, signal to abort or re-run the bus cycle.

The 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 DTACK may precede data is given as parameter #31u 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 AS to the assertion of DTACK. This is because the MPU will insert wait cycles of one clock period each until DTACK is recognized.

4.4.2. SYNCHRONOUS OPERATION. To allow for those systems which use the system clock as a signal to generate 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 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 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 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.

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

SECTION 5

PROCESSING STATES

This section describes the actions of the TS68000 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 TS68000 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 design vide an efficient context switch so that the processor may handle unusual conditions.

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

5.1. PRIVILEGE STATES

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

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

The privilege mechanism provides security by allowing most programs to execute in user state. In this state, the accesses are controlled, and the effects on other parts of the system are limited. The operating system executes in the supervisor state, has access to all resources, and performs the overhead tasks for the user 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 bus cycles generated by instructions executed in the supervisor state are classified as supervisor references. While the processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the supervisor stack pointer.

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

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 interrupt acknowledge. Table 5.1 lists the classification of references.

Table 5.1 : Bus Cycle Classification.

Function Code output			Reference Class
FC2	FC1	FC0	
0	0	0	(unassigned)
0	0	1	User Data
0	1	0	User Program
0	1	1	(unassigned)
1	0	0	(unasigned)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

5.2. EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made and the status register is set for exception processing. In the second step the exception vector is determined and the third step is the saving of the current processor context. In the fourth step a new 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 through D7. The processor translates the vector number into a full 24-bit address, shown in figure 5.3. The memory layout of for exception vectors is given in table 5.2.

Figure 5.1 : Format of Vector Table Entries.

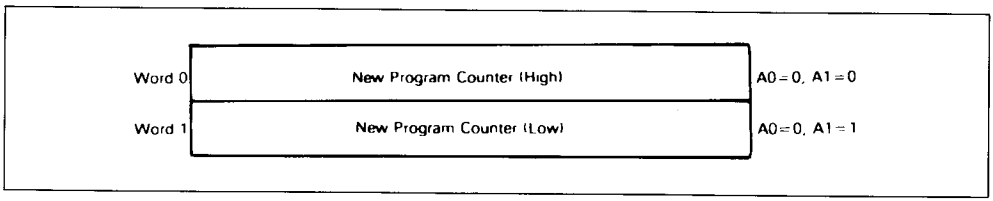


Figure 5.2 : Vector Number Format.

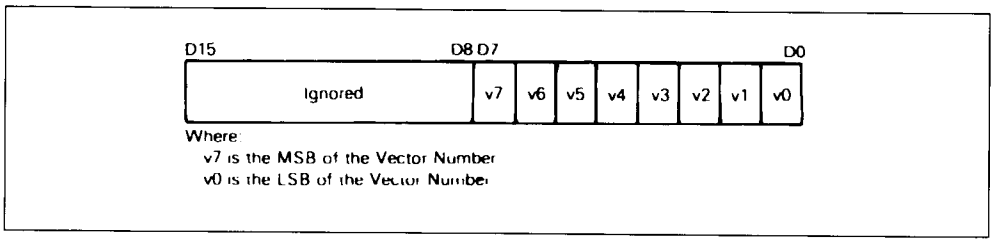


Figure 5.3 : Exception Vector Address Calculation.

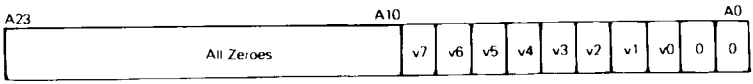


Table 5.2 : Exception Vector Table.

Vector Number(s)	Address			Assigment
	Dec	Hex	Space	
0	0	000	SP	Reset Initial SSP
–	4	004	SP	Reset Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(unassigned, reserved)
13*	52	034	SD	(unassigned, reserved)
14*	56	038	SD	(unassigned, reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16–23*	64	04C	SD	(unassigned, reserved)
	95	05F		–
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32–47	128	080	SD	TRAP Instruction Vectors
	191	0BF		–
48–63*	192	0C0	SD	(unassigned, reserved)
	255	0FF		–
64–225	256	100	SD	User Interrupt Vectors
	1023	3FF		–

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

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.

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.

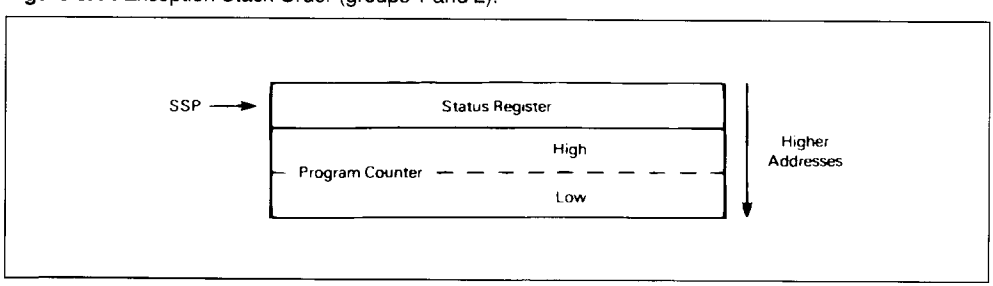
The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register arer stacked using the supervisor stack pointer as shown in figure 5.4. The program counter value stacked usually points to the next unexecuted 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.

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 instructions. These exceptions allow the current instruction to execute to completion, but pre-empt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Figure 5.4 : Exception Stack Order (groups 1 and 2).



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

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instruction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T bit is asserted, the trace exception has priority, and is processed first. Before instruction processing resumes, however, the interrupt exception is also processed, and instruction processing commences finally in the interrupt handler routine. A summary of exception grouping and priority is given in table 5.3.

Table 5.3 : Exception Grouping and Priority.

Group	Exception	Processing
0	Reset Address Error Bus Error	Exception processing begins within two clock cycles.
1	Trace Interrupt Illegal Privilege	Exception processing begins before the next instruction.
2	TRAP, TRAPV CHK Zero Divivde	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 exceptionsn, how each arises, and how each is processed.

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

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

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

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

If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. A copy of the status register is saved, the privilege state is sent to the supervisor stack, tracing is suppressed, and the processor priority level is set to the level of the interrupt acknowledged. The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests an automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number.

If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the program counter and status register on the supervisor stack. The saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine. A flowchart for the interrupt acknowledge sequence is given in figure 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 request level changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

5.3.3. UNINITIALIZED INTERRUPT. An interrupting device asserts VPA or provides an interrupt during an interrupt acknowledge cycle to the TS68000. If the vector register has not been initialized, the responding TS68000 Family peripheral will provide vector 15, the uninitialized interrupt vector. This provides a uniform way to recover from a programming error.

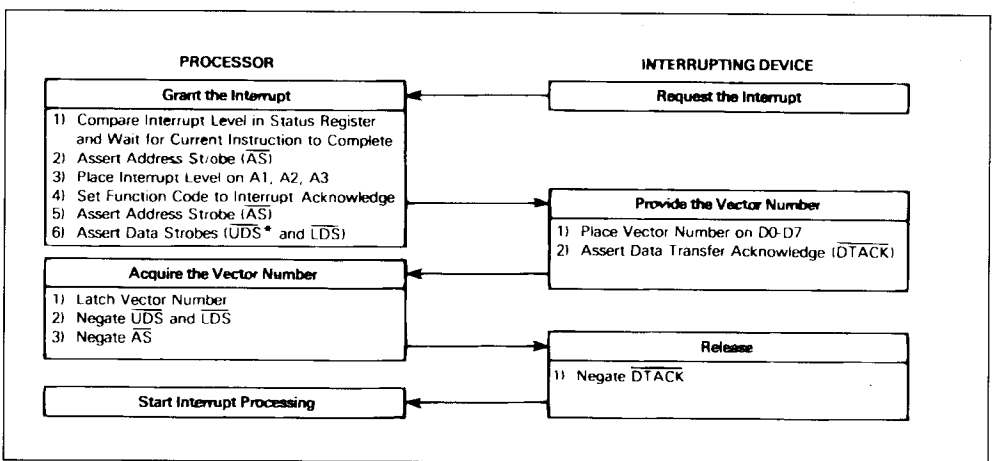
5.3.4. SPURIOUS INTERRUPT. If during the interrupt acknowledge cycle no device responds by asserting DTACK or VPA, the bus error line should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by fetching the spurious interrupt vector instead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

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

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

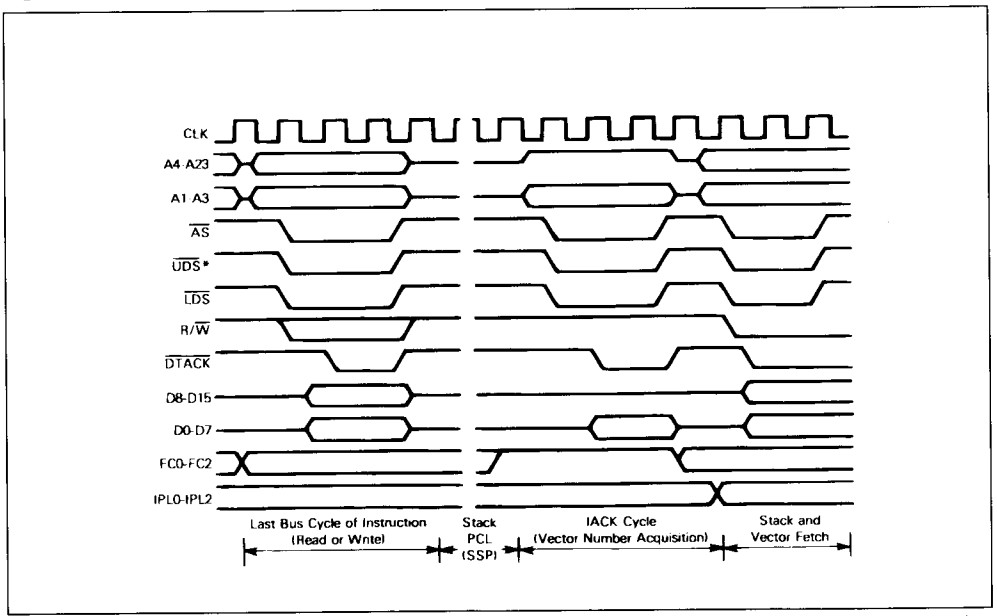
The signed divide (DIVS) and unsigned (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

Figure 5.5 : Vector Acquisition Flowchart.



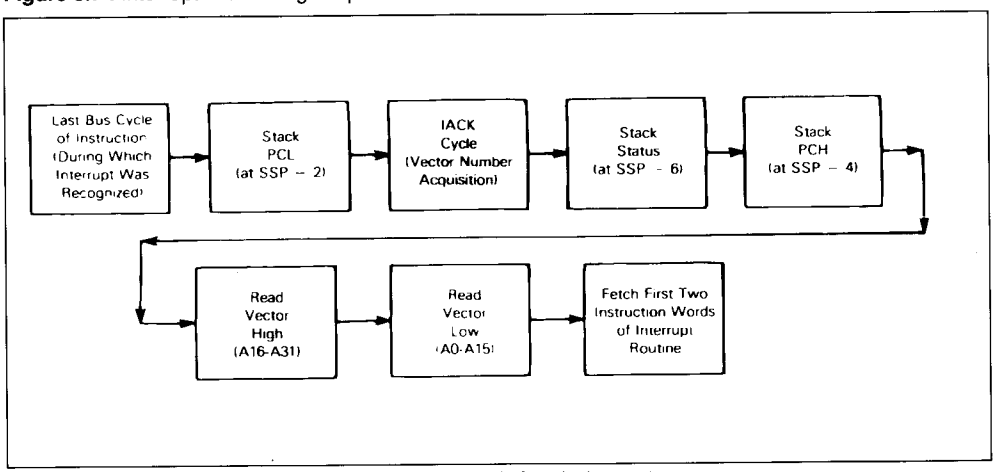
* Although a vector number is one byte, both data stobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D8 through D15 at this time.

Figure 5.6 : Interrupt Acknowledge Cycle Timing Diagram.



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

Figure 5.7 : Interrupt Processing Sequence.



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

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. SGS-THOMSON reserves the right to define instructions whose opcodes may be any of the illegal instructions. Three bit patterns will always force an illegal instruction trap on all TS68000 Family compatible microprocessors. They are : $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for SGS-THOMSON Microelectronics system products. The third pattern, S4AFC, is reserved for customer use.

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

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 TS68000 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 trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus error, or address error exception. If the instruction is indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If, during the execution of the instruction an exception is forced by that instruction, the forced exception is processed before the trace exception.

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

5.3.9. BUS ERROR. Bus error exceptions occur when the external logic requests that a bus error be processed by an exception. The current bus cycle which the processor is making is then aborted. 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.

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 destroy any memory contents. Only the 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 bys cycle followed by exception processing.

Figure 5.8 : Exception Stack Order (group 0).

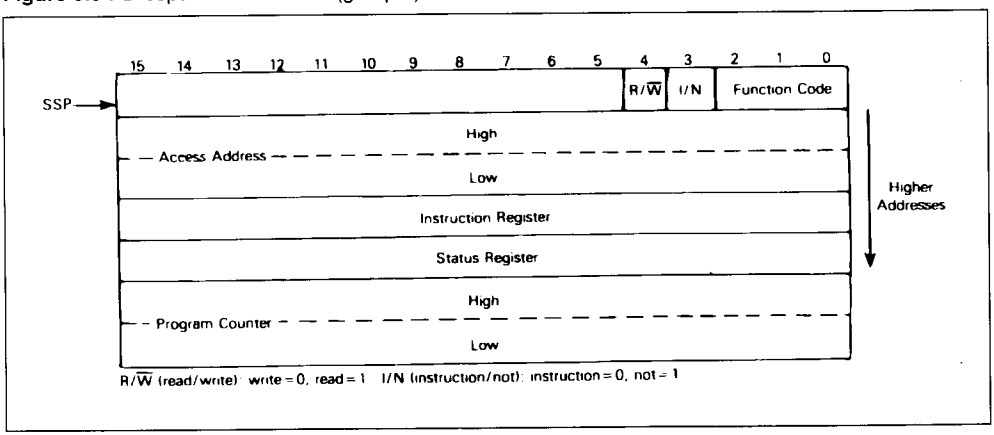
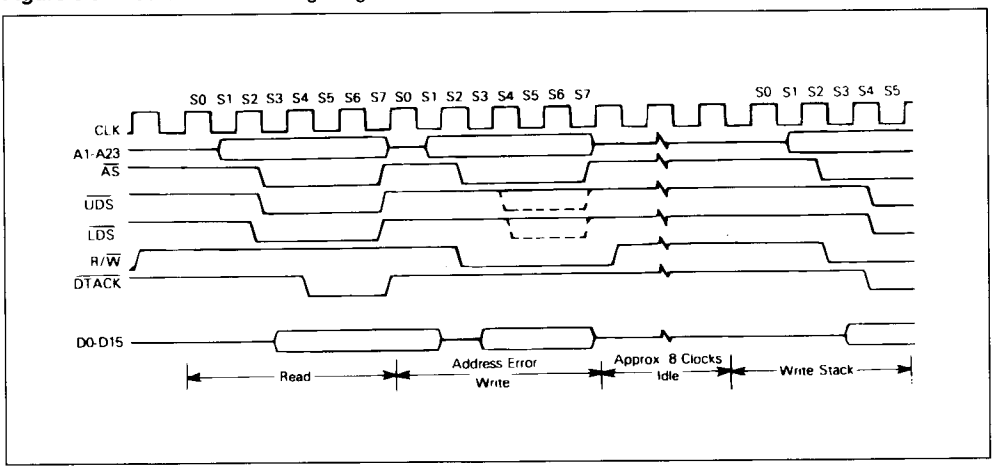


Figure 5.9 : Address Error Timing Diagram.



SECTION 6

INTERFACE WITH EF6800 PERIPHERALS

SGS-THOMSON Microelectronics extensive line of EF6800 peripherals are directly compatible with the TS68000. Some of these devices that are particularly useful are :

EF6821 Peripheral Interface Adapter

EF6840 Programmable Timer Module

EF9345, EF9367 CRT Controllers

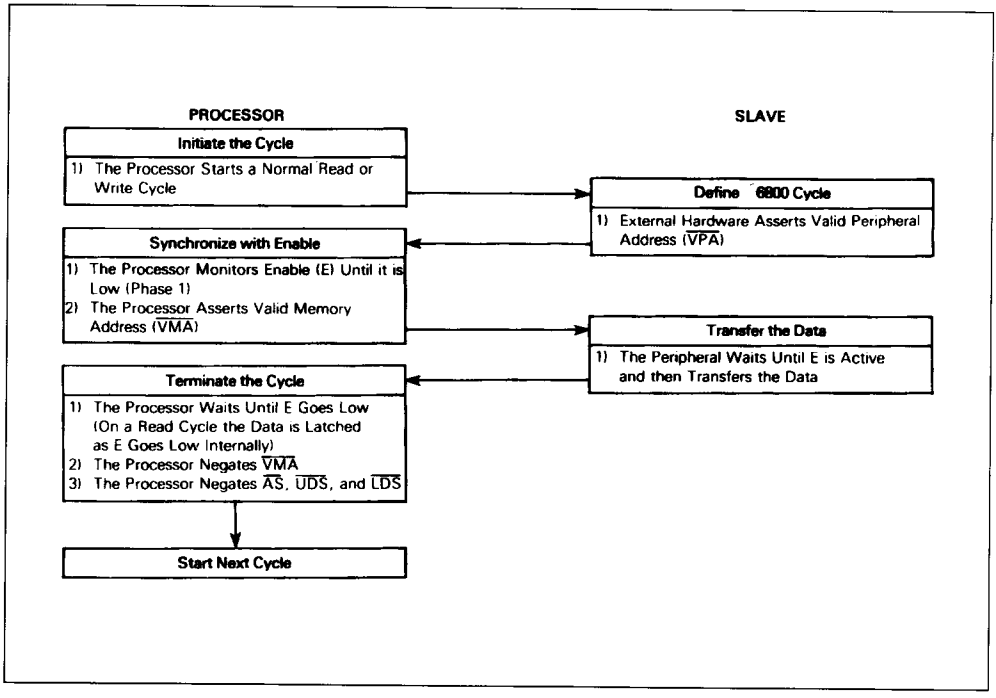
EF6850 Asynchronous Communications Interface Adapter

EF6852 Synchronous Serial Data Adapter

EF6854 Advanced Data Link Controller

To interface the synchronous EF6800 peripherals with the asynchronous TS68000, the processor modifies its bus cycle to meet the EF6800 cycle requirements whenever an EF6800 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 EF6800 devices.

Figure 6.1 : EF6800 Interfacing Flowchart.



6.1. DATA TRANSFER OPERATION

Three signals on the processor provide the EF6800 interface. They are : enable (E), valid memory address ($\overline{VMA}$), and valid peripheral address (VPA). Enable corresponds to the E or phase 2 signal in existing 6800 systems. The bus frequency in one tenth of the incoming TS68000 clock frequency. The timing of E allows 1 megahertz peripherals to be used with 8 megahertz TS68000s. 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 VPA accesses on successive E pulses.

EF6800 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 : TS68000 to EF6800 Peripheral Timing – Best Case.

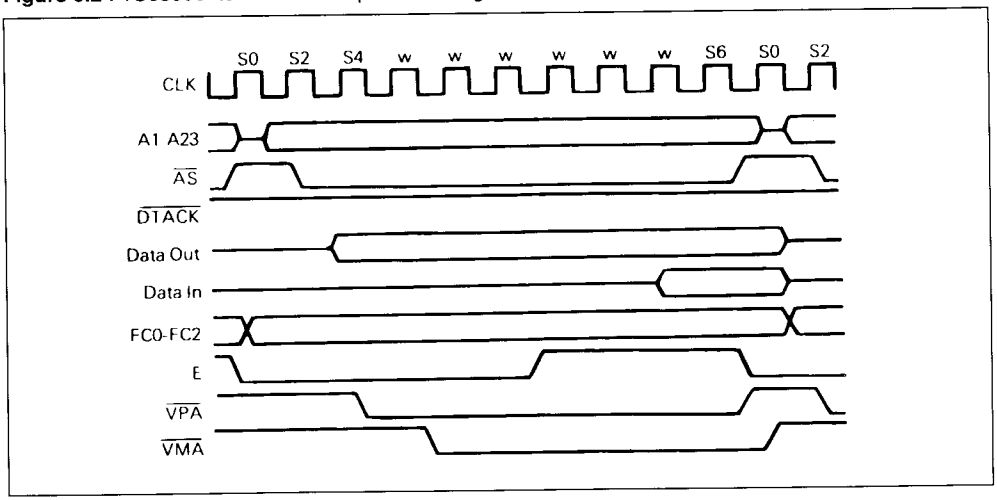
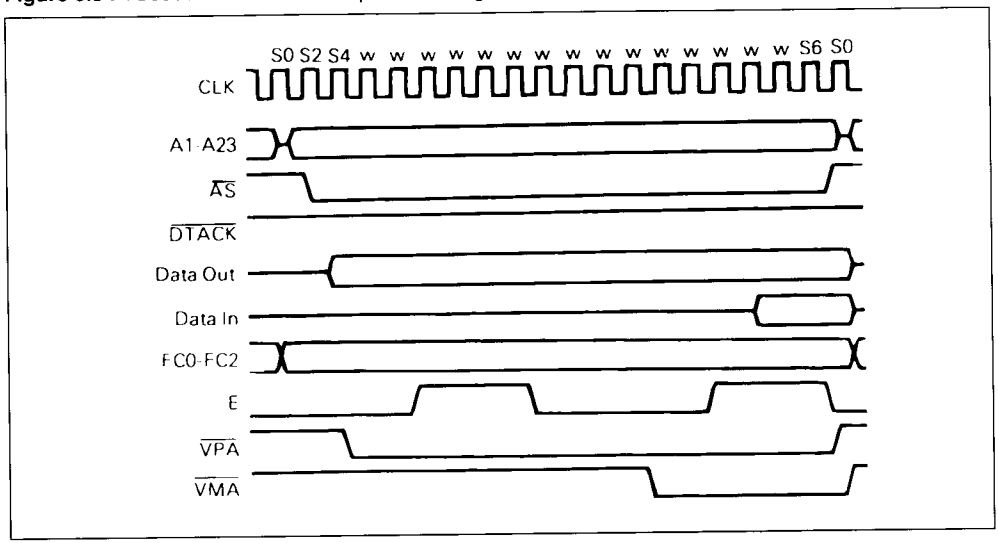


Figure 6.3 : TS68000 to EF6800 Peripheral Timing – Worst Case.



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 valid data on the data bus. The processor now inserts wait states until it recognizes the assertion of $\overline{VPA}$.

The $\overline{VPA}$ input signals the processor that the address on the bus is the address of an EF6800 device (or an area reserved for EF6800 devices) and that the bus should conform to the phase 2 transfer characteristics of the EF6800 bus. Valid peripheral address is derived by decoding the address bus, conditioned by the address strobe. Chip select for the EF6800 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 EF6800 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 EF6800 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-impedence state.

During a write cycle, the data bus is put in the high-impedence state and the read/write signal is switched high. The peripheral logic must remove VPA within one clock after the address strobe is negated.

$\overline{DTACK}$ should not be asserted while $\overline{VPA}$ is asserted. Notice that the TS68000 $\overline{VMA}$ is active low, contrasted with the active high EF6800 VMA. This allows the processor to put its buses in the high-impedance state on DMA requests without inadvertently selecting the peripherals.

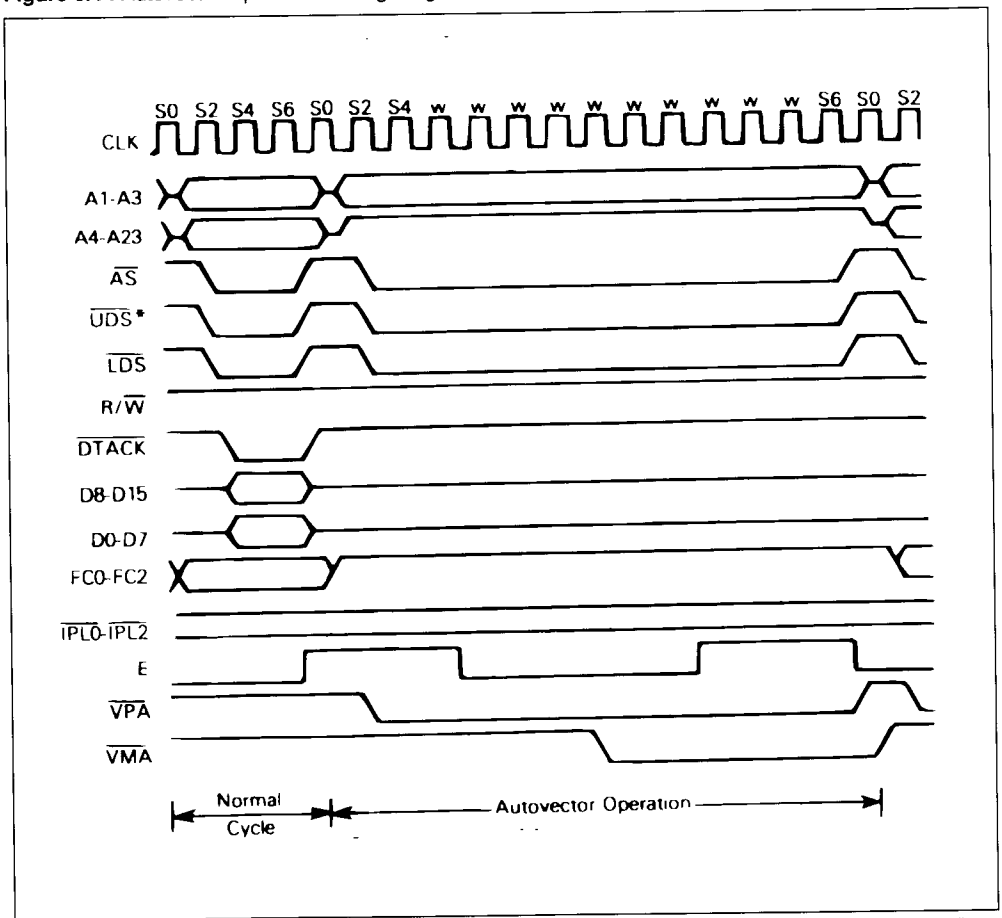
6.2. INTERRUPT INTERFACE OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector, the $\overline{VPA}$ is asserted, the TS68000 will assert $\overline{VMA}$ and complete a normal EF6800 read cycle as shown in figure 6.4. The processor will then use an internally generated vector that is a function of the interrupt being serviced. This process is known as autovectoring. The seven autovectors are vector numbers 25 through 31 (decimal).

Autovectoring operates in the same fashion (but is not restricted to) the EF6800 interrupt sequence. The basic difference is that there are six normalinterrupt vectors and one NMI type vector. As with both the EF6800 and the TS68000'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 that 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 bus during the autovector cycle. The vector number is generated internally.

SECTION 7

INSTRUCTION SET AND EXECUTION TIMES

7.1. INSTRUCTION SET

The following paragraphs provide information about the addressing categories and instruction set of the TS68000.

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	X	–	–	X
An	001	Register Number	–	–	–	X
(An)	010	Register Number	X	X	X	X
(An) +	011	Register Number	X	X	–	X
– (An)	100	Register Number	X	X	–	X
d(An)	101	Register Number	X	X	X	X
d(An, ix)	110	Register Number	X	X	X	X
xxx. W	111	000	X	X	X	X
xxx. L	111	001	X	X	X	X
d(PC)	111	010	X	X	X	–
d(PC, ix)	111	011	X	X	X	–
#xxx	111	X	X	X	–	–

Table 7.2 : Instruction Set.

Mnemonic	Description	Operation	Conditions 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	–	–	*	–	–
BSR	Branch to Subroutine	PC → – (SP) ; PC + d → PC	–	–	–	–	–
BTST	Test a Bit	~(<bit number>) OF Destination → Z	–	–	*	–	–
CHK	Check Register against Bounds	If Dn < 0 or Dn > (<ea>) then TRAP	–	*	U	U	U
CLR	Clear and Operand	0 → Destination	–	0	1	0	0
CMP	Compare	(destination) – (source)	–	*	*	*	*
CMPA	Compare Address	(destination) – (source)	–	*	*	*	*
CMPI	Compare Immediate	(destination) – Immediate Data	–	*	*	*	*
CMPM	Compare Memory	(destination) – (source)	–	*	*	*	*
DBCC	Test Condition, Decrement and Branch	If ~CC then Dn – 1 → Dn ; Dn ≠ – 1 then PC + d → PC	–	–	–	–	–
DIVS	Signed Divide	(destination)/(source) → Destiantion	–	*	*	*	0
DIVU	Unsigned Divide	(destination)/(source) → Destination	–	*	*	*	0
EOR	Exclusive OR Logical	(destination) ⊕ (source) → Destination	–	*	*	0	0
EORI	Exclusive OR Immediate	(destination) ⊕ Immediate Data → Destination	–	*	*	0	0
EORI to CCR	Exclusive OR Immediate to Condition Codes	(source) ⊕ CCR → CCR	*	*	*	*	*
EORI to SR	Exclusive OR Immediate to Status Register	(source) ⊕ SR → SR	*	*	*	*	*
EXG	Exchange Register	Rx ↔ Ry	–	–	–	–	–
EXT	Sign Extend	(destination) Sign-extended → Destination	–	*	*	0	0

∧ logical AND
ς logical OR
⊕ logical exclusive OR
~ logical complement

* affected
– unaffected
0 cleared
1 set
U undefined

Table 7.2 : Instruction Set (continued).

Mnemonic	Description	Operation	Conditions Codes				
			X	N	Z	V	C
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	–	–	–	–	–
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) X (source) → Destination	–	*	*	0	0
MULU	Unsigned Multiply	(destination) X (source) → Destination	–	*	*	0	0
NBCD	Negate Decimal with Extend	0 – (destination)$_{10}$ – X → Destination	*	U	*	U	*
NEG	Negate	0 – (destination) → Destination	*	*	*	*	*
NEGX	Negate with Extend	0 – (destination) – X → Destination	*	*	*	*	*
NOP	No Operation	–	–	–	–	–	–
NOT	Logical Complement	~(destination) → Destination	–	*	*	0	0
OR	Inclusive OR Logical	(destination) v (source) → Destination	–	*	*	0	0
ORI	Inclusive OR Immediate	(destination) v Immediate Data → Destination	–	*	*	0	0
ORI to CCR	Inclusive OR Immediate to Condition Codes	(source) v CCR → CCR	*	*	*	*	*
ORI to SR	Inclusive OR Immediate to Status Register	(source) v SR → SR	*	*	*	*	*
PEA	Push Effective Address	<ea> → – (SP)	–	–	–	–	–
RESET	Reset External Device	–	–	–	–	–	–

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

* affected
– unaffected
0 cleared
1 set
U undefined

Table 7.2 : Instruction Set (continued).

Mnemonic	Description	Operation	Conditions Codes				
			X	N	Z	V	C
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	$(destination)_{10} - (source)_{10} - X \rightarrow$ 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 Resgister Halves	Register [13:16] + Register [15:0]	–	*	*	0	0
TAS	Test and Set an Operand	(destination) tested → CC, 1 → [7] OF Destination	–	*	*	0	0
TRAP	Trap	PC → – (SSp) ; SR → – (SSP) ; (vector) → PC	–	–	–	–	–
TRAPV	Trap on Overflow	If V then TRAP	–	–	–	–	–
TST	Test and Operand	(destination) tested → CC	–	*	*	0	0
UNLK	Unlink	An → SP ; (SP) + → An	–	–	–	–	–

[ ] = bit number
Λ logical AND
ς logical OR
⊕ logical exclusive OR
– logical complement

* affected
– unaffected
0 cleared
1 set
U undefined

7.1.2. INSTRUCTION PREFETCH. The TS68000 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 read and write cycle times are four clock periods. Any wait states caused by a longer memory cycle must be added to the total instruction time. The number of bus read and write cycles for each instruction is also included with the timing data. This 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.

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.

7.2.2. MOVE INSTRUCTION EXECUTION TIMES. Tables 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 7.3 : Effective Address Calculation Times.

Addressing Mode		Byte, Word	Long
	Register		
Dn	Data Register Direct	**0**(0/0)	**0**(0/0)
An	Address Register Direct	**0**(0/0)	**0**(0/0)
	Memory		
(An)	Address Register Indirect	**4**(1/0)	**8**(2/0)
(An)+	Address Register Indirect with Postincrement	**4**(1/0)	**8**(2/0)
–(An)	Address Register Indirect with Predecrement	**6**(1/0)	**10**(2/0)
d(An)	Address Register Indirect with Displacement	**8**(2/0)	**12**(3/0)
d(AN, ix)*	Address Register Indirect with Index	**10**(2/0)	**14**(3/0)
xxx W	Absolute Short	**8**(2/0)	**12**(3/0)
xxx L	Absolute Long	**12**(3/0)	**16**(4/0)
d(PC)	Program Counter with Displacement	**8**(2/0)	**12**(3/0)
d(PC, ix)*	Program Counter with Index	**10**(2/0)	**14**(3/0)
#xxx	Immediate	**4**(1/0)	**8**(2/0)

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

Table 7.4 : Move Byte and Word Instruction Execution Times.

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

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

Table 7.5 : Move Long Instruction Execution Times.

Source	Destination								
	Dn	An	(An)	(An)+	– (An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
(An)	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
(An)+	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
– (An)	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
d(An)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
d(An, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
xxxW	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)
xxxL	20(5/0)	20(5/0)	28(5/2)	28(5/2)	28(5/2)	32(6/2)	34(6/2)	32(6/2)	36(7/2)
d(PC)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(5/2)
d(PC, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
#xxx	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)

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

Table 7.6 : Standard Instruction Execution Times.

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

Notes : + add effective address calculation time
† word or long 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 TS68000 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 >
MULU : n = concatanate the < ea > with a zero as the LSB ; n is the resultant number of 10 or 01 patterns in the 17-bit source ; i.e., worst case happens when the source is $5555.

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 meanings : An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand

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, and SR = status register.

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.

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.

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 parethesis 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 7.7 : Immediate Instruction Execution Times.

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte, Word	**8**(2/0)	–	**12**(2/1) +
	Long	**16**(3/0)	–	**20**(3/2) +
ADDO	Byte, Word	**4**(1/0)	**8**(1/1)*	**8**(1/1)+
	Long	**8**(1/0)	**8**(1/0)	**12**(1/2)+
ANDI	Byte, Word	**8**(2/0)	–	**12**(2/1)+
	Long	**16**(3/0)	–	**20**(3/1)+
CMPI	Byte, Word	**8**(2/0)	–	**8**(2/0)+
	Long	**14**(3/0)	–	**12**(3/0)+
EORI	Byte, Word	**8**(2/0)	–	**12**(2/1)+
	Long	**16**(3/0)	–	**20**(3/2)+
MOVEQ	Long	**4**(1/0)	–	–
ORI	Byte, Word	**8**(2/0)	–	**12**(2/1)+
	Long	**16**(3/0)	–	**20**(3/2)+
SUBI	Byte, Word	**8**(2/0)	–	**12**(2/1)+
	Long	**16**(3/0)	–	**20**(3/2)+
SUBQ	Byte, Word	41/0)	**8**(1/0)*	**8**(1/1)+
	Long	**8**(1/0)	**8**(1/0)	**12**(1/2)+

+ add effective address calculation time
* word only

Table 7.8 : Single Operand Instruction Execution Times.

Instruction	Size	Register	Memory
CLR	Byte, Word	**4**(1/0)	**8**(1/1)+
	Long	**6**(1/0)	**12**(1/2)+
NBCD	Byte	**6**(1/0)	**8**(1/1)+
NEG	Byte, Word	**4**(1/0)	**8**(1/1)+
	Long	**6**(1/0)	**12**(1/1)+
NEGX	Byte, Word	**4**(1/0)	**8**(1/2)+
	Long	**6**(1/0)	**12**(1/2)+
NOT	Byte, Word	**4**(1/0)	**8**(1/1)+
	Long	**6**(1/0)	**12**(1/2)+
S_{cc}	Byte, False	**4**(1/0)	**8**(1/1)+
	Byte, True	**6**(1/0)	**8**(1/2)+
TAS	Byte	**4**(1/0)	**10**(1/1)+
TST	Byte, Word	**4**(1/0)	**4**(1/0)+
	Long	**4**(1/0)	**4**(1/0)+

+ add effective address calculation time

Table 7.9 : Shift/rotate Instruction Execution Times.

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

+ add effective address calculation time

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	**8**(1/0)*	–	**12**(2/0)*	–
BCLR	Byte	–	**8**(1/1)+	–	**12**(2/1)+
	Long	**10**(1/0)*	–	**14**(2/0)*	–
BSET	Byte	–	**8**(1/1)+	–	**12**(2/1)+
	Long	**8**(1/0)*	–	**12**(2/0)*	–
BTST	Byte	–	**4**(1/0)+	–	**8**(2/0)+
	Long	**6**(1/0)	–	**10**(2/0)	–

+ 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
B_{CC}	Byte	**10**(2/0)	**8**(1/0)
	Word	**10**(2/0)	**12**(2/0)
BRA	Byte	**10**(2/0)	–
	Word	**10**(2/0)	–
BSR	Byte	**18**(2/2)	–
	Word	**18**(2/2)	–
DB_{CC}	CC True	–	**12**(2/0)
	CC False	**10**(2/0)	**14**(3/0)

+ add effective address calculation time
* indicates maximum value

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)	d(An)	d(An,ix)+	xxx.W	xxx.L	d(PC)	d(PC, ix)*
JMP	–	**8**(2/0)	–	–	**10**(2/0)	**14**(3/0)	**10**(2/0)	**12**(3/0)	**10**(2/0)	**14**(3/0)
JSR	–	**16**(2/2)	–	–	**18**(2/2)	**22**(2/2)	**18**(2/2)	**20**(3/2)	**18**(2/2)	**22**(2/2)
LEA	–	**4**(1/0)	–	–	**8**(2/0)	**12**(2/0)	**8**(2/0)	**12**(3/0)	**8**(2/0)	**12**(2/0)
PEA	–	**12**(1/2)	–	–	**16**(2/2)	**20**(2/2)	**16**(2/2)	**20**(3/2)	**16**(2/2)	**20**(2/2)
MOVEM M → R	Word	**12 + 4n** (3 + n/0)	**12 + 4n** (3 + n/0)	–	**16 + 4n** (4 + n/0)	**18 + 4n** (4 + n/0)	**16 + 4n** (4 + n/0)	**20 + 4n** (5 + n/0)	**16 + 4n** (4 + n/0)	**18 + 4n** (4 + n/0)
	Long	**12 + 8n** (3 + 2n/0)	**12 + 8n** (3 + 2n/0)	–	**16 + 8n** (4 + 2n/0)	**18 + 8n** (4 + 2n/0)	**16 + 8n** (4 + 2n/0)	**20 + 8n** (5 + 2n/0)	**16 + 8n** (4 + 2n/0)	**18 + 8n** (4 + 2n/0)
MOVEM R → M	Word	**8 + 4n** (2/n)	–	**8 + 4n** (2/n)	**12 + 4n** (3/n)	**14 + 4n** (3/n)	**12 + 4n** (3/n)	**16 + 4n** (4/n)	– –	– –
	Long	**8 + 8n** (2/2n)	–	**8 + 8n** (2/2n)	**12 + 8n** (3/2n)	**14 + 8n** (3/2n)	**12 + 8n** (3/2n)	**16 + 8n** (4/2n)	– –	– –

n is the number of registers to move
* is the size of the index register (ix) 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 meanings : Dn = data register operand and 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	**8**(1/0)	**30**(5/2)
CMPM	Byte, Word		**12**(3/0)
	Long		**20**(5/0)
SUBX	Byte, Word	**4**(1/0)	**18**(3/1)
	Long	**8**(1/0)	**30**(5/2)
ABCD	Byte	**6**(1/0)	**18**(3/1)
SBCD	Byte	**6**(1/0)	**18**(3/1)

7.2.11. MISCELLANEOUS INSTRUCTION EXECUTION TIMES. Tables 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**(2/0)	**12**(2/0)+
MOVE to SR	–	**12**(2/0)	**12**(2/0)+
EXG	–	**6**(1/0)	–
EXT	Word	**4**(1/0)	–
	Long	**4**(1/0)	–
LINK	–	**16**(2/2)	–
MOVE from USP	–	**4**(1/0)	–
MOVE to USP	–	**4**(1/0)	–
NOP	–	**4**(1/0)	–
RESET	–	**132**(1/0)	–
RTE	–	**20**(5/0)	–
RTR	–	**20**(5/0)	–
RTS	–	**16**(4/0)	–
STOP	–	**4**(0/0)	–
SWAP	–	**4**(1/0)	–
TRAPV	–	**4**(1/0)	–
UNLK	–	**12**(3/0)	–

+ 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	**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	**44**(5/4)+
Divide by Zero	**42**(5/4)
Illegal Instruction	**34**(4/3)
Interrupt	**44**(5/3)*
Privilege Violation	**34**(4/3)
$\overline{\text{RESET}}$**	**40**(6/0)
Trace	**34**(4/3)
TRAP Instruction	**38**(4/4)
TRAPV Instruction	**34**(4/3)

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

SECTION 8

ELECTRICAL SPECIFICATIONS

This section contains electrical specifications and associated timing information for the TS68000. These specifications represent an improvement over previously published specifications for the 8,10,12.5, 16 MHz.TS68000 and are valid only for products bearing date codes of 8922 and later.

8.1. ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{CC}	Supply Voltage	– 0.3 to 7	V
V_{IN}	Input Voltage	– 0.3 to 7	V
T_A	Operating Temperature Range TS68000C TS68000V TS68000M	T_L to T_H 0 to 70 – 40 to 85 – 55 to 125	°C
T_{stg}	Storage Temperature	– 55 to 150	°C

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

8.2. THERMAL DATA

Package	T_A Range	θ_{JA} (°C/W)	P_D (W) @ T_A Min.	T_J (°C) @ T_A Min.	P_D (W) @ T_A Max.	T_J (°C) @ T_A Max.
Plastic DIL	0 °C to 70 °C	30	1.2	36	1.0	100
PLCC	0 °C to 70 °C	45	1.2	54	1.0	115
Ceramic PGA	0 °C to 70 °C	33	1.2	40	1.0	103
	0 °C to 85 °C	33	1.2	40	1.0	118
	– 40 °C to 85 °C	33	1.5	10	1.0	118

8.3. DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 5 Vdc ± 5 % ; GND = 0 Vdc ; $T_A = T_L$ to T_H ; see figures 8.1, 8.2 and 8.3)

Symbol	Parameter		Min.	Max.	Unit
V_{IH}	Input High Voltage		2	V_{CC}	V
V_{IL}	Input Low Voltage		GND – 0.3	0.8	V
I_{IN}	Input Leakage Current @ 5.25 V	$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$	–	2.5	µA
		CLK, $\overline{IPL0}$-$\overline{IPL2}$, $\overline{VPA}$	–	20	
		$\overline{HALT}$, $\overline{RESET}$			
I_{TSI}	Three-state (off state) Input Current @ 2.4 V/0.4 V	$\overline{AS}$, A1-A23, D0-D15, FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	–	20	µA
V_{OH}	Output High Voltage (I_{OH} = – 400 µA)	E*	V_{CC} – 0.75	–	V
		E, $\overline{AS}$, A1-A23, $\overline{BG}$, D0-D15, FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	2.4	–	
V_{OL}	Output Low Voltage				
	(I_{OL} = 1.6 mA)	$\overline{HALT}$	–	0.5	V
	(I_{OL} = 3.2 mA)	A1-A23, $\overline{BG}$, FC0-FC2	–	0.5	
	(I_{OL} = 5 mA)	$\overline{RESET}$	–	0.5	
	(I_{OL} = 5.3 mA)	E, $\overline{AS}$, D0-D15, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	–	0.5	
P_D***	Power Dissipation (see 8.4 POWER CONSIDERATIONS)		–	–	W
C_{IN}	Capacitance (V_{in} = 0 V, T_A = 25 °C ; Frequency = 1 MHz)**		–	20	pF

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

Figure 8.1 : $\overline{\text{RESET}}$ Test Load.

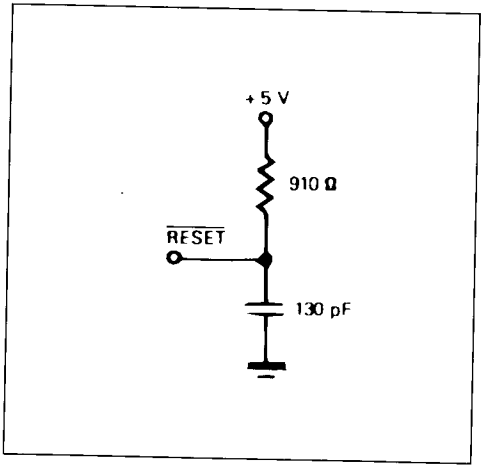


Figure 8.2 : $\overline{\text{HALT}}$ Test Load.

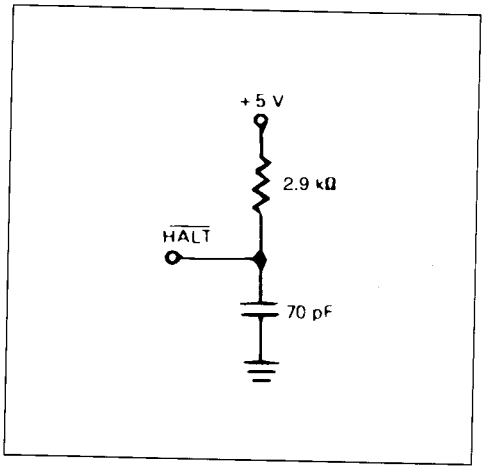
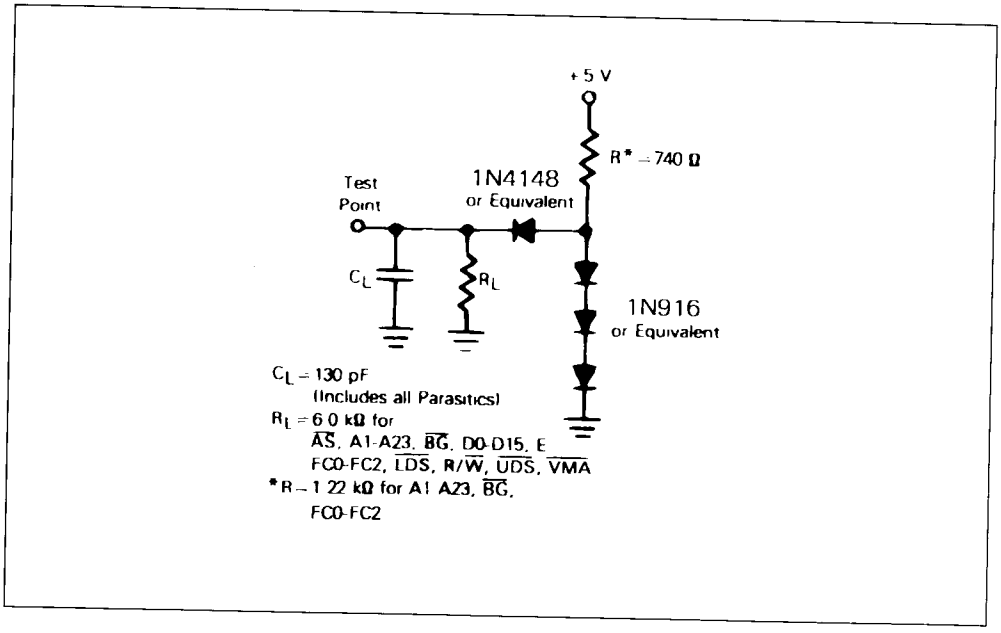


Figure 8.3 : Test Loads.



8.4. POWER CONSIDERATIONS

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

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

Where :

T_A = Ambient Temperature, °C

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

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

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

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

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

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

$P_D = K \div (T_J + 273°C)$

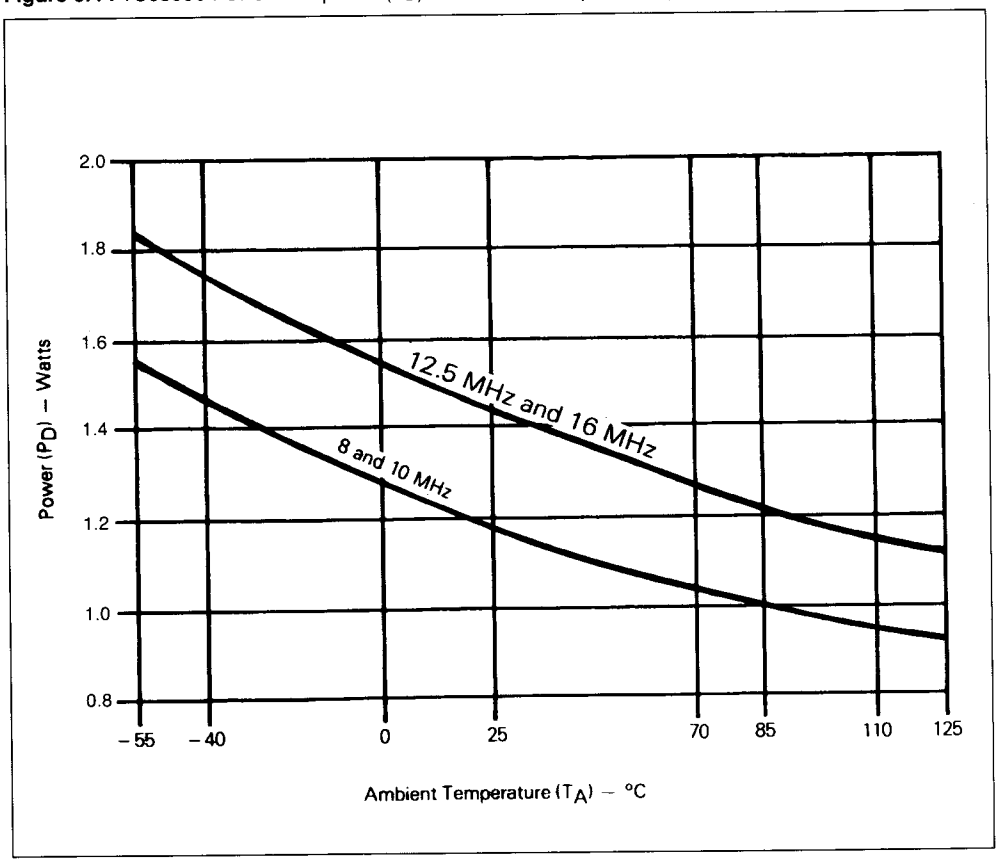
Solving equations 1 and 2 for K gives :

$K = PD \cdot (T_A + 273°C) + \theta_{JA} \cdot P_D^2$

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

Figure 8.4 illustrates the graphic solution to the equations, given above, for the specification power dissipations of 1.50 and 1.75 watts over the ambient temperature range of – 55°C to 125°C using an average θ_{JA} of 40°C/watt to represent the various TS68000 packages. However, actual θJA's in the range of 30°C to 50°C/watt only change the curves slightly.

Figure 8.4 : TS68000 Power Dissipation (P_D) vs Ambient Temperature (T_A).



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

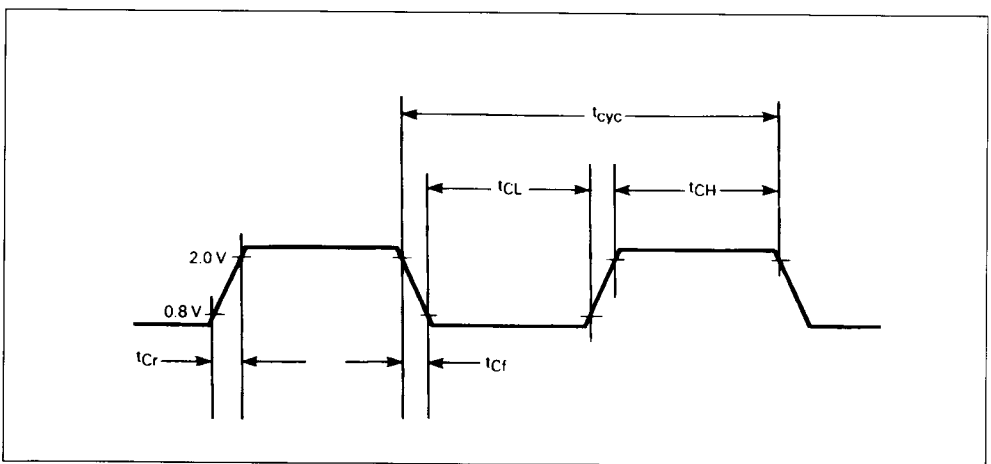
$\theta_{JA} = \theta_{JC} + \theta_{CA}$

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

8.5. AC ELECTRICAL SPECIFICATIONS – CLOCK TIMING (see figure 8-5)

Symbol	Parameter	8 MHz		10 MHz		12.5 MHz		16 MHz		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
f	Frequency of Operation	4	8	4	10	8	12.5	8	16.7	MHz
t_{cyc}	Cycle Time	125	250	100	250	80	125	60	125	ns
t_{CL} t_{CH}	Clock Pulse Width	55 55	125 125	45 45	125 125	35 35	62.5 62.5	25 25	62.5 62.5	ns
t_{Cr} t_{Cf}	Rise and Fall Times	– –	10 10	– –	10 10	– –	5 5	– –	5 5	ns

Figure 8.5 : Clock Input Timing Diagram.



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.

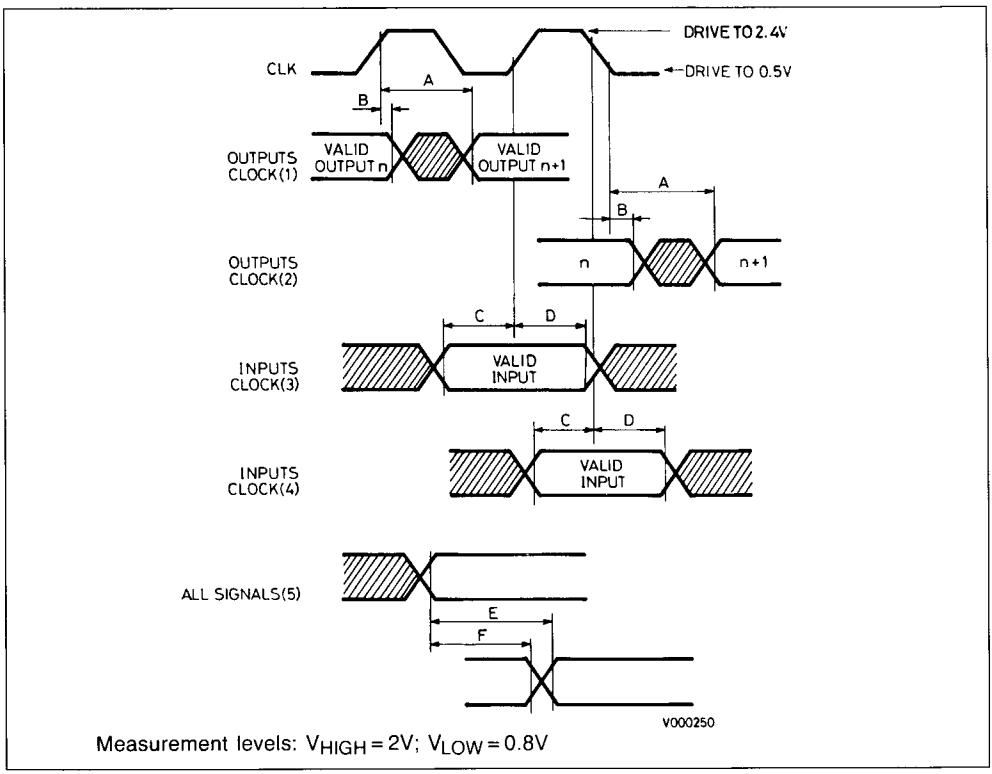
8.6. AC ELECTRICAL SPECIFICATION DEFINITIONS

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 clock and possibly to one or more other signals.

The measurement of the AC specifications is defined by the waveforms shown in figure 8.6 in order to test the parameters guaranteed by SGS-THOMSON. Inputs must be driven to the voltage levels specified in the figure 8.6. Outputs are specified with minimum and/or maximum limits, as appropriate, and are measured as shown in the same figure. Inputs are specified with minimum setup and hold times, and are measured as shown. Finally the measurement for signal-to-signal specifications are also shown.

Note that 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 characteristics.

Figure 8.6 : Drive Levels and Test Points for AC Specifications.



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 (max. or min.).
F. Signal valid to signal invalid specification (max. or min.).

8.7. AC ELECTRICAL SPECIFICATIONS – READ CYCLES

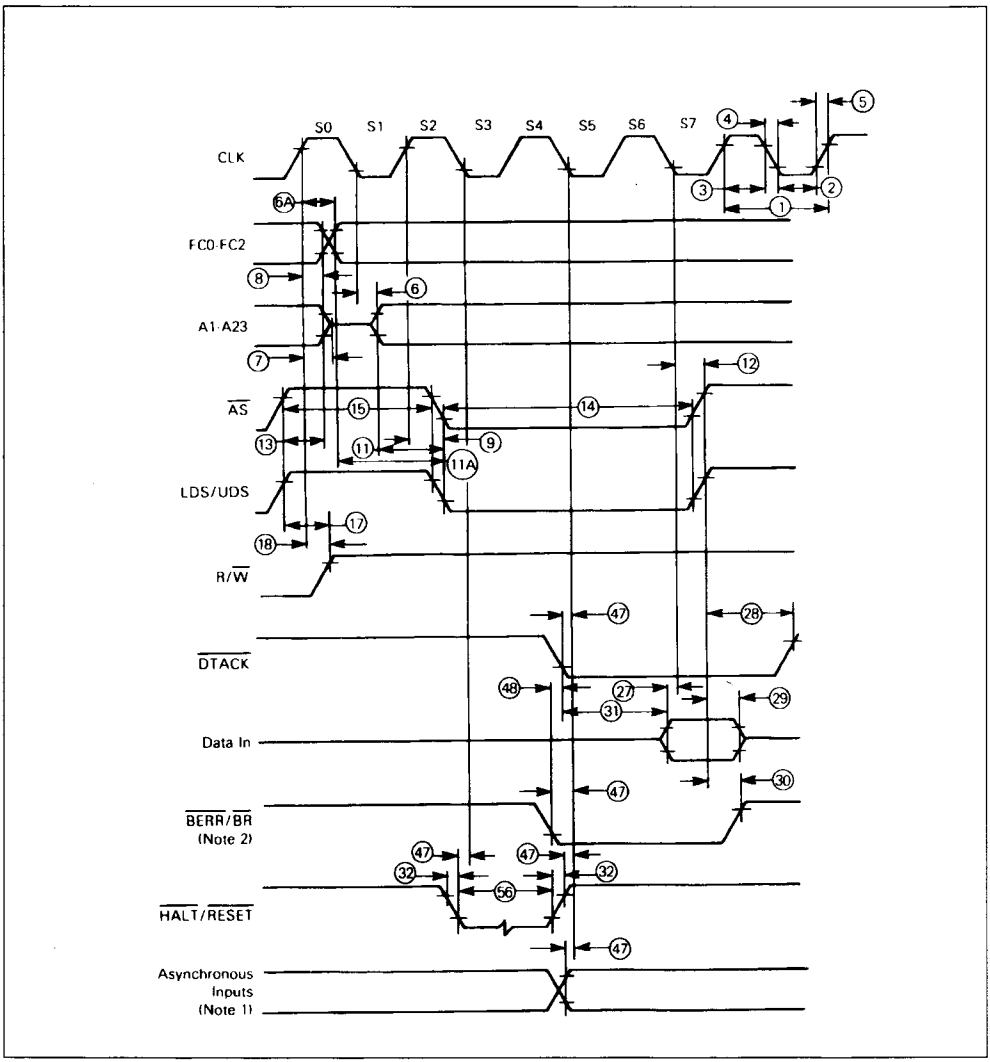
(V_{CC} = 5Vdc ± 5% ; GND = 0Vdc ; $T_A = T_L$ to T_H ; see figure 8.7)

N°	Symbol	Parameter	8 MHz		10 MHz		12.5 MHz		16 MHz		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
1	t_{CYC}	Clock Period	125	250	100	250	80	125	60	125	ns
2	t_{CL}	Clock Width Low	55	125	45	125	35	62	25	62	ns
3	t_{CH}	Clock Width High	55	125	45	125	35	62	25	62	ns
4	t_{Cf}	Clock Fall Time	–	10	–	10	–	5	–	5	ns
5	t_{Cr}	Clock Rise Time	–	10	–	10	–	5	–	5	ns
6	t_{CLAV}	Clock Low to Address Valid	–	60	–	50	–	50	–	40	ns
6A	t_{CHFCV}	Clock High to FC Valid	–	60	–	50	–	50	–	40	ns
7	t_{CHADZ}	Clock High to Address, Data Bus High Impedance (maximum)	–	70	–	60	–	50	–	50	ns
8	t_{CHAFI}	Clock High to Address, FC Invalid (minimum)	0	–	0	–	0	–	0	–	ns
9(1)	t_{CHSL}	Clock High to $\overline{AS}$, $\overline{DS}$ Low	0	60	0	55	0	50	0	45	ns
11(2)	t_{AVSL}	Address Valid to $\overline{AS}$, $\overline{DS}$ Low	20	–	15	–	10	–	5	–	ns
11A(2)	t_{FCVSL}	FC Valid to $\overline{AS}$, $\overline{DS}$ Low	50	–	40	–	35	–	30	–	ns
12(1)	t_{CLSH}	Clock Low to $\overline{AS}$, $\overline{DS}$ High	–	50	–	45	–	40	–	30	ns
13(2)	t_{SHAFI}	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	30	–	20	–	10	–	10	–	ns
14(2)	t_{SL}	$\overline{AS}$, $\overline{DS}$ Width Low	240	–	195	–	160	–	120	–	ns
15(2)	t_{SH}	$\overline{AS}$, $\overline{DS}$ Width High	150	–	105	–	65	–	50	–	ns
17(2)	t_{SHRH}	$\overline{AS}$, $\overline{DS}$ High to R/$\overline{W}$ High	40	–	20	–	10	–	10	–	ns
18(1)	t_{CHRH}	Clock High to R/$\overline{W}$ High	0	55	0	45	0	40	0	45	ns
27(5)	t_{DICL}	Data in to Clock low (setup time)	15	–	10	–	10	–	10	–	ns
28(2)	t_{SHDAH}	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High	0	240	0	190	0	150	0	80	ns
29	t_{SHDII}	$\overline{AS}$, $\overline{DS}$ High to Data In Invalid (hold time)	0	–	0	–	0	–	0	–	ns
30	t_{SHBEH}	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	0	–	0	–	0	–	0	–	ns
31(2.5)	t_{DALDI}	$\overline{DTACK}$ Low to Data in (setup time)	–	70	–	50	–	40	–	30	ns
32	$t_{RHr,f}$	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	0	200	0	200	0	150	0	150	ns
47(5)	t_{ASI}	Asynchronous Input Setup Time	20	–	20	–	10	–	10	–	ns
48(3)	t_{BELDAL}	$\overline{BERR}$ Low to $\overline{DTACK}$ Low	20	–	20	–	10	–	10	–	ns
56(4)	t_{HRPW}	$\overline{HALT}$/$\overline{RESET}$ Pulse Width	10	–	10	–	10	–	10	–	Clk.Per.

Notes :
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. Actual value depends on clock period.
3. If 47 is satisfied for both $\overline{DTACK}$ and $\overline{BERR}$, 48 may be 0 nanosecond.
4. For power up, the MPU must be held in $\overline{RESET}$ state for 100 milliseconds to allow stabilization of on-chip circuitry. After the system is powered up, 56 refers to the minimum pulse width required to reset the system.
5. If the asynchronous setup time (47) requirements are satisfied, the $\overline{DTACK}$ low-to-data setup time (31) requirement can be ignored. The data must only satisfy the data-in to clock-low setup time (27) for the following cycle.

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

Figure 8.7 : Read Cycle Timing Diagram.



Notes : 1. Setup time for the synchronous inputs BGACK, IPL0-2, and VPA guarantees their recognition at the next falling edge of the clock
2. BR need fall at this time only in order to insure being recognized at the end of this bus cycle
3. Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage 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.

8.8. AC ELECTRICAL SPECIFICATIONS – WRITE CYCLES

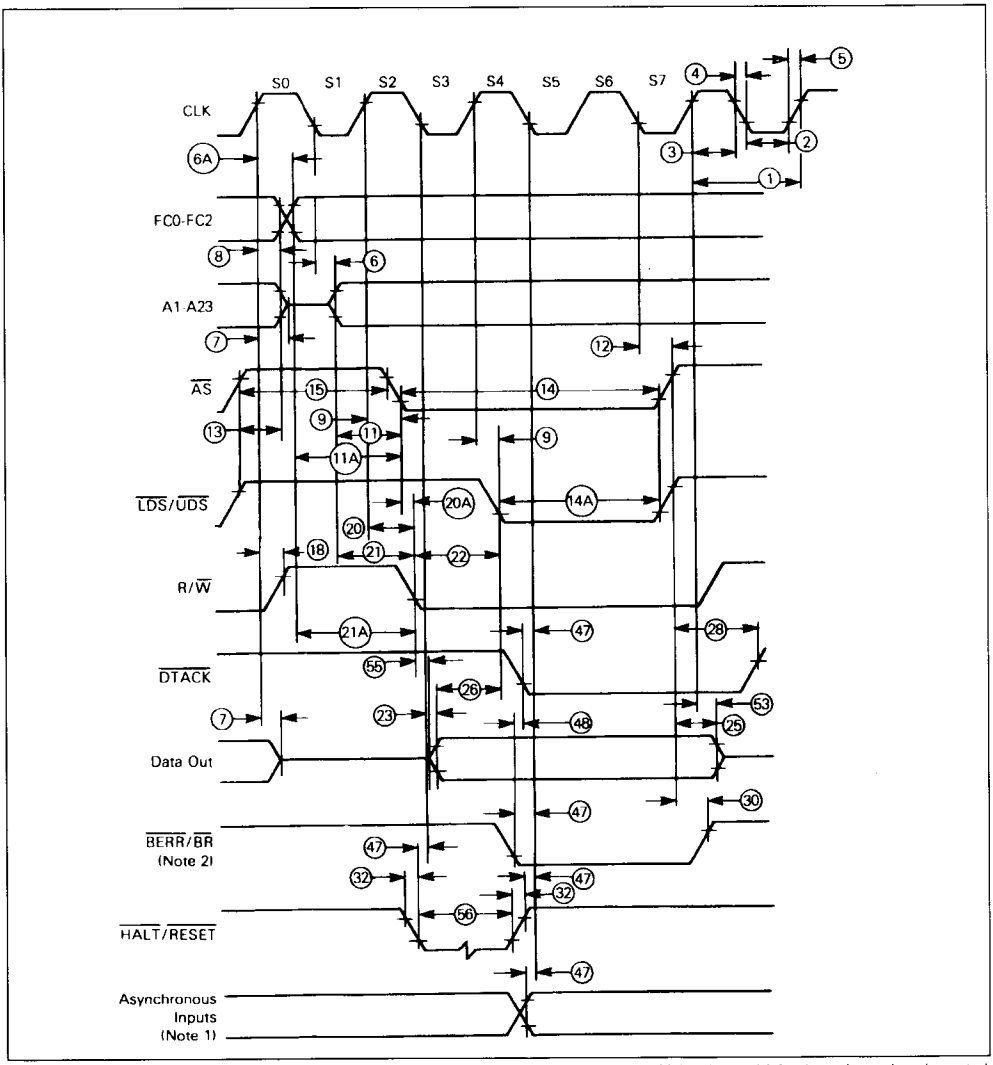
(V_{CC} = 5 Vdc ± 5 % ; GND = 0 Vdc ; $T_A = T_L$ to T_H ; see figure 8.8)

N°	Symbol	Parameter	8 MHz		10 MHz		12.5 MHz		16 MHz		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
1	t_{CYC}	Clock Period	125	250	100	250	80	125	60	125	ns
2	t_{CL}	Clock Width Low	55	125	45	125	35	62	25	62	ns
3	t_{CH}	Clock Width High	55	125	45	125	35	62	25	62	ns
4	t_{Cf}	Clock Fall Time	–	10	–	10	–	5	–	5	ns
5	t_{Cr}	Clock Rise Time	–	10	–	10	–	5	–	5	ns
6	t_{CLAV}	Clock Low to Address Valid	–	60	–	50	–	50	–	40	ns
6A	t_{CHFCV}	Clock High to FC Valid	–	60	–	50	–	50	–	40	ns
7	t_{CHADZ}	Clock High to Address, Data Bus High Impedance (maximum)	–	70	–	60	–	50	–	50	ns
8	t_{CHAFI}	Clock High to Address, FC Invalid (minimum)	0	–	0	–	0	–	0	–	ns
9(1)	t_{CHSL}	Clock High to $\overline{AS}$, $\overline{DS}$ Low	0	60	0	55	0	50	0	45	ns
11(2)	t_{AVSL}	Address Valid to $\overline{AS}$, $\overline{DS}$ Low	20	–	15	–	10	–	5	–	ns
11A(2)	t_{FCVSL}	FC Valid to $\overline{AS}$, Low	50	–	40	–	35	–	30	–	ns
12(1)	t_{CLSH}	Clock Low to $\overline{AS}$, $\overline{DS}$ High	–	50	–	45	–	40	–	30	ns
13(2)	t_{SHAFI}	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	30	–	20	–	10	–	10	–	ns
14(2)	t_{SL}	$\overline{AS}$ Low	240	–	195	–	160	–	120	–	ns
14A(2)	t_{DSL}	$\overline{DS}$ Width Low	115	–	95	–	80	–	60	–	ns
15(2)	t_{SH}	$\overline{AS}$, $\overline{DS}$ Width High	150	–	105	–	65	–	50	–	ns
18(1)	t_{CHRH}	Clock High to R/$\overline{W}$ High	0	55	0	45	0	40	0	40	ns
20(1)	t_{CHRL}	Clock High to R/$\overline{W}$ Low	–	55	–	45	–	40	–	30	ns
20A(6)	t_{ASRV}	$\overline{AS}$, Low to R/$\overline{W}$ Valid	–	20	–	20	–	20	–	20	ns
21(2)	t_{AVRL}	Address Valid to R/$\overline{W}$ Low	20	–	0	–	0	–	0	–	ns
21A(2)	t_{FCVRL}	FC Valid to R/$\overline{W}$ Low	60	–	50	–	30	–	20	–	ns
22(2)	t_{RLSL}	R/$\overline{W}$ Low to $\overline{DS}$ Low	80	–	50	–	30	–	20	–	ns
23	t_{CLDO}	Clock Low to Data out Valid	–	60	–	50	–	50	–	40	ns
25(2)	t_{SHDOI}	$\overline{AS}$, $\overline{DS}$ High to Data Out Invalid	30	–	20	–	15	–	10	–	ns
26(2)	t_{DOSL}	Data out Valid to $\overline{DS}$ Low	30	–	20	–	15	–	10	–	ns
28(2)	t_{SHDAH}	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ high	0	240	0	190	0	150	0	80	ns
30	t_{SHBEH}	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ high	0	–	0	–	0	–	0	–	ns
32	$t_{RHr,\,f}$	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	0	200	0	200	0	150	0	150	ns
47(5)	t_{ASI}	Asynchronous Input Setup Time	20	–	20	–	10	–	10	–	ns
48(3)	t_{BELDAL}	$\overline{BERR}$ Low to $\overline{DTACK}$ Low	20	–	20	–	20	–	10	–	ns
53	t_{CHDOI}	Clock High to Data Out Invalid	0	–	0	–	0	–	0	–	ns
55	t_{RLDBD}	R/$\overline{W}$ to Data Bus Driven	30	–	20	–	10	–	0	–	ns
56(4)	t_{HRPW}	$\overline{HALT}$/$\overline{RESET}$ Pulse Width	10	–	10	–	10	–	10	–	ns

Notes :
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. Actual value depends on clock period
3. If 47 is satisfied for both DTACK and $\overline{BERR}$, 48 may be 0 nanoseconds.
4. For power up, the MPU must be held in $\overline{RESET}$ state for 100 milliseconds to allow stabilization of on-chip circuitry. After the system is powered up, 56 refers to the minimum pulse width required to reset the system.
5. If the asynchronous setup time (47) requirements are satisfied, the DTACK low-to-data setup time (31) requirement can be ignored. The data must only satisfy the data-in to clock-low setup time (27) for the following cycle.
6. When $\overline{AS}$, and R/$\overline{W}$ are equally loaded (± 20%), Subtract 10 nanoseconds from the values in these columns.

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

Figure 8.8 : Write Cycle Timing Diagram.



Notes : 1. Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage of 2.0 volts, unless otherwise noted. 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.

2. Because of loading variations, R/$\overline{W}$ may be valid after $\overline{AS}$ even through both are initiated by the rising edge of S2 (specification 20A).

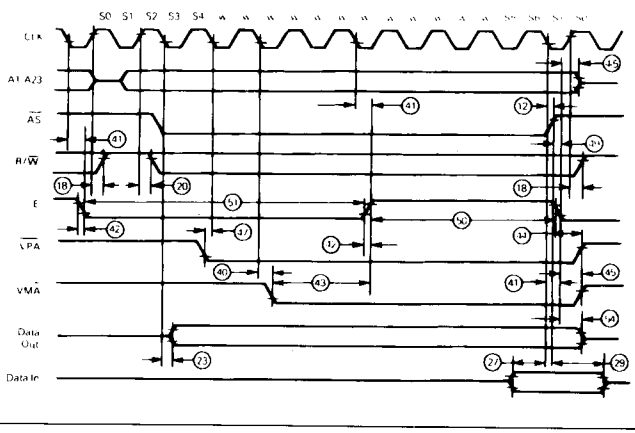
8.9. AC ELECTRICAL SPECIFICATIONS – TS68000 to 6800 PERIPHERAL

(V_{CC} = 5 Vdc ± 5 % ; GND = 0 Vdc ; $T_A = T_L$ to T_H ; refer to figures 8.9 and 8.10)

N°	Symbol	Parameter	8 MHz		10 MHz		12.5 MHz		16 MHz		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
12(1)	t_{CLSH}	Clock Low to $\overline{AS}$, $\overline{DS}$ High	–	50	–	45	–	40	–	30	ns
18(1)	t_{CHRH}	Clock High to R/$\overline{W}$ High	0	55	0	45	0	40	0	40	ns
20(1)	t_{CHRL}	Clock High to R/$\overline{W}$ Low (write)	–	55	–	45	–	40	–	30	ns
23	t_{CLDO}	Clock Low to Data out Valid (write)	–	60	–	50	–	50	–	40	ns
27(2)	t_{CLDO}	Data in to Clock Low (setup time on read)	15	–	10	–	10	–	10	–	ns
29	t_{SHDII}	$\overline{AS}$, $\overline{DS}$ high to Data in Invalid (hold time on read)	0	–	0	–	0	–	0	–	ns
40	t_{CLVML}	Clock Low to $\overline{VMA}$ Low	–	70	–	70	–	70	–	50	ns
41	t_{CLET}	Clock Low to E Transistion	–	55	–	45	–	35	–	35	ns
42	$t_{Er,f}$	E Output Rise and Fall Time	–	25	–	15	–	15	–	15	ns
43	t_{VMLEH}	$\overline{VMA}$ Low to E high	200	–	150	–	90	–	70	–	ns
44	t_{SHVPH}	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	0	120	0	90	0	70	0	50	ns
45	t_{ELCAI}	E Low to Control, Address Bus Invalid (address hold time)	30	–	10	–	10	–	10	–	ns
47(2)	t_{ASI}	Asynchronous Input Setup Time	20	–	20	–	10	–	10	–	ns
49(3)	t_{SHEL}	$\overline{AS}$, $\overline{DS}$ High to E Low	– 70	70	– 55	55	– 45	45	– 35	+ 35	ns
50	t_{EH}	E Width High	450	–	350	–	280	–	210	–	ns
51	t_{EL}	E Width low	700	–	550	–	440	–	330	–	ns
54	t_{ELDOI}	E Low to Data out Invalid	30	–	20	–	15	–	10	–	ns

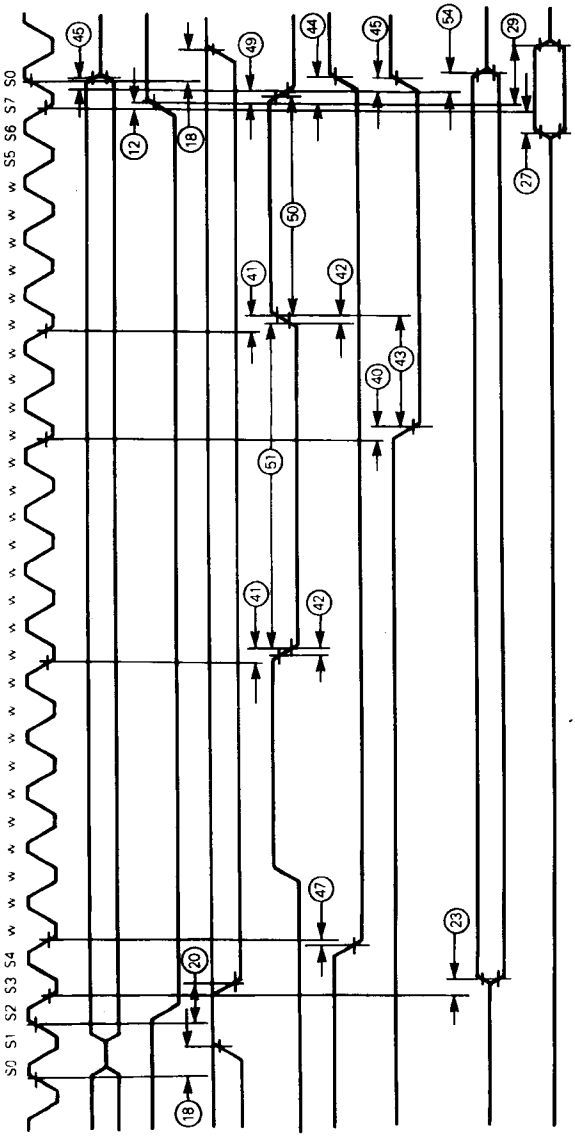
Notes : 1. For a loading capacitance of less than or equal to 50 pF, subtract 5 nanoseconds from the value given in the maximum columns.
2. If the asynchronous setup time (47) requirements are satisfied, the DTACK low-to-data setup time (31) requirement can be ignored. The data must only satisfy the date-in clock-low setupt time (27) for the following cycle.
3. The falling edge of S6 triggers both the negation of the strobes (AS, and x DS) and the falling edge of E. Either of these events can occur first, depending upon the loading on each signal. Specification 49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and the falling edge of the E clock.

Figure 8.9 : TS68000 to 6800 Peripheral Timing Diagram – Best Case.



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

Figure 8.10 : TS68000 to 6800 Peripheral Timing Diagram – Worst 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.

8.10. AC ELECTRICAL SPECIFICATIONS – BUS ARBITRATION

(V_{CC} = 5 Vdc ± 5 % ; GND = 0 Vdc ; $T_A = T_L$ to T_H ; see figures 8.11, 8.12 and 8.13)

N°	Symbol	Parameter	8 MHz		10 MHz		12.5 MHz		16 MHz		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
7	t_{CHADZ}	Clock high to Address, Data Bus High Impedance	–	70	–	60	–	50	–	50	ns
16	t_{CHCZ}	Clock High to Control Bus High Impedance	–	80	–	70	–	60	–	50	ns
33	t_{CHGL}	Clock High to $\overline{BG}$ Low	–	60	–	50	–	40	–	40	ns
34	t_{CHGH}	Clock High to $\overline{BG}$ High	–	60	–	50	–	40	–	40	ns
35	t_{BRLGL}	$\overline{BR}$, Low to $\overline{BG}$ Low	1.5	90ns + 3.5	1.5	80ns + 3.5	1.5	60ns + 3.5	1.5	60ns + 3.5	Clk Per.
36(1)	t_{BKHGH}	$\overline{BR}$, High to $\overline{BG}$ High	1.5	90ns + 3.5	1.5	80ns + 3.5	1.5	70ns + 3.5	1.5	60ns + 3.5	Clk Per.
37	T_{GALGH}	$\overline{BGACK}$ Low to $\overline{BG}$ High	1.5	90ns + 3.5	1.5	80ns + 3.5	1.5	70ns + 3.5	1.5	60ns + 3.5	Clk Per.
37A(2)	t_{GALBRH}	$\overline{BGACK}$ Low to $\overline{BG}$ High	20	1.5 Clocks	20	1.5 Clocks	20	1.5 Clocks	10	1.5 Clocks	ns
38	t_{GLZ}	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance (AS High)	–	80	–	70	–	60	–	50	ns
39	T_{GH}	$\overline{BG}$ Width High	1.5	–	1.5	–	1.5	–	1.5	–	Clk Per.
46	t_{GAL}	$\overline{BGACK}$ Width Low	1.5	–	1.5	–	1.5	–	1.5	–	Clk Per.
47(3)	t_{ASI}	Asynchronous Input Setup Time	20	–	20	–	10	–	10	–	ns
57	t_{GABD}	$\overline{BGACK}$ High to Control Bus Driven $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$	1.5	–	1.5	–	1.5	–	1.5	–	Clk Per.
		FC_x, R/$\overline{W}$, $\overline{VMA}$	1		1		1		1		
58(1)	t_{RHBD}	$\overline{BR}$ High to Control Bus Driven	1.5	–	1.5	–	1.5	–	1.5		Clk Per.

Notes :
1. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.
2. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, $\overline{BG}$ may be reasserted.
3. If the asynchronous setup time (47) requirements are satisfied, the $\overline{DTACK}$ low-to-data setup time (31) requirement can ban ign red. The data must only satisfy the date-in clock-low setup time (27) for the following cycle.

Figures 8.11, 8.12, and 8.13 depict the three bus arbitration cases that can arise. Figure 8.11 shows the timing where $\overline{AS}$ is negated when the processor asserts BG (Idle Bus Case). Figure 8.12 shows the timing where $\overline{AS}$ is asserted when the processor asserts $\overline{BG}$ (Active Bus Case). Figure 8.13 shows the timing where more than one bus master are requesting the bus. Refer to **4.2.2. Bus Arbitration** for a complete discussion of bus arbitration.

The waveforms shown in figures 8.11, 8.12, and 8.13 should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

Figure 8.11 : Bus Arbitration Timing Diagram – Idle Bus Case.

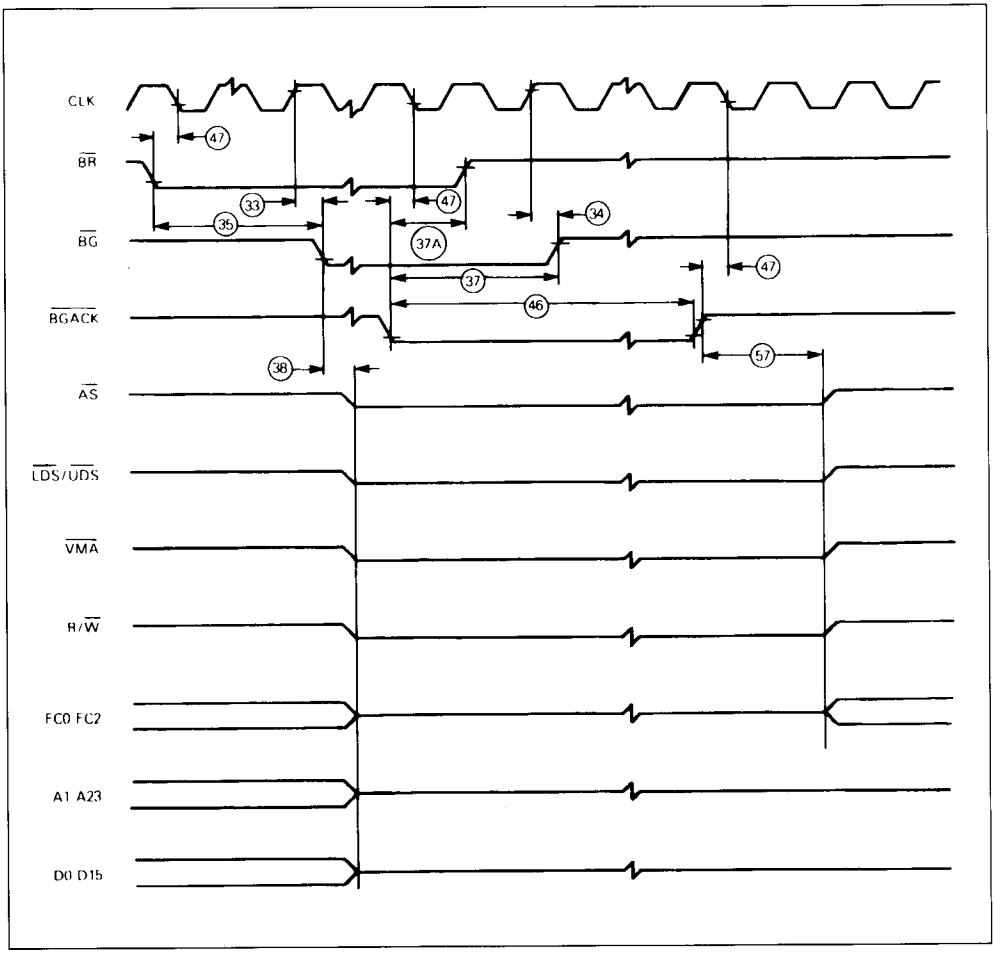


Figure 8.12 : Bus Arbitration Timing Diagram – Active Bus Case.

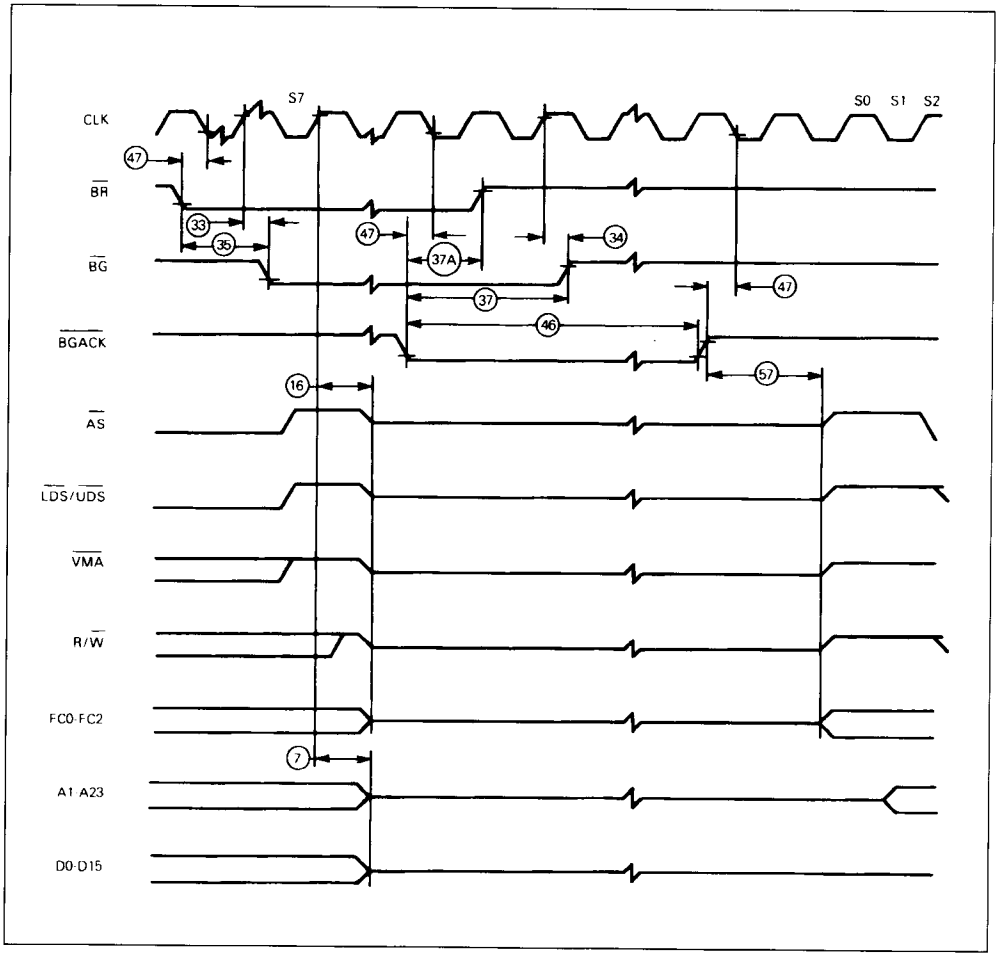
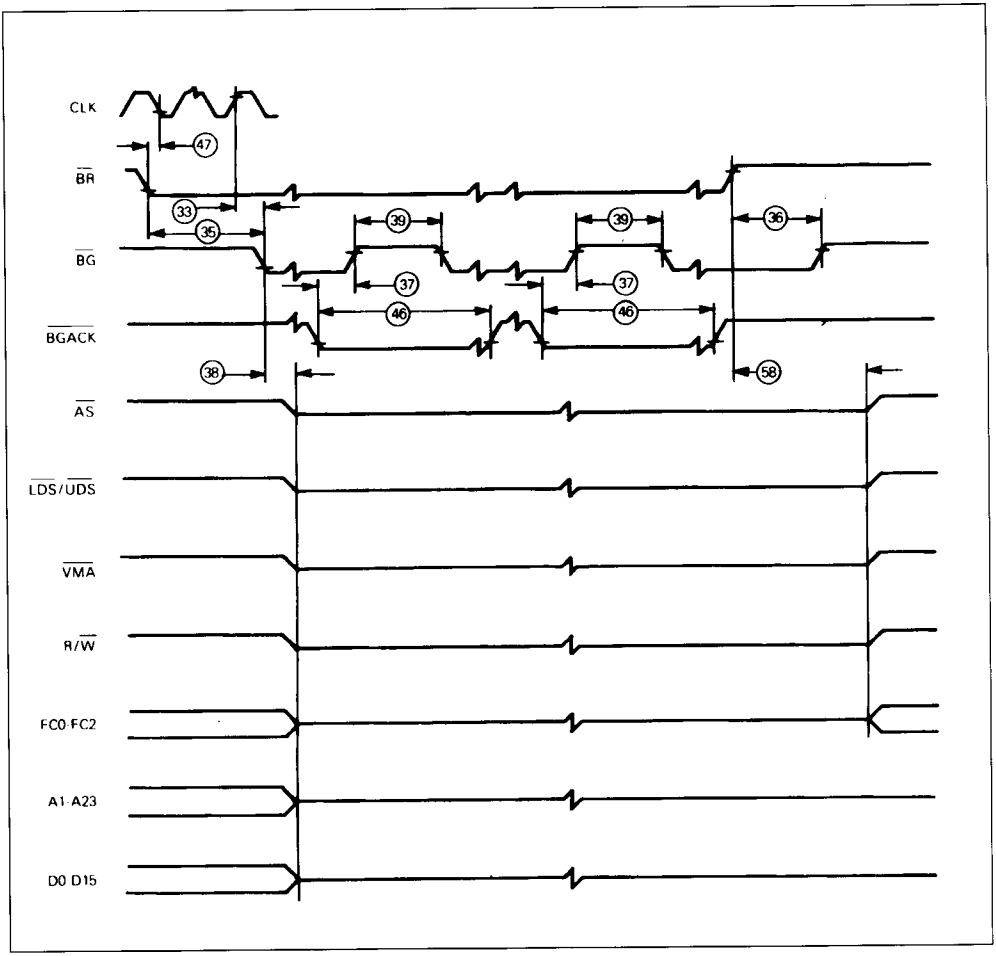


Figure 8.13 : Bus Arbitration Timing Diagram – Multiple Bus Requests.



SECTION 9

ORDER CODES

This section contains detailed information to be used as a guide when ordering the TS68000

9.1. STANDARD VERSIONS

Part Number	Frequency (MHz)	Temperature Range	Package Type
TS68000 CP8	8.0	0°C to +70°C	Plastic DIL
TS68000 VP8	8.0	−40°C to +85°C	P Suffix
TS68000 CP10	10.0	0°C to +70°C	
TS68000 VP10	10.0	−40°C to +85°C	
TS68000 CP12	12.5	0°C to +70°C	
TS68000 CP16	16.0	0°C to +70°C	
TS68000 CFN8	8.0	0°C to +70°C	PLCC
TS68000 VFN8	8.0	−40°C to +85°C	FN Suffix
TS68000 CFN10	10.0	0°C to +70°C	
TS68000 VFN10	10.0	−40°C to +85°C	
TS68000 CFN12	12.5	0°C to +70°C	
TS68000 CFN16	16.0	0°C to +70°C	
TS68000 CR8	8.0	0°C to +70°C	Pin Grid Array
TS68000 CR10	10.0	0°C to +70°C	R Suffix
TS68000 CR12	12.5	0°C to +70°C	
TS68000 CR16	16.0	0°C to +70°C	

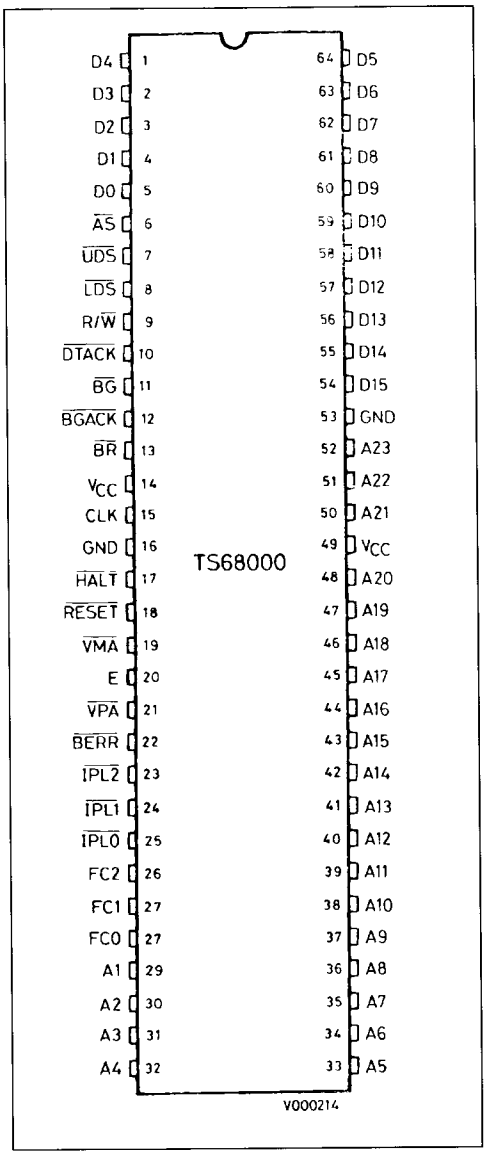
SECTION 10

MECHANICAL DATA

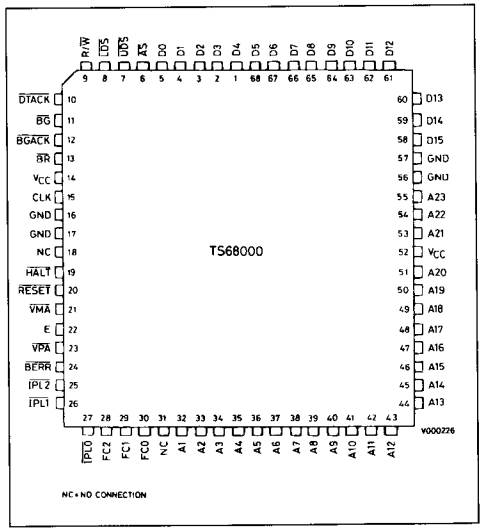
This section contains the pin assignments and package dimensions for the 64–pin dual–in–line, the 68–terminal chip carrier (LCCC), the 68–pin grid array, and the 68–pin quad pack (PLCC), versions of the TS68000.

10.1. PIN ASSIGNMENTS

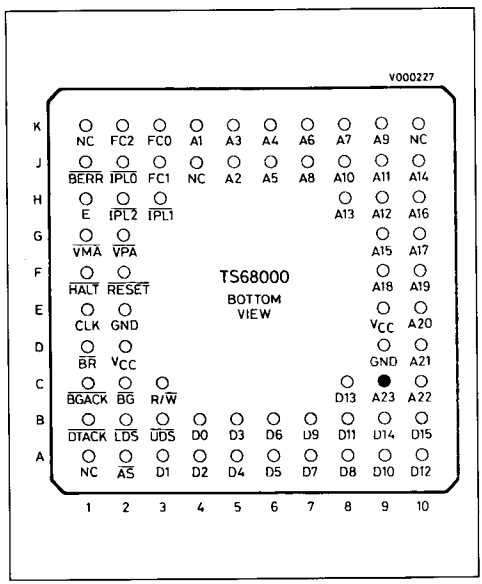
64–Pin Dual–in–Line Package



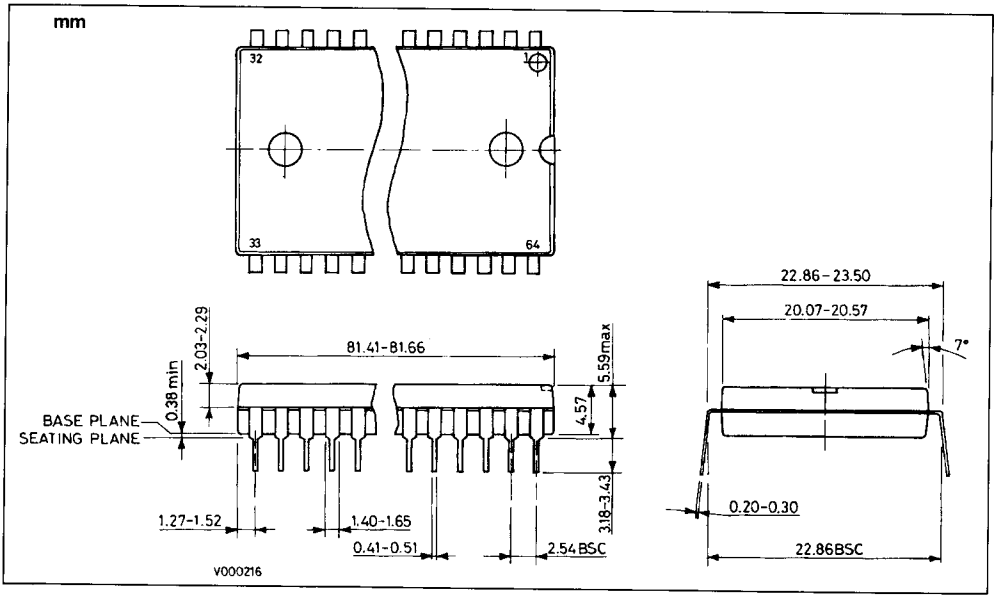
68–Pin Quad Pack (PLCC)

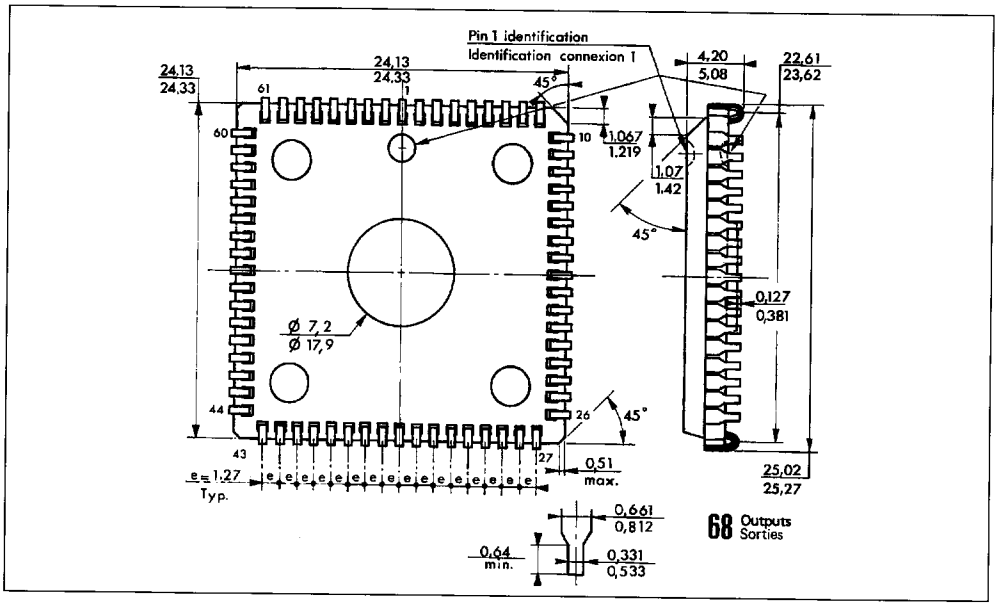


68–Pin Grid Array

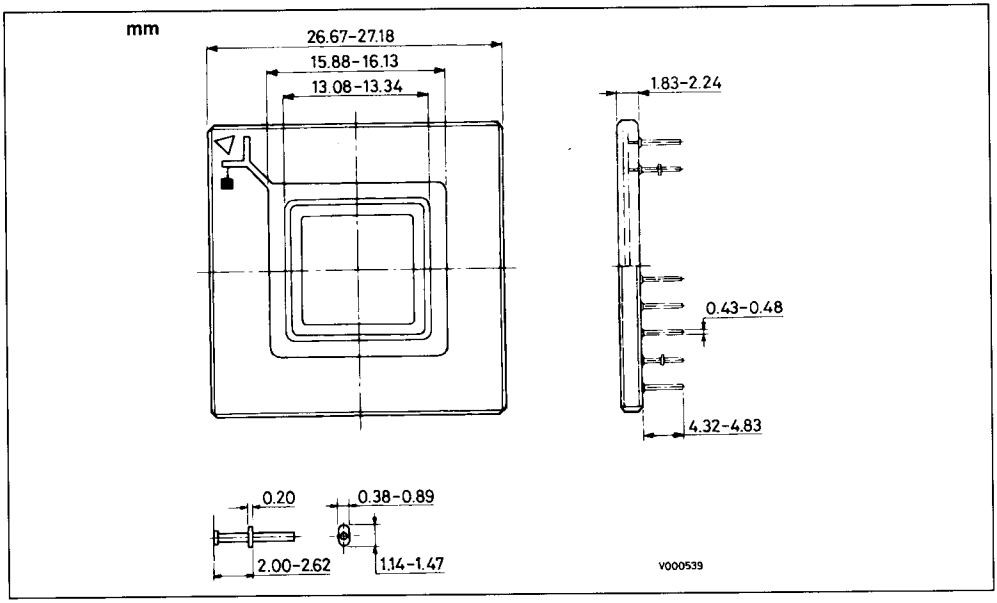


10.2. PACKAGE DIMENSIONS





10.2. PACKAGE DIMENSIONS (continued)



This is advanced information and specifications are subject to change without notice. Please inquire with our sales offices about the availability to the different packages.